ФОРТРАН 90. МЕЖДУНАРОДНЫЙ СТАНДАРТ

Книга содержит перевод на русский язык официального описания международного стандарта языка Фортран 90, который значительно расширяет возможности своих предшественников, сохранив при этом полную с ними преемственность. Наиболее важные его новые возможности: структурные типы данных; операции над массивами и секциями массивов; средства задания точности вычислений; указатели; механизмы динамического размещения объектов в памяти; новые средства для описания и использования глобальных объектов; гибкие управляющие конструкции; определяемые пользователем операции; элементы объектно-ориентированного программирования и др.

Для прикладных и системных программистов, работающих с современным Фортраном.

Содержание

Предисловие редактора перевода XXIII
Предисловие к русскому переводу XXVII
Предисловие XXVIII
ВВЕДЕНИЕ XXIX
Стандартный язык программирования Фортран XXIX
ОБЩИЕ СВЕДЕНИЯ XXIX
Операции над массивами XXIX
Числовые вычисления XXX
Параметризованный символьный тип данных XXX
Производные типы XXX
Модульные определения XXX
Указатели XXX
Эволюция языка XXXI
ОРГАНИЗАЦИЯ ДАННОГО МЕЖДУНАРОДНОГО СТАНДАРТА XXXI
Понятия верхнего/нижнего уровней XXXI
Концепция данных XXXI
Вычисления XXXII
Управление исполнением XXXII
Ввод/вывод XXXII
Программные единицы XXXIII
Область действия и правила связывания XXXIII
Глава 1 ОБЩИЕ СВЕДЕНИЯ 1
1.1. Цель. 1
1.2. Процессор 1
1.3. Что включено в стандарт и что не включено 1
1.3.1. Что включено 1
1.3.2. Что не включено 1
1.4. Соответствие стандарту 2
1.4.1. Совместимость с ФОРТРАНОМ 77 3
1.5. Условные обозначения 3
1.5.1. Синтаксические правила 4

1.5.2. Подразумеваемые синтаксические правила 5
1.5.3. Синтаксические соглашения и характеристики 5
1.5.4. Текстовые обозначения 5
1.6. Исключенные и устаревшие средства 5
1.6.1. Природа исключенных средств 5
1.6.2. Природа устаревших средств 6
1.7. Модули 6
1.8. Нормативные ссылки 6
Глава 2 ТЕРМИНЫ И ПОНЯТИЯ ФОРТРАНА 7
2.1. Синтаксис верхнего уровня 7
2.2. Понятия программной единицы 9
2.2.1. Выполняемая программа 9
2.2.2. Главная программная единица 9
2.2.3. Процедура 10
2.2.3.1. Внешняя процедура 10
2.2.3.2. Модульная процедура 10
2.2.3.3. Внутренняя процедура 10
2.2.3.4. Блок интерфейса процедуры 10
2.2.4. Модуль 10
2.3. Концепции выполнения 11
2.3.1. Выполняемые/невыполняемые операторы 11
2.3.2. Порядок операторов 11
2.3.3. Оператор END 12
2.3.4 Последовательность выполнения 12
2.4. Концепции данных: 13
2.4.1. Тип данных 13
2.4.1.1. Встроенный тип 13
2.4.1.2. Производный тип 13
2.4.2. Значения данных 14
2.4.3. Данное. 14
2.4.3.1. Объект данных 14
2.4.3.2. Выражение 15
2.4.3.3. Ссылка на функцию 15
2.4.4. Скаляр 15
2.4.5. Массив 15
2.4.6. Указатель 16
2.4.7. Память. 16
2.5. Основные термины 16
2.5.1. Имя и обозначение 16
2.5.2. Ключевое слово 16
2.5.3 Объявление 16
2.5.4. Определение и описание 17
2.5.5. Ссылка 17
2.5.6. Связь. 17

2.5.7. Встроенный 17
2.5.8. Знак операции 17
2.5.9 Последовательность 18
Глава 3 СИМВОЛЫ, ЛЕКСИЧЕСКИЕ ЭЛЕМЕНТЫ И ФОРМА ИСХОДНОГО ТЕКСТА 19
3.1. Набор символов процессора 19
3.1.1. Буквы 19
3.1.2. Цифры 19
3.1.3. Символ подчеркивания 19
3.1.4 Специальные символы 20
3.1.5. Другие символы 20
3.2 Синтаксис нижнего уровня 20
3.2.1. Ключевые слова 20
3.2.2. Имена 21
3.2 3 Константы 21
3.2.4. Знаки операций 21
3.2.5 Метки операторов 22
3.2.6. Ограничители 22
3.3. Форма исходного текста 22
3.3.1. Свободная форма 23
3.3.1.1. Комментарий в свободной форме 23
3.3.1.2. Разделение операторов в свободной форме 24
3.3.1.3. Продолжение оператора в свободной форме 24
3.3.1.4. Операторы в свободной форме 24
3.3.2. Фиксированная форма 24
3.3.2.1. Комментарии в фиксированной форме 25
3.3.2.2. Разделение операторов в фиксированной форме 25
3.3.2.3. Продолжение оператора в фиксированной форме 25
3.3.2.4. Операторы в фиксированной форме 25
3.4. Включение исходного текста 25
Глава 4 ВСТРОЕННЫЕ И ПРОИЗВОДНЫЕ ТИПЫ ДАННЫХ 27
4.1. Концепция типа данных 27
4.1.1. Множество значений 27
4.1.2. Константы 28
4.1.3. Операции 28
4.2. Взаимосвязь типов и значений объектов 28
4.3. Встроенные типы данных 29
4.3.1. Числовые типы 29
4.3.1.1. Тип целый 29
4.3.1.2. Вещественный тип 30
4.3.1.3. Комплексный тип 31
4.3.2 Нечисловые типы 32
4.3.2.1. Символьный тип 32
4.3.2.2. Логический тип 34

4.4. Производные типы 35
4.4.1. Описание производного типа 35
4.4.2. Определение производных типов 38
4.4.3. Значения производного типа 40
4.4.4. Конструирование значений производного типа 40
4.4.5. Операции и присваивание для производного типа 41
4.5. Конструирование значений массива 41
Глава 5 ОБЪЯВЛЕНИЯ И СПЕЦИФИКАЦИИ ОБЪЕКТОВ ДАННЫХ 43
5.1. Операторы объявления типа 43
5.1.1. Спецификаторы типа 45
5.1.1.1. INTEGER 45
5.1.1.2. REAL 45
5.1.1.3 DOUBLE PRECISION 45
5.1.1.4. COMPLEX 46
5.1.1.5. CHARACTER 46
5.1.1.6. LOGICAL 47
5.1.1.7. Производный тип 47
5.1.2. Атрибуты 48
5.1.2.1. Атрибут PARAMETER 48
5.1.2.2. Атрибут доступности 48
5.1.2.3. Атрибут INTENT 48
5.1.2.4. Атрибут DIMENSION 49
5.1.2.5. Атрибут SAVE 52
5.1.2.6. Атрибут OPTIONAL 52
5.1.2.7. Атрибут POINTER 52
5.1.2.8. Атрибут TARGET 53
5.1.2.9. Атрибут ALLOCATABLE 53
5.1.2.10. Атрибут EXTERNAL 53
5.1.2.11. Атрибут INTRINSIC 53
5.2. Операторы спецификации атрибутов 53
5.2.1. Оператор INTENT 53
5.2.2. Оператор OPTIONAL 54
5.2.3. Операторы PRIVATE и PUBLIC 54
5.2.4. Оператор SAVE 55
5.2.5 Оператор DIMENSION 55
5.2.6. Оператор ALLOCATABLE 55
5.2.7. Оператор POINTER 56
5.2.8. Оператор TARGET 56
5.2.9. Оператор DATA 56
5.2.10. Оператор PARAMETER 58
5.3. Оператор IMPLICIT 59
5.4. Оператор NAMELIST 61
5.5. Связь объектов данных по памяти 61

5.5.1. Оператор EQUIVALENCE 61
5.5.1.1. Связь по эквивалентности 62
5.5.1.2. Эквивалентность объектов стандартного символьного типа 62
5.5.1.3. Имена массивов и обозначение элементов массива 63
5.5.1.4. Ограничения на оператор EQUIVALENCE 63
5.5.2. Оператор COMMON 63
5.5.2.1. Последовательность памяти общего блока 64
5.5.2.2. Размер общего блока 65
5.5.2.3. Связь по общему блоку 65
5.5.2.4. Разница между именованным и неименованным общими блоками 65
5.5.2.5. Ограничения на COMMON и EQUIVALENCE 66
Глава 6 ИСПОЛЬЗОВАНИЕ ОБЪЕКТОВ ДАННЫХ 67
6.1. Скаляры 68
6.1.1. Подстроки 68
6.1.2. Компоненты структур 68
6.2. Массивы 69
6.2.1. Полный массив 69
6.2.2. Элементы и секции массива 70
6.2.2.1. Элементы массива 70
6.2.2.2. Последовательность элементов массива 70
6.2.2.3. Секции массива 71
6.3. Динамическая связь 73
6.3.1. Оператор ALLOCATE 73
6.3.1.1. Размещение динамически размещаемых массивов 74
6.3.1.2. Размещение адресатов указателей 74
6.3.2. Оператор NULLIFY 74
6.3.3. Оператор DEALLOCATE 74
6.3.3.1. Удаление из памяти динамически размещаемых массивов 75
6.3.3.2. Удаление из памяти адресатов указателей 75
Глава 7 ВЫРАЖЕНИЯ И ПРИСВАИВАНИЕ 77
7.1. Выражения 77
7.1.1. Форма выражения 77
7.1.1.1. Первичное 77
7.1.1.2. Выражения уровня-1 78
7.1.1.3. Выражения уровня-2 78
7.1.1.4. Выражения уровня-3 79
7.1.1.5. Выражения уровня-4 79
7.1.1.6. Выражения уровня-5 79
7.1.1.7. Общая форма выражения 80
7.1.2. Встроенные операции 80
7.1.3. Определяемые операции 81

7.1.4. Тип, параметры типа и конфигурация выражения 82
7.1.4.1. Тип, параметры типа и конфигурация первичного 82
7.1.4.2. Тип, параметры типа и конфигурация результата операции 83
7.1.5. Правила согласования для встроенных операций 83
7.1.6. Выражения-скаляры и выражения-массивы 84
7.1.6.1. Константное выражение 84
7.1.6.2. Выражение спецификации 85
7.1.7. Выполнение операций 87
7.1.7.1. Вычисление операндов 88
7.1.7.2. Целостность скобок 88
7.1.7.3. Выполнение числовых встроенных операций 88
7.1.7.4. Выполнение встроенной символьной операции 89
7.1.7.5. Выполнение встроенных операций отношения 90
7.1.7.6. Выполнение логических встроенных операций 90
7.1.7.7. Выполнение определяемых операций 90
7.2. Интерпретация встроенных операций 90
7.2.1. Числовые встроенные операции 90
7.2.1.1. Целочисленное деление 91
7.2.1.2. Комплексное возведение в степень 91
7.2.2 Символьная встроенная операция 91
7.2.3. Встроенные операции отношения 92
7.2.4. Логические встроенные операции 94
7.3. Интерпретация определяемых операций 94
7.3.1. Унарная определяемая операция 94
7.3.2. Бинарная определяемая операция 95
7.4. Старшинство операций 95
7.5. Присваивание. 97
7.5.1. Оператор присваивания 97
7.5.1.1. Общая форма 97
7.5.1.2. Встроенный оператор присваивания 97
7.5.1.3. Определяемый оператор присваивания 98
7.5.1.4. Правила согласования для встроенного присваивания 98
7.5.1.5. Интерпретация встроенного присваивания 99
7.5.1.6. Интерпретация определяемых операторов присваивания 100
7.5.2. Оператор присваивания указателю 100
7.5.3. Присваивание массивам по маске - WHERE 101
7.5.3.1. Общая форма присваивания массивам по маске 101
7.5.3.2. Интерпретация присваивания массиву по маске 102
Глава 8 УПРАВЛЕНИЕ ВЫПОЛНЕНИЕМ 103
8.1. Выполняемые конструкции 103
8.1.1. Правила управления блоками 103
8.1.1.1. Выполняемые конструкции в блоках 103
8.1.1.2. Передача управления в блоках 103

8.1.1.3. Выполнение блока 104
8.1.2. Конструкция IF 104
8.1.2.1. Форма конструкции IF 104
8.1.2.2. Выполнение конструкции IF 104
8.1.2.3. Примеры конструкции IF 105
8.1.2.4. Оператор IF 105
8.1.3. Конструкция CASE 105
8.1.3.1. Форма конструкции CASE 105
8.1.3.2. Выполнение конструкции CASE 106
8.1.3.3. Примеры конструкции CASE 107
8.1.4 Конструкция DO 108
8.1.4.1. Формы конструкции DO 108
8.1.4.2. Область конструкции DO 110
8.1.4.3. Активные и неактивные конструкции DO 110
8.1.4.4. Выполнение конструкции DO 110
8.1.4.5. Примеры конструкции DO 112
8.2. Операторы перехода (операторы ветвления) 115
8.2.1. Метки операторов 115
8.2.2. Оператор GO TO 115
8.2.3. Вычисляемый оператор GO TO 115
8.2.4. Оператор ASSIGN и оператор GO TO по предписанию 115
8.2.5. Арифметический оператор IF 116
8.3. Оператор CONTINUE 116
8.4 Оператор STOP 116
8.5. Оператор PAUSE 116
Глава 9 ОПЕРАТОРЫ ВВОДА/ВЫВОДА 117
9.1. Записи 117
9.1.1. Форматная запись 117
9.1.2. Неформатная запись 117
9.1.3. Запись "конец файла" 117
9.2. Файлы 118
9.2 1. Внешние файлы 118
9.2.1.1. Существование файла 118
9.2.1.2. Доступ к файлу 118
9.2.1.3. Позиция файла 119
9.2.2. Внутренние файлы 121
9.2.2.1. Свойства внутренних файлов 121
9.2.2.2. Ограничения на внутренние файлы 121
9.3. Подсоединение файла 121
9.3.1. Существование устройства 122
9.3.2 Подсоединение файла к устройству 122
9.3.4. Предварительное подсоединение 123
9.3.4. Оператор OPEN 123
9.3.4.1. Спецификатор FILE= в операторе OPEN 124

9.3.4.2. Спецификатор STATUS= в операторе OPEN 124
9.3.4.3. Спецификатор ACCESS= в операторе OPEN 125
9.3.4.4. Спецификатор FORM= в операторе OPEN 125
9.3.4.5. Спецификатор RECL= в операторе OPEN 125
9.3.4.6. Спецификатор BLANK= в операторе OPEN 125
9.3.4.7. Спецификатор POSITION= в операторе OPEN 125
9.3.4.8 Спецификатор ACTION= в операторе OPEN 126
9.3.4.9. Спецификатор DELIM= в операторе OPEN 126
9.3.4.10. Спецификатор PAD= в операторе OPEN 126
9.3.5. Оператор CLOSE 126
9.3.5.1. Спецификатор STATUS= в операторе CLOSE 127
9.4. Операторы передачи данных 127
9.4.1. Список управляющей информации 128
9.4.1.1. Спецификатор формата 129
9.4.1.2. Спецификатор NAMELIST 130
9.4.1.3. Номер записи 130
9.4.1.4. Статус ввода/вывода 130
9.4.1.5. Ветвь ошибки 131
9.4.1.6. Ветвь "конец файла" 131
9.4.1.7. Ветвь "конец записи" 131
9.4.1.8. Спецификатор продвижения 132
9.4.1.9. Число символов 132
9.4.2 Список ввода/вывода для передачи данных 132
9.4.3. Ситуации ошибки, "конец записи" и "конец файла" 133
9.4.4 Выполнение операторов ввода/вывода для передачи данных 134
9.4.4.1. Направление передачи данных 134
9.4.4.2. Идентификация устройства 134
9.4.4.3 Установление формата 135
9.4.4.4. Передача данных 135
9.4.4.5. Форматирование, управляемое списком 137
9.4.4.6. Форматирование по NAMELIST 137
9.4.5 Печать форматных записей 137
9.4.6. Завершение операторов передачи данных 137
9.5. Операторы установки позиции файла 138
9.5.1. Оператор BACKSPACE 138
9.5.2. Оператор ENDFILE 138
9.5.3 Оператор REWIND 139
9.6. Запрос о файле 139
9.6.1. Спецификаторы запроса 139
9.6.1.1 Спецификатор FILE= в операторе INQUIRE 140
9.6.1.2. Спецификатор EXIST= в операторе INQUIRE 140
9.6.1.3. Спецификатор OPENED= в операторе INQUIRE 140
9.6.1.4. Спецификатор NUMBER= в операторе INQUIRE 141
9.6.1.5. Спецификатор NAMED= в операторе INQUIRE 141

9.6.1.6. Спецификатор NAME= в операторе INQUIRE 141
9.6.1.7. Спецификатор ACCESS= в операторе INQUIRE 141
9.6.1.8. Спецификатор SEQUENTIAL= в операторе INQUIRE 141
9.6.1.9. Спецификатор DIRECT= в операторе INQUIRE 141
9.6.1.10. Спецификатор FORM= в операторе INQUIRE 141
9.6.1.11. Спецификатор FORMATTED= в операторе INQUIRE 141
9.6.1.12. Спецификатор UNFORMATTED= в операторе INQUIRE 142
9.6.1.13. Спецификатор RECL= в операторе INQUIRE 142
9.6.1.14. Спецификатор NEXTREC= в операторе INQUIRE 142
9.6.1.15. Спецификатор BLANK= в операторе INQUIRE 142
9.6.1.16. Спецификатор POSITION в операторе INQUIRE 142
9.6.1.17. Спецификатор ACTION= в операторе INQUIRE 142
9.6.1.18. Спецификатор READ= в операторе INQUIRE 143
9.6.1.19. Спецификатор WRITE= в операторе INQUIRE 143
9.6.1.20. Спецификатор READWRITE= в операторе INQUIRE 143
9.6.1.21. Спецификатор DELIM= в операторе INQUIRE 143
9.6.1.22. Спецификатор PAD= в операторе INQUIRE 143
9.6.2. Ограничения на спецификаторы запроса 143
9.6.3. Запрос о списке вывода 143
9.7. Ограничения для ссылок на функции и для элементов списка 144
9.8. Ограничения на операторы ввода/вывода 144
Глава 10 РЕДАКТИРОВАНИЕ ВВОДА/ВЫВОДА 145
10.1. Методы явной спецификации формата 145
10.1.1. Оператор FORMAT 145
10.1.2. Символьная спецификация формата 145
10.2. Форма списка элементов формата 146
10.2.1. Описатели редактирования 146
10.2.2. Поля 147
10.3. Взаимодействие между списком ввода/вывода и форматом 147
10.4. Установка позиции файла при управлении форматом 148
10.5 Описатели редактирования данных 149
10.5.1. Описатели редактирования чисел 149
10.5.1.1. Редактирование целых данных 149
10.5.1.2. Описатели редактирования вещественных и комплексных данных 150
10.5.2. Редактирование логических данных 153
10.5.3. Редактирование символьных данных 153
10.5.4. Обобщенные описатели редактирования 154
10.5.4.1. Обобщенные описатели редактирования чисел 154
10.5.4.2. Обобщенные описатели редактирования данных логического типа 155
10.5.4.3. Обобщенные описатели редактирования данных символьного типа 155

10.6. Управляющие описатели редактирования 155
10.6.1. Описатели редактирования позиции 155
10.6.1.1. Описатели редактирования T, TL и TR 156
10.6.1.2. Описатель редактирования X 156
10.6.2. Описатель редактирования "дробная черта" 156
10.6.3 Описатель редактирования "двоеточие" 157
10.6.4. Описатели редактирования S, SP и SS 157
10.6.5. Описатель редактирования P 157
10.6.5.1. Масштабный множитель 157
10.6.6. Описатели редактирования BN и BZ 158
10.7. Описатели редактирования "символьная строка" 158
10.7.1. Описатель редактирования "символьная константа" 158
10.7.2. Описатель редактирования H 158
10.8. Форматирование, управляемое списком 158
10.8.1. Ввод, управляемый списком 159
10.8.1.1. Пустые значения 160
10.8.1.2. Пример ввода, управляемого списком 160
10.8.2. Вывод, управляемый списком 161
10.9. Форматирование по NAMELIST 162
10.9.1. Ввод по NAMELIST 162
10.9.1.1. Имена объектов группы NAMELIST 162
10.9.1.2. Значения данных, вводимых по NAMELIST 163
10.9.1.3. Элементы списка объектов группы NAMELIST 163
10.9.1.4. Пустые значения 164
10.9.1.5. Пробелы 164
10.9.1.6. Пример ввода по NAMELIST 165
10.9.2 Вывод по NAMELIST 165
10.9.2.1. Редактирование вывода по NAMELIST 165
10.9.2.2. Записи вывода по NAMELIST 166
Глава 11 ПРОГРАММНЫЕ ЕДИНИЦЫ 167
11.1. Главная программная единица 167
11.1.1. Спецификации главной программной единицы 167
11.1.2 Раздел выполняемых конструкций главной программной единицы 167
11.1.3. Внутренние процедуры главной программной единицы 168
11.2. Внешние процедуры 168
11.3. Модули 168
11.3.1. Ссылка на модуль 168
11.3.2. Оператор USE и связь по использованию 169
11.3.3. Примеры использования модулей 170
11.3.3.1. Идентичные общие блоки 170
11.3.3.2. Глобальные данные 170
11.3.3.3. Производные типы 171
11.3.3.4. Глобальные динамически размещаемые массивы 171

11.3.3.5. Библиотеки процедур 171
11.3.3.6. Расширение семантики знаков операций 172
11.3.3.7. Абстракция данных 172
11.3.3.8. Переименованные общедоступные объекты 172
11.4. Программные единицы - блоки данных 173
Глава 12. ПРОЦЕДУРЫ 175
12.1. Классификации процедур 175
12.1.1. Классификация процедур по ссылке 175
12.1.2. Классификация процедур по способу описания 175
12.1.2.1. Встроенные процедуры 175
12.1.2.2. Внешние, внутренние и модульные процедуры 175
12.1.2.3. Формальные процедуры 177
12.1.2.4. Операторные функции 177
12.2. Характеристики процедур 178
12.2.1. Характеристики формальных аргументов 178
12.2.1.1. Характеристики формальных объектов данных 178
12.2.1.2. Характеристики формальных процедур 178
12.2.1.3. Характеристики формального аргумента - звездочки 178
12.2.2 Характеристики результатов функции 178
12.3. Интерфейс процедуры 178
12.3.1. Явный и неявный интерфейсы 178
12.3.1.1. Явный интерфейс 179
12.3.2. Спецификация интерфейса процедуры 179
12.3.2.1. Блок интерфейса процедуры 179
12.3.2.2. Оператор EXTERNAL 183
12.3.2.3. Оператор INTRINSIC 183
12.3.2.4. Спецификация неявного интерфейса 183
12.4. Ссылка на процедуру 184
12.4.1. Список фактических аргументов 184
12.4.1.1. Аргументы, связанные с формальными объектами данных 185
12.4.1.2. Аргументы, связанные с формальными процедурами 186
12.4.1.3. Аргументы, связанные с указателями альтернативного возврата 186
12.4.1.4. Связь по последовательности 186
12.4.2 Ссылка на функцию 187
12.4.3. Поэлементная ссылка на встроенную функцию 187
12.4.4. Ссылка на подпрограмму 187
12.4.5. Поэлементная ссылка на встроенную подпрограмму 187
12.5. Определение процедуры 188
12.5.1. Определение встроенной процедуры 188
12.5.2 Процедуры, определяемые средствами Фортрана 188
12.5.2.1. Действие атрибута INTENT в процедурах 188
12.5.2.2. Процедура-функция 188

12.5.2.3. Подпрограмма 190
12.5.2.4. Экземпляры процедуры 190
12.5.2.5. Оператор ENTRY 190
12.5.2.6. Оператор RETURN 192
12.5.2.7. Оператор CONTAINS 192
12.5.2.8. Ограничения на отсутствующие формальные аргументы 192
12.5.2.9. Ограничения на объекты, связанные с формальными аргументами 193
12.5.3. Определение процедур способом, отличным от Фортрана 194
12.5.4. Операторная функция 195
12.4. Ссылка на процедуру 184
12.4.1. Список фактических аргументов 184
12.4.1.1. Аргументы, связанные с формальными объектами данных 185
12.4.1.2. Аргументы, связанные с формальными процедурами 186
12.4.1.3. Аргументы, связанные с указателями альтернативного возврата 186
12.4.1.4. Связь по последовательности 186
Глава 13. ВСТРОЕННЫЕ ПРОЦЕДУРЫ 197
13.1. Встроенные функции 197
13.2. Поэлементные встроенные процедуры 197
13.2.1. Аргументы и результаты поэлементной встроенной функции 197
13.2.2. Аргументы встроенной поэлементной подпрограммы 197
13.3. Позиционные аргументы и аргументы с ключевыми словами 198
13.4. Функция запроса о наличии аргументов 198
13.5. Числовые, математические, символьные, логические, битовые процедуры и процедуры для параметра вида 198
13.5.1. Числовые функции 198
13.5.2. Математические функции 198
13.5.3. Символьные функции 198
13.5.4. Символьная справочная функция 198
13.5.5. Функции для параметра вида 198
13.5.6 Логическая функция 198
13.5.7. Процедуры обработки и справочные функции для битовых данных 199
13.6. Функция передачи типа 199
13.7. Функции обработки и справочные функции для числовых данных 199
13.7.1. Модели для целых и вещественных данных 199
13.7.2. Числовые справочные функции 200
13.7.3. Функции обработки данных с плавающей запятой 200
13.8. Встроенные функции для массивов 200
13.8.1. Конфигурация массивов-аргументов 200

13.8.2. Аргументы-маски 201
13.8.3. Функции векторного и матричного умножения 201
13.8.4. Функции редукции 201
13.8.5. Справочные функции для массивов 201
13.8.6. Функции конструирования массивов 201
13.8.7. Функция изменения конфигурации массива 202
13.8.8. Функции обработки массивов 202
13.8.9. Функции определения расположения в массиве 202
13.8.10. Справочная функция для запроса о статусе связанности указателя 202
13.9. Встроенные подпрограммы 202
13.9.1. Подпрограммы даты и времени 202
13.9.2. Псевдослучайные числа 202
13.9.3. Подпрограмма копирования битов 202
13.10. Обобщенные встроенные функции 203
13.10.1. Справочная функция для запроса о наличии аргументов 203
13.10.2. Числовые функции 203
13.10.3. Математические функции 204
13.10.4. Символьные функции 204
13.10.5. Символьная справочная функция 204
13.10.6. Справочные функции вида 204
13.10.7. Логическая функция 205
13.10.8. Числовые справочные функции 205
13.10.9. Битовая справочная функция 205
13.10.10. Функции для операций с битами 205
13.10.11. Функция передачи типа 205
13.10.12. Функции для операций с плавающей точкой 205
13.10.13. Функции векторного и матричного умножения 206
13.10.14. Функции редукции 206
13.10.15. Функции запроса о массиве 206
13.10.16. Функции конструирования массива 206
13.10.17. Функция изменения конфигурации массива 206
13.10.18. Функции манипуляции с массивами 206
13.10.19. Функции определения положения в массиве 207
13.10.20. Функция запроса о статусе связанности указателя 207
13.11. Встроенные подпрограммы 207
13.12. Специфические имена для встроенных функций 207
13.13. Спецификации встроенных процедур 209
13.13.1. ABS (A) 209
13.13.2. ACHAR (I) 209
13.13.3. ACOS (X) 210
13.13.4. ADJUSTL (STRING) 210
13.13.5. ADJUSTR (STRING) 210
13.13.6. AIMAG (Z) 210

13.13.7. AINT(A, KIND) 211
13.13.8. ALL (MASK, DIM) 211
13.13.9. ALLOCATED (ARRAY) 212
13.13.10. ANINT(A, KIND) 212
13.13.11. ANY (MASK, DIM) 212
13.13.12. ASIN(X) 213
13.13.13. ASSOCIATED (POINTER, TARGET) 213
13.13.14. ATAN(X) 214
13.13.15. ATAN2(Y, X) 214
13.13.16. BIT_SIZE (I) 214
13.13.17. BTEST(I, POS) 215
13.13.18. CEILING (A) 215
13.13.19. CHAR (I, KIND) 215
13.13.20. CMPLX (X, Y, KIND) 216
13.13.21. CONJG(Z) 216
13.13.22. COS (X) 216
13.13.23. COSH(X) 216
13.13.24. COUNT (MASK, DIM) 217
13.13.25. CSHIFT (ARRAY, SHIFT, DIM) 217
13.13.26. DATE_AND_TIME (DATE, TIME, ZONE, VALUES) 218
13.13.27. DBLE(A) 219
13.13.28. DIGITS (X) 219
13.13.29. DIM (X, Y) 220
13.13.30. DOT_PRODUCT (VECTOR_A, VECTOR_B) 220
13.13.31. DPROD(X.Y) 221
13.13.32. EOSHIFT (ARRAY, SHIFT, BOUNDARY, DIM) 221
13.13.33. EPSILON(X) 222
13.13.34. EXP (X) 222
13.13.35. EXPONENT (X) 223
13.13.36. FLOOR (A) 223
13.13.37. FRACTION (X) 223
13.13.38. HUGE(X). 223
13.13.39. IACHAR (C) 224
13.13.40. IAND (I, J) 224
13.13.41. IBCLR (I, POS) 224
13.13.42. IBITS (I, POS, LEN) 225
13.13.43. IBSET(I, POS) 225
13.13.44. ICHAR(C) 225
13.13.45. IEOR (I.J) 226
13.13.46. INDEX (STRING, SUBSTRING, BACK) 226
13.13.47. INT (A, KIND) 226
13.13.48. IOR (I, J) 227
13.13.49. ISHFT (I, SHIFT) 227
13.13.50. ISHFTC (I, SHIFT, SIZE) 228

13.13.51. KIND(X) 228
13.13.52. LBOUND (ARRAY, DIM) 228
13.13.53. LEN (STRING) 229
13.13.54. LEN_TRIM (STRING) 229
13.13.55. LGE(STRING_A, STRING_B) 229
13.13.56. LGT (STRING_A, STRING_B) 230
13.13.57. LLE (STRING_A, STRING_B) 230
13.13.58. LLT(STRING_A, STRING_B) 230
13.13.59. LOG (X) 231
13.13.60. LOG10(X) 231
13.13.61. LOGICAL (L, KIND) 231
13.13.62. MATMUL (MATRIX_A, MATRIX_B) 231
13.13.63. MAX(A1, A2, A3,...) 232
13.13.64. MAXEXPONENT(X) 233
13.13.65. MAXLOC (ARRAY, MASK) 233
13.13.66. MAXVAL(ARRAY, DIM, MASK) 234
13.13.67. MERGE (TSOURCE, FSOURCE, MASK) 235
13.13.68. MIN(Al, A2, A3,…) 235
13.13.69. MINEXPONENT(X) 235
13.13.70. MINLOC(ARRAY, MASK) 236
13.13.71. MINVAL (ARRAY, DIM, MASK) 236
13.13.72. MOD (A, P) 237
13.13.73. MODULO (A, P) 238
13.13.74. MVBITS (FROM, FROMPOS, LEN, TO, TOPOS) 238
13.13.75. NEAREST (X, S) 238
13.13.76. NINT(A, KIND) 239
13.13.77. NOT (I) 239
13.13.78. PACK (ARRAY, MASK, VECTOR) 239
13.13.79. PRECISION (X) 240
13.13.80. PRESENT (X) 240
13.13.81. PRODUCT (ARRAY, DIM, MASK) 240
13.13.82. RADIX (X) 241
13.13.83. RANDOM_NUMBER (HARVEST) 242
13.13.84. RANDOM_SEED (SIZE, PUT, GET) 242
13.13.85. RANGE (X) 242
13.13.86. REAL (A, KIND) 243
13.13.87. REPEAT (STRING, NCOPIES) 243
13.13.88. RESHAPE (SOURCE, SHAPE, PAD, ORDER) 243
13.13.89. RRSPACING (X) 244
13.13.90. SCALE (X, I) 245
13.13.91. SCAN (STRING, SET, BACK) 245
13.13.92. SELECTED_INT_KIND (R) 245
13.13.93. SELECTED_REAL_KIND (P, R) 246
13.13.94. SET_EXPONENT (X, I) 246

13.13.95. SHAPE (SOURCE) 247
13.13.96. SIGN (A, B) 247
13.13.97. SIN (X). 247
13.13.98. SINH (X) 247
13.13.99. SIZE (ARRAY, DIM) 248
13.13.100. SPACING (X) 248
13.13.101. SPREAD (SOURCE, DIM, NCOPIES) 248
13.13.102. SQRT(X) 249
13.13.103. SUM (ARRAY, DIM, MASK) 249
13.13.104. SYSTEM_CLOCK (COUNT, COUNT_RATE, COUNT_MAX) 250
13.13.105. TAN(X) 250
13.13.106. TANH(X) 251
13.13.107. TINY(X) 251
13.13.108. TRANSFER (SOURCE, MOLD, SIZE) 251
13.13.109. TRANSPOSE (MATRIX) 252
13.13.110. TRIM (STRING) 252
13.13.111. UBOUND (ARRAY, DIM) 252
13.13.112. UNPACK (VECTOR, MASK, FIELD) 253
13.13.113. VERIFY (STRING, SET, BACK) 254
Глава 14. ОБЛАСТЬ ДЕЙСТВИЯ, СВЯЗЬ И ОПРЕДЕЛЕННОСТЬ 255
14.1. Область действия имен 255
14.1.1. Глобальные объекты 255
14.1.2. Локальные объекты 255
14.1.2.1. Общие блоки 256
14.1.2.2. Результаты функций 256
14.1.2.3. Однозначные обобщенные ссылки на процедуры 256
14.1.2.4. Разрешение ссылок на процедуры 257
14.1.2.5 Компоненты 259
14.1.2.6. Ключевые слова аргументов 259
14.1.3. Операторные объекты 259
14.2. Область действия меток 260
14.3. Область действия внешних устройств ввода/вывода 260
14.4. Область действия знаков операций 260
14.5. Область действия символа присваивания 260
14.6. Связывание. 260
14.6.1. Связь по имени 260
14.6.1.1. Связь по аргументу 260
14.6.1.2. Связь по использованию и по объемлющему контексту 261
14.6.2. Связь по указателю 261
14.6.2.1. Статус связанности указателя 261
14.6.2.2. Статус определенности указателя 261
14.6.2.3. Взаимосвязь между статусом связанности и статусом определенности 261

14.6.3 Связь по памяти 262
14.6.3.1 Последовательность памяти 262
14.6.3.2. Связь последовательностей памяти 262
14.6.3.3. Связь скалярных объектов данных 263
14.7. Определенность и неопределенность переменных 264
14.7.1. Определение объектов и подобъектов 264
14.7.2. Переменные, которые всегда определены 264
14.7.3. Переменные, которые определены начально 264
14.7.4. Переменные, не определенные начально 265
14.7.5. События, которые приводят к определению объектов 265
14.7.6. События, приводящие к неопределенности 266
14.8. Статус размещенности 268
ПРИЛОЖЕНИЕ А. Словарь технических терминов 269
ПРИЛОЖЕНИЕ В. Устаревшие средства 281
В.1. Исключенные средства 281
В.2. Отживающие средства 281
В.2.1. Альтернативный возврат 281
В.2.2. Оператор PAUSE 282
В.2.3. Операторы ASSIGN и GO TO по предписанию 282
В.2.4. Присваиваемые спецификаторы формата 282
В.2.5. Описатель редактирования Н 282
ПРИЛОЖЕНИЕ С. Примечания к главам 283
С.1. Примечания к главе 1 283
С.1.1. Соответствие стандарту (1.4) 283
С.2. Примечания к главе 2 283
С.2.1. Ключевые слова 283
С.3 Примечания к главе 3 283
С.3.1. Представимые символы (3.1.5.) 283
С.3.2. Строки комментария (3.3.1.1, 3.3.2.1) 283
С.3.3. Метки оператора (3.2.5) 284
С.3.4. Форма исходного текста (3.3) 284
С.4. Примечания к главе 4 284
С.4.1. Нуль (4.3.1) 284
С.4.2. Символы (4.2) 284
С.4.3. Встроенные и производные типы данных (4.3, 4.4) 285
С.4.4 Выбор способов аппроксимации 285
С.4.5. Память для производных типов (4.4.1) 286
С.4.6. Указатели 286
С.5. Примечания к главе 5 287
С.5.1. Операторы объявления типа (5.1) 287
С.5.2. Атрибут POINTER (5.1.2.7) 288
С.5.3. Атрибут TARGET (5.1.2.8) 288
С.5.4. Операторы PARAMETER и IMPLICIT NONE (5.2.10, 5.3) 289
С.5.5. Расширения оператора EQUIVALENCE (5.5.1) 289

С.5.6. Расширения оператора COMMON (5.5.2) 289
С.6. Примечания к главе 6 290
С.6.1. Подстроки (6.1.1) 290
С.6.2. Ссылки на элементы массива (6.2.2) 290
С.6.3. Компоненты структуры (6.1.2) 290
С.6.4. Размещение и связывание указателей 291
С.7. Примечания к главе 7 292
С.7.1. Символьное присваивание 292
С.7.2. Вычисление ссылок на функции 292
С.7.3. Указатели в выражениях 292
С.7.4. Указатели в левой части присваивания 292
С.8. Примечания к главе 8 293
С.8.1. Управление циклом 293
С.8.2 Конструкция CASE 293
С.8.3 Примеры неверных конструкций DO 293
С.9. Примечания к главе 9 294
С.9.1. Записи ввода/вывода (9.1) 294
С.9.2. Файлы (9.2) 294
С.9.2.1. Подсоединение файла (9.3) 294
С.9.2.2. Существование файла (9.2.1.1) 294
С.9.2.3. Имена файлов (9.3.4.1) 295
С.9.2.4. Доступ к файлу (9.2.1.2) 295
С.9.2.5. Ввод/вывод без продвижения (9.2.1.3.1) 295
С.9.3. Оператор OPEN (9.3.4) 296
С.9.4. Характеристики подсоединения (9.3.2) 298
С.9.5. Оператор CLOSE (9.3.5) 299
С.9.6. Оператор INQUIRE 299
С.9.7. Спецификаторы с ключевыми словами 299
С.9.8. Спецификации формата (9.4.1.1) 299
С.9.9. Неформатный ввод/вывод (9.4.4.4.1) 299
С.9.10. Ограничения на ввод/вывод 300
С.9.11. Указатели в списке ввода/вывода 300
С.9.12. Объекты производного типа в списке ввода/вывода (9.4.2) 301
С.10. Примечания к главе 10 301
С.10.1. Спецификация формата в виде символьной константы (10.1.2, 10.7.1) 301
С.10.2. Описатель редактирования T (10.6.1.1) 301
С.10.3. Длины форматных записей 301
С.10.4. Количество записей (10.3,10.4,10.6,2) 302
С.10.5. Ввод/вывод, управляемый списком (10.8) 302
С.10.6. Ввод, управляемый списком (10.8.1) 302
С.10.7. Элементы списка NAMELIST для ввода символьных данных (10.9.1.7) 303
С.10.8. Записи вывода по NAMELIST (10.9.2.2) 303

С.11. Примечания к главе 11 303
С.11.1. Главная программная единица и программная единица-блок данных (11.1, 11.4) 303
С.11.2. Зависимая компиляция (11.3) 303
С.11.2.1. Оператор USE и зависимая компиляция (11.3.2) 304
С.11.2.2. Атрибуты доступности (11.3.1) 305
С.11.3. Указатели в модулях 306
С.11.4. Пример модуля (11.3) 306
С.12. Примечания к главе 12 309
С.12.1. Примеры связи по объемлющему контексту (12.1.2.2.1) 309
С.12.2. Внешние процедуры (12.3.2.2) 310
С.12.3. Процедуры, определяемые способом, отличным от Фортрана (12.5.3) 310
С.12.4. Интерфейсы процедур (12.3) 310
С.12.5. Связь и вычисление аргументов (12.4.1) 311
С.12.6. Спецификация назначения аргументов (12.4.1.1) 312
С.12.7. Ограничения на формальные аргументы (12.5.2.9) 312
С.12.8. Указатели и адресаты в качестве аргументов 312
С.12.9. Функция ASSOCIATED (13.13.13) 313
С.12.10. Ограничения на внутренние процедуры 314
С.12.11. Результирующая переменная (12.5.2.2) 314
С.13. Примечания к главе 13 314
С.13.1. Резюме 314
С.13.1.1. Выражения и присваивания для массивов 314
С.13.1.2. Секции массива (2.4.5, 6.2.2.3). 314
С.13.1.3. Оператор WHERE (7.5.3) 315
С.13.1.4. Динамически и автоматически размещаемые массивы (5.1, 5.1.2.4.3) 315
С.13.1.5. Конструкторы массива (4.5) 315
С.13.1.6. Встроенные функции 315
С.13.2. Примеры 316
С.13.2.1. Безусловные вычисления массива 316
С.13.2.2. Условные вычисления массива 317
С.13.2.3. Простая программа: модель Изинга 318
С.13.2.3.1. Задачи, которые должны быть решены 318
С.13.2.3.2. Решения в Фортране 318
С.13.2.3.3. Полная подпрограмма на Фортране 319
С.13.2.3.4. Уменьшение памяти 320
С.13.3. Перевод формул и обработка массивов 320
С.13.3.1. Сумма произведений 320
С.13.3.2. Произведение сумм 321
С.13.3.3. Сложение выбираемых элементов 321
С.13.4. Сумма квадратичных отклонений 321
С.13.5. Нормы вектора: норма-бесконечность и норма-один 321

C.13.6. Нормы матрицы: норма-бесконечность и норма-один 322
C.13.7. Логические вопросы 322
C.13.8. Параллельные вычисления 322
C.13.9. Примеры поэлементного вычисления 323
C.13.10. Процедуры операций с битами 323
C.14. Примечания к главе 14 323
C.14.1. Связь по памяти для объектов нулевого размера 323
ПРИЛОЖЕНИЕ D. Синтаксические правила 325
D.1. Синтаксические правила и ограничения 325
D.1.1. Общие сведения 325
D.1.2. Термины и понятия Фортрана 325
D.1.3. Символы, лексические элементы и форма исходного текста 327
D.1.4. Встроенные и производные типы данных 329
D.1.5. Спецификации и описания объектов данных 331
D.1.6. Использование объектов данных 337
D.1.7. Выражения и присваивание 339
D.1.8. Управление выполнением 341
D.1.9. Операторы ввода/вывода 344
D.1.10. Редактирование ввода/вывода 347
D.1.11. Программные единицы 349
D.1.12. Процедуры 350
D.1.13. Встроенные процедуры 353
D.1.14. Область действия, связь и определенность 353
D.2. Перекрестные ссылки 353
D.2.1. Нетерминальные символы, которые определены 353
D.2.2. Нетерминальные символы, которые не определены 361
D.2.3 Терминальные символы 362
ПРИЛОЖЕНИЕ E. Переупорядоченный индекс для заголовков 366
ПРИЛОЖЕНИЕ F. ИНДЕКС 367

ПРЕДИСЛОВИЕ РЕДАКТОРА ПЕРЕВОДА

Язык Фортран представляет собой уникальное явление в области языков программирования. Появившись одним из первых и будучи вначале несовершенным (с современной точки зрения), он привлек к себе многочисленных пользователей.

Фортран постоянно развивается и совершенствуется в соответствии с развитием вычислительной техники, языков и технологии программирования. Одной из наиболее важных причин популярности и живучести Фортрана является огромный фонд прикладных программ, который накоплен за десятилетия существования языка. Этому в немалой степени способствовали удобство и эффективность выполнения программ, а также стандартизация языка.

Стандартизация языков программирования, и в частности Фортрана, обеспечивает мобильность программ при их переносе из одной вычислительной среды в другую, облегчает обмен программами, что приводит к существенному уменьшению времени разработки и стоимости программного обеспечения.

Бытующее мнение о том, что Фортран устарел – ошибочно. Конечно, в связи с широким внедрением персональных ЭВМ во многие сферы человеческой деятельности удельный вес Фортрана в общем объеме программного обеспечения снизился. Однако при решении научно-технических задач, требующих большого объема вычислений, и особенно при использовании современных многопроцессорных высокопроизводительных вычислительных систем Фортран по-прежнему занимает лидирующее положение.

Предлагаемая читателю книга содержит перевод на русский язык официального описания международного стандарта языка Фортран, неформальное название которого Фортран 90.

Этот стандарт языка Фортран разработан совместно Техническим комитетом (X3J3) Американского национального института стандартов (ANSI) и Рабочей группой Международной организации по стандартизации (ISO/IEC JTC1/SC22/WG5).

Фортран 90 существенно расширяет возможности своих предшественников. В то же время практически сохраняется преемственность с предыдущими стандартами языка, что позволяет использовать ранее созданный фонд прикладного программного обеспечения. Фортран 90 позволяет создавать более мобильные и более надежные программы по сравнению с ФОРТРАНом 77 и обеспечивает современный стиль программирования.

Наиболее важные его новые черты: свободный формат исходной программы, производные типы данных (структуры данных), средства параметризации встроенных типов данных, операции над массивами и секциями массивов, указатели, механизмы динамического размещения объектов в памяти, новые средства для описания и использования глобальных объектов, гибкие управляющие конструкции, определяемые пользователем операции, элементы объектно-ориентированного программирования, более развитые средства использования процедур (внутренние процедуры, рекурсии, возможность специфицировать аргументы по назначению, возможность использования необязательных и ключевых параметров при вызове процедур), а также большой набор новых стандартных процедур и т.п.

Что дают эти новые возможности?

Возможность описания и использования производных типов и указателей позволяет специфицировать сложные структуры данных: списки, деревья, графы и т.п. Новый вид программных единиц (программные единицы-модули), которые предназначены для спецификации глобальных объектов, вместе с возможностью описания новых типов данных и операций являются мощным средством расширяемости языка, выражением концепции абстрактных типов данных. Эти средства позволяют создавать пакеты (модули), ориентированные на конкретные приложения.

Операции над массивами и секциями массивов задают параллелизм действий над компонентами массивов (или массива). Такие средства, с одной стороны, позволяют пользователю лаконично и сжато описать алгоритм обработки массивов и, с другой стороны, дают возможность компилятору генерировать более эффективный код с учетом особенностей конкретной ЭВМ. Очевидно, что наиболее эффективно использование этих возможностей для вычислительных систем, имеющих аппаратные средства для векторной обработки.

Параметризация встроенных типов дает возможность компиляторам поддерживать короткие целые, более двух видов точности для вещественных и комплексных данных, а также многобайтовые символьные данные (для языков с большим набором символов, таких, как китайский или японский) или использовать дополнительные наборы символов для конкретных приложений.

Введенные в язык механизмы динамического размещения массивов позволяют пользователю организовать работу с массивами, размер которых заранее не известен и вычисляется в процессе выполнения программы, а также оперативно отводить память под массивы, требующиеся на том или ином этапе выполнения программы.

Представляет интерес принятая концепция эволюционного развития языка. Некоторые конструкции языка Фортран в настоящее время устарели и стали излишними после введения новых элементов; такие конструкции отнесены к категории устаревших черт. Наличие в языке устаревших черт усложняет язык и компилятор, что негативно отражается на эффективности изготовляемых программ. Кроме того, некоторые устаревшие элементы являются источником ненадежности, другие – влияют на снижение мобильности, многие из таких черт противоречат концепции структурного программирования и препятствуют распараллеливанию программ.

Концепция устаревших черт вызвала много споров; исключение из языка каких-либо элементов при очередном пересмотре стандарта может быть весьма болезненным, так как оно может привести к тому, что невозможно будет использовать существующий фонд программного обеспечения на Фортране. Вместе с тем ясно, что если в язык будут добавляться новые средства без удаления уже ненужных, устаревших элементов, язык станет очень громоздким с большим числом дублирующих элементов.

Учитывая эти противоречивые факторы, было принято следующее решение: элементы языка, которые являются кандидатами на удаление, заранее объявляются устаревшими, нерекомендуемыми для использования; в каждом стандарте языка Фортран должно быть два списка устаревших черт с различным статусом. Первый список содержит удаляемые из данного стандарта черты – те элементы языка, которые в предыдущем стандарте были объявлены как устаревшие. Для Фортрана 90 этот список пустой, т.е. ФОРТРАН 77 является точным подмножеством Фортрана 90 (в ФОРТРАНе 77, как

известно, не было вообще списков устаревших черт). Второй список содержит отживающие черты – те элементы языка, которые являются кандидатами на удаление в следующей ревизии стандарта (хотя они необязательно должны быть удалены). Такие черты в данном стандарте выделены мелким шрифтом. Список устаревших черт и средства достижения аналогичного результата с помощью новых элементов языка приведены в приложении В.

Новый стандарт предъявляет более строгие требования к компиляторам. Компилятор должен выявлять и сообщать пользователю не только конструкции, которые не отвечают требованиям языка, но также и устаревшие конструкции и расширения языка.

В настоящее время Фортран 90 реализован практически для всех современных компьютеров от персональных ЭВМ до больших параллельных вычислительных систем. Для некоторых компьютеров имеется более одного компилятора, больше всего реализаций для персональных компьютеров. Современные компиляторы, как правило, снабжены хорошими сервисными возможностями.

Созданы и другие программные продукты. Для того чтобы эффективно использовать ранее написанные на ФОРТРАНе 77 программы, разработаны трансляторы, переводящие программы с ФОРТРАНа 77 на Фортран 90. Такие трансляторы заменяют устаревшие конструкции на новые, введенные в Фортран 90, а также те конструкции, для которых имеются лучшие решения в Фортране 90.

Разработаны библиотеки численных методов на Фортране 90 (NAG, IMSL). Помимо официального описания стандарта языка, выпущены десятки книг на разных языках, которые содержат неформальное описание Фортрана 90. В ряде университетов читаются лекции по Фортрану 90; созданы разнообразные курсы по изучению этого языка.

Развитие Фортрана продолжается и сейчас; активно ведется разработка проекта будущего стандарта, рабочее название которого Фортран 2000.

Заслуживает внимания разработка и реализация подмножества Фортрана 90, получившего название «язык F». Подмножество содержит все современные элементы Фортрана 90 и не содержит устаревших конструкций (сохранившихся в Фортране 90 для преемственности), поскольку для них имеются более мощные средства в языке. Удаление таких конструкций, а также введение дополнительных ограничений позволило сделать язык F более компактным и элегантным по сравнению с Фортраном 90 и обеспечить хороший стиль программирования. Язык F ориентирован на обучение и на использование в профессиональной деятельности. Другой вариант подмножества, основанного на аналогичном подходе, – Elf 90, разработанный и реализованный известной фирмой по разработке Фортран-компиляторов Lahey.

На базе Фортрана 90 разработан язык HPF (High Performance Fortran), который предназначен для написания параллельных программ для многопроцессорных вычислительных комплексов. Язык является расширением Фортрана 90 и существенно использует его новые возможности. HPF содержит директивы для описания способов разбиения больших массивов данных между параллельно работающими процессорами, а также операторы и директивы для явного указания параллельности.

Предлагаемый читателю стандарт языка Фортран 90 представляет собой строго формализованное описание языка. В то же время описание содержит много примеров. Весьма полезно для читателя приложение С, содержащее разъяснения многих новых и наиболее трудных элементов языка.

Несомненно, что учебная и методическая литература обычно более понятна для читателя, особенно при первоначальном освоении нового языка. Однако она, как правило, содержит неформальное и недостаточно полное описание и не может дать ответы на все возникающие тонкие вопросы, ответы на которые можно получить из формального описания.

Хотя приложения не являются частью стандарта, они весьма полезны для читателя. Словарь технических терминов (приложение A), перечень всех синтаксических правил и ограничений (приложение D) и индексы (приложение F) удобны для использования документа в качестве справочного пособия. О приложениях B и C упоминалось выше.

Можно надеяться, что российским специалистам будет полезно ознакомиться с переводом данного документа, ознакомиться с новым и уже реализованным языком программирования, отражающим самые современные тенденции.

* * *

Большую пользу редактору перевода принесло участие в деятельности Международной рабочей группы ISO/IEC JTC1/SC22/WG5 и личные контакты в ходе рабочих совещаний. Пользуясь случаем, хочу выразить мою глубокую признательность всем членам рабочей группы, с которыми мне довелось контактировать.

А.М. Горелик

Литература на русском языке по Фортрану 90

1. **Горелик А.М., Ушкова В.Л.** Фортран сегодня и завтра. – М.: Наука, 1990.
2. **Меткалф М., Рид Дж.** Описание языка программирования Фортран 90: Пер. с англ. – М.: Мир, 1995.
3. **Горелик А.М.** Фортран жил, жив и будет жить // Открытые системы сегодня. – 1995. – №1.
4. **Горелик А.М.** Эволюция языка Фортран. Устаревшие черты и современные элементы языка для их замены. – М.: Препринт Института прикладной математики РАН. – 1997. – № 66.

ПРЕДИСЛОВИЕ К РУССКОМУ ПЕРЕВОДУ

Читателю предлагается перевод с английского языка международного стандарта языка Фортран, неформально называемого Фортран 90. Фортран 90 практически полностью включает ФОРТРАН 77 и является его существенным расширением.

При переводе стандарта на русский язык мы столкнулись с двумя основными проблемами. Первая из них связана с тем, что в синтаксических правилах и затем в тексте для нетерминальных символов часто используются сокращения, которые легко понимаются в английском тексте (например, stmt, expr). Благодаря таким сокращениям и вследствие лаконичности английского языка синтаксические правила довольно компактны и наглядны. В русском языке из-за падежных окончаний и вследствие некоторых других особенностей языка использование подобных сокращений делает текст нечитабельным. В связи с этим было решено в русском тексте использовать сокращения только в очень редких случаях.

Вторая проблема при переводе – терминология. При переводе традиционных для Фортрана терминов мы стремились использовать наиболее часто встречающиеся в отечественной литературе эквиваленты. Для новых фортрановских понятий и названий синтаксических классов, которые используются в формальных правилах, подобрать русские эквиваленты оказалось непростой задачей.

* * *

Мы благодарны И.С. Голосову за поддержку данного перевода и признательны также нашим коллегам из Белоруссии З.С.Брич и С.Дорошко за участие в обсуждении. Мы рады выразить благодарность также М.Р.Когаловскому за помощь и поддержку, Н.Ю.Петрашковой и В.Л.Ушковой за замечания и ценные советы.

Мы благодарим координатора Международной рабочей группы по Фортрану Ж.Мартин, а также компанию Amoco Production Research за поддержку. Мы особенно признательны члену X3J3 ANSI Дж.Вагенеру, благодаря огромным усилиям которого этот перевод был подготовлен к печати и напечатан малым тиражом.

А.М. Горелик, С.Г.Дробышевич, 1994

ПРЕДИСЛОВИЕ

ИСО (ISO) – Международная организация по стандартизации и МЭК (IEC) – Международная электротехническая комиссия образуют специализированную систему для всемирной стандартизации. Национальные группы, которые являются членами ИСО или МЭК, участвуют в разработке Международных стандартов через технические комитеты, созданные соответствующей организацией для работы в конкретных областях технической деятельности. Технические комитеты ИСО и МЭК сотрудничают в областях взаимных интересов. Другие международные организации, правительственные и неправительственные, в сотрудничестве с ИСО и МЭК тоже принимают участие в работе.

В области информационных технологий ИСО и МЭК организовали объединенный комитет ISO/IEC JTC1. Проекты Международных стандартов, принятые объединенным комитетом, рассылаются в национальные группы для голосования. Для публикации в качестве Международного стандарта требуется поддержка как минимум 75% национальных групп, участвующих в голосовании.

Международный стандарт ISO/IEC 1539 подготовлен Объединенным техническим комитетом ISO/IEC JTC1 *Информационные технологии.*

Эта вторая редакция отменяет и заменяет первую редакцию (ISO 1539:1980), которая была пересмотрена.

Приложения A, B, C, D, E и F являются только информационными.

ВВЕДЕНИЕ

Стандартный язык программирования Фортран

Данный Международный стандарт определяет форму и устанавливает интерпретацию программ, написанных на языке Фортран (известный неформально как "Фортран 90"). Он содержит спецификацию языка Фортран. Подмножества данным Международным стандартом не определяются. Кроме некоторых ограничений, перечисленных в 1.4.1, данный Международный стандарт полностью включает синтаксис и семантику Международного стандарта, известного как ФОРТРАН 77. Таким образом, любая соответствующая стандарту ФОРТРАНа 77 программа соответствует и данному стандарту. В такие программы можно включать новые языковые средства, исключения явно указываются в тексте стандарта.

Стандартный Фортран-процессор соответствует также и стандарту ФОРТРАН 77.

Заметим, что название этого языка, Фортран, отличается от ФОРТРАН 77 тем, что только первая буква в нем - заглавная. И ФОРТРАН 77, и ФОРТРАН 66 использовали в официальном названии языка только заглавные буквы. Фортран 90 не продолжил эту традицию.

ОБЩИЕ СВЕДЕНИЯ

В данном Международном стандарте можно выделить семь основных дополнений к ФОРТРАНу 77:

1) операции над массивами;
2) улучшенные средства для численных вычислений;
3) параметризованные встроенные типы данных;
4) определяемые пользователем типы данных;
5) средства для модуляризации данных и описания процедур;
6) указатели;
7) концепция развития языка.

В данный стандарт включен и ряд других расширений, таких, как улучшенная форма исходного текста, большее число управляющих конструкций, рекурсия, дополнительные средства ввода/вывода и динамически размещаемые массивы.

Операции над массивами

Важной частью инженерных и научных расчетов являются вычисления, включающие большие массивы. В Фортране массивы можно использовать как объекты. Операции для обработки полных массивов и подмассивов (секций массивов) включены в язык по двум основным причинам:

1) эти средства делают язык более компактным и повышают его уровень, что позволяет программистам быстро и надежно разрабатывать и обслуживать научно-технические приложения;
2) эти средства могут существенно обеспечить оптимизацию операций над массивами на многих компьютерных архитектурах.

Арифметические, логические, символьные операции и встроенные (предопределенные) функции ФОРТРАНа 77 расширены операциями над операндами-массивами. Расширения для массивов включают присваивание значений полному массиву, части массива и присваивание по маске, константы и выражения со значением-массивом и возможности описания определяемых пользователем функций со значением-массивом. Новые встроенные процедуры предназначены для манипуляций с массивами, их конструирования, для выполнения операций слияния/разделения массивов и для поддержания расширенных вычислительных возможностей обработки массивов. Например, имеется встроенная функция, выполняющая суммирование элементов массива.

Числовые вычисления

Научные вычисления относятся к числу основных прикладных областей применения Фортрана, и руководящим принципом всей работы является укрепление Фортрана как средства реализации научного программного обеспечения. И хотя нечисловые вычисления в научных приложениях чрезвычайно разрастаются, все-таки доминируют числовые вычисления. Соответственно, расширения включают мобильное управление заданием числовой точности, запрос относительно характеристик числового представления и улучшенное управление выполнением численных программ.

Параметризованный символьный тип данных

Благодаря введению необязательного параметра вида для символьного типа данных добавляются возможности использования многобайтовых символьных данных для языков с большим набором символов, таких, как китайский и японский. Это средство позволяет также использовать дополнительные наборы символов для специальных целей, такие, как символы для математики, химии или музыки.

Производные типы

Производный тип - это термин, относящийся в данном Международном стандарте к такому набору средств, который позволяет программисту определять произвольные структуры данных и операции над ними. Структуры данных - это определенные пользователем агрегаты из встроенных и производных типов данных. Встроенные применения структурированных объектов включают присваивание, ввод/вывод и использование в качестве аргументов процедуры. Без дополнительно определенных пользователем операций над производным типом эти возможности производного типа данных являются просто механизмом структурирования данных. При наличии описания дополнительных операций производные типы обеспечивают эффективный механизм абстракции данных.

Для описания операций над встроенными или производными типами и невстроенных присваиваний для встроенных и производных типов могут быть использованы описания процедур.

Модульные определения

В ФОРТРАНе 77 не было способа определить область глобальных данных в одном месте с тем, чтобы все программные единицы в прикладной программе могли использовать это определение. Кроме того, оператор ENTRY является крайне неудобным и ограничивающим элементом языка при реализации связанного набора процедур, возможно, включающего общие объекты данных. И, наконец, в ФОРТРАНе 77 не было способа, посредством которого описания процедур, особенно информация об интерфейсе, могли бы стать известными внутри программной единицы. Эти и другие проблемы решаются введением нового вида программной единицы, которая может содержать любые комбинации объявлений объектов данных, описаний производных типов и процедур, информацию об интерфейсе процедуры. Эту программную единицу, называемую модулем, можно рассматривать как обобщение и замену для программной единицы - блока данных. Модуль может быть доступным для любой программной единицы, что делает доступным его содержимое для этой программной единицы. Таким образом, модули предоставляют улучшенные возможности для описания областей глобальных данных, пакетов процедур и для абстракций инкапсулированных данных.

Указатели

Указатели позволяют динамически определять размеры и границы массивов, связывать структуры, образуя списки, деревья и графы. Объект любого встроенного или производного типа может быть описан с атрибутом POINTER. Будучи однажды связан с

адресатом, такой объект может появиться почти везде, где может появиться объект, не являющийся указателем, с тем же типом, параметрами типа и той же конфигурации.

Эволюция языка

С добавлением новых возможностей некоторые старые средства становятся избыточными и могут быть окончательно исключены из языка в связи с сокращением их использования. Например, числовые возможности, упомянутые выше, перекрывают возможности двойной точности; при новых возможностях работы с массивами неконформная связь аргументов (такая, как связь элемента массива с формальным массивом) становится ненужной (и в действительности ее и нельзя считать операцией над массивом); программные единицы - блоки данных являются избыточными и более низкого уровня, чем модули. Как часть эволюции языка выделяются категории языковых средств (исключаемые и устаревшие), которые позволили бы удалить из будущих стандартов неиспользуемые средства языка.

ОРГАНИЗАЦИЯ ДАННОГО МЕЖДУНАРОДНОГО СТАНДАРТА

Данный документ состоит из 14 глав в соответствии с семью концептуальными областями. Вот эти семь областей и главы, посвященные каждой из них.

Понятия верхнего/нижнего уровней	главы 1, 2, 3
Концепция данных	главы 4, 5, 6
Вычисления	главы 7, 13
Управление исполнением	глава 8
Ввод/вывод	главы 9, 10
Программные единицы	главы 11, 12
Область действия и правила связывания	глава 14

Понятия верхнего/нижнего уровней

Глава 2 (Термины и понятия Фортрана) содержит многие из понятий верхнего уровня Фортрана. Они включают понятие выполняемой программы и отношения между ее основными частями. Кроме того, включены синтаксис программных единиц, правила следования операторов и определения многих фундаментальных терминов, используемых во всем документе.

Глава 3 (Символы, лексемы и форма ввода исходного текста) описывает элементы нижнего уровня Фортрана, такие, как набор символов и допустимые формы для исходных программ. Она также содержит правила построения литеральных констант и имен для объектов Фортрана и списки всех операций Фортрана.

Концепция данных

Операции с массивами (массивы как объекты данных) и структуры данных предоставляют широкий спектр понятий, относящихся к данным в Фортране. Основные понятия относятся к типам данных, объектам данных и к использованию объектов данных, которые описаны соответственно в главах 4, 5 и 6.

Глава 4 (Встроенные и производные типы данных) описывает разницу между типом данных и объектом данных, сосредотачиваясь затем на типе данных. Она определяет тип данных как множество значений данных, соответствующие формы (константы) для представления этих значений и операции над этими значениями. Вводится понятие встроенного типа данных и описываются характеристики встроенных типов в Фортране (INTEGER, REAL, COMPLEX, LOGICAL и CHARACTER). Заметим, что здесь описываются только понятия типа, а не объявления и не свойства объектов данных.

Глава 4 вводит также понятие производных типов данных, которые являются составными типами, компоненты которых, в конце концов, распадаются на встроенные типы данных. Приводятся детали описания производного типа (заметим, что здесь нет

аналогии со встроенными типами данных, которые являются предопределенными и поэтому не должны и не могут быть переопределены программистом). Как и в случае со встроенными типами, эта глава касается только свойств типа, но не объявлений объектов производного типа.

Глава 5 (Объявления и спецификации объектов данных) подробно описывает, как надо объявлять именованные объекты данных и задавать им нужные характеристики (атрибуты). Важным атрибутом (задаваемым для каждого объекта данных только однажды) является тип объекта данных, поэтому операторы объявления типа являются основной темой данной главы. Подробно описываются различные атрибуты и два способа их задания (оператор объявления типа и операторы спецификации атрибутов). В этой главе также описываются неявная типизация, связь по памяти (COMMON и EQUIVALENCE) и инициализация значений объектов данных.

Глава 6 (Использование объектов данных) посвящена, в основном, понятию переменной и описывает различные формы, которые может принимать переменная. Скалярные переменные включают символьные строки и подстроки, структурированные (производного типа) объекты, компоненты структуры и элементы массива. Массивы, так же как и секции массива, рассматриваются как переменные. Из свойств массивов здесь рассматриваются секции массива (подмассивы), отведение памяти под массивы и ее освобождение (управляемые пользователем динамические массивы).

Вычисления

Глава 7 (Выражения и присваивание) описывает, как в Фортране изображаются вычисления. Она включает и формы, которые могут принимать операнды (первичные) в выражениях, и функцию знаков операций в этих выражениях. Старшинство операций строго определяется синтаксическими правилами и обобщается в виде таблиц. Это описание включает взаимосвязь между операциями, определяемыми пользователем, и встроенными операциями (+, *, .AND., .OR. и т.д.). Подробно описываются правила вычисления выражений и интерпретации (семантика) встроенных и определяемых операций.

В главе 7 также описывается присваивание результатов вычисления объектам данных, имеющее три основные формы: общепринятый оператор присваивания, оператор присваивания указателю и оператор и конструкция WHERE. Оператор и конструкция WHERE позволяют выполнять присваивание массиву по маске.

Глава 13 (Встроенные процедуры) описывает более сотни встроенных процедур, обеспечивающих богатый набор вычислительных возможностей. В дополнение ко встроенным функциям ФОРТРАНа 77 включено много функций обработки массивов, широкий набор функций запроса о числовой среде, набор процедур для работы с битами в неотрицательных целых данных.

Управление исполнением

Глава 8 (Управление выполнением) описывает управляющие конструкции (IF, CASE и DO), операторы ветвления (различные формы GO TO) и другие управляющие операторы (IF, арифметический IF, CONTINUE, STOP и PAUSE). Они такие же, как в ФОРТРАНе 77, но добавлены конструкция CASE и расширение цикла DO включением возможности окончания цикла с помощью оператора END DO, дополнительными видами управляющей информации и операторами EXIT и CYCLE.

Ввод/вывод

Глава 9 (Операторы ввода/вывода) содержит описания записей, файлов, подсоединения файлов (OPEN, CLOSE и предварительно подсоединенные файлы), операторов передачи данных (READ, WRITE и PRINT), включая обработку частей записей и обработку записей переменной длины, установление позиции файла и запрос о файле (INQUIRE).

Глава 10 (Редактирование ввода/вывода) описывает форматирование ввода/вывода, включая оператор FORMAT и спецификатор FMT=, описатели редактирования, ввод/вывод, управляемый списком, и ввод/вывод, управляемый списком имен (NAMELIST).

Программные единицы

Глава 11 (Программные единицы) описывает главные программные единицы, внешние процедуры, модули и блоки данных. Модули, вместе с оператором USE, описаны как механизм для инкапсуляции данных и описаний процедур, которые могут быть использованы другими программными единицами (доступны для других программных единиц). Модули описываются как средства описания глобальных производных типов, объявления глобальных объектов данных, библиотек процедур и комбинаций всего перечисленного.

Глава 12 (Процедуры) содержит исчерпывающую трактовку описаний и вызовов процедур, включая определяемые пользователем функции и подпрограммы. Разъясняются понятия явного и неявного интерфейсов процедуры и ситуации, когда требуется явное задание интерфейса. Описаны правила управления формальными и фактическими аргументами и правила их связывания.

Кроме того, в главе 12 описывается использование спецификации OPERATOR в блоках интерфейса, позволяющей вызывать функции как в форме инфиксной и префиксной операции, так и в традиционной для функций форме. Аналогично описывается спецификация ASSIGNMENT в блоках интерфейса, допускающая альтернативный синтаксис для вызовов некоторых процедур. Эта глава содержит также описание рекурсивных процедур, операторов RETURN и ENTRY, внутренних процедур и оператора CONTAINS, операторных функций, обобщенных имен процедур и способов обращения к нефортрановским процедурам.

Область действия и правила связывания

Глава 14 (Область действия, связь и определенность) разъясняет использование термина "область действия" (особенно важного сейчас, когда добавились внутренние процедуры, модули и другие новые средства) и описывает особенности области действия различных объектов, включая имена и знаки операций. Кроме того, описываются основные правила, управляющие установлением связи аргументов процедуры, связи указателей и связи по памяти. И, наконец, глава 14 описывает случаи, когда переменные становятся определенными (имеют предсказуемое значение), и случаи, когда переменные становятся неопределенными.

Глава 1. ОБЩИЕ СВЕДЕНИЯ

1.1. Цель

Данный Международный стандарт задает форму и устанавливает интерпретацию программ, написанных на языке Фортран. Цель данного Международного стандарта - способствовать мобильности, надежности, удобству сопровождения, эффективности исполнения Фортран-программ с целью использования их в разнообразных вычислительных системах.

1.2. Процессор

Сочетание вычислительной системы и механизма, преобразующего программу для использования ее в этой вычислительной системе, называется в данном Международном стандарте **процессором**.

1.3. Что включено в стандарт и что не включено

Данный Международный стандарт задает границы языка Фортран, обозначая те элементы, которые в него включены, и те элементы, которые не включены.

1.3.1. Что включено

Данный Международный стандарт определяет:

1) формы, которые может принимать программа, написанная на языке Фортран;
2) правила интерпретации смысла программы и ее данных;
3) форму входных данных, которую должна обрабатывать такая программа;
4) форму выходных данных, полученных при использовании этой программы.

1.3.2. Что не включено

Данный Международный стандарт не определяет:

1) механизм преобразования программ для использования их в вычислительных системах;
2) действия, необходимые для запуска и управления при использовании программ в вычислительных системах;
3) способ перевода программ и их входных и выходных данных в память и из памяти;
4) поведение программ и процессоров в случаях, когда правил данного Международного стандарта недостаточно для установления интерпретации (кроме перечисленных в пунктах 2-8 раздела 1.4 требований к процессору относительно обнаружения ошибок и выдачи сообщений);
5) размер и сложность программы и ее данных, превышающие мощность какой-либо конкретной вычислительной системы или возможности конкретного процессора;
6) физические характеристики представления величин и способ округления при вычислении числовых значений в конкретном процессоре;
7) физические характеристики записей, файлов и устройств ввода/вывода;
8) физические характеристики и организация памяти в реализации.

1.4. Соответствие стандарту

Требования, запреты и режимы, определяемые в этом Международном стандарте, относятся к формам и отношениям, допустимым для соответствующих стандарту программ, а не для процессоров.

Выполняемая программа (2.2.1) является **соответствующей стандарту программой**, если она использует только те формы и отношения, которые здесь описаны, и если выполняемая программа имеет интерпретацию, соответствующую данному Международному стандарту. Программная единица (2.2) соответствует данному Международному стандарту, если ее можно включить в выполняемую программу таким образом, чтобы выполняемая программа соответствовала стандарту.

Процессор соответствует данному Международному стандарту, если

1) он выполняет соответствующую стандарту программу таким образом, чтобы была реализована предписанная стандартом интерпретация, с учетом всех ограничений, которые процессор может наложить на форму и сложность программы;
2) он располагает возможностью обнаружить в представленной программной единице и выдать сообщение по поводу тех форм, которые обозначены в стандарте как исключенные или устаревшие, насколько такое использование может быть выявлено посредством ссылки на пронумерованные синтаксические правила и их ограничения;
3) в нем есть возможность обнаружить в представленной программной единице и выдать сообщение по поводу дополнительных форм и отношений, не разрешенных пронумерованными синтаксическими правилами и связанными с ними ограничениями;
4) он располагает возможностью обнаружить в представленной программной единице использование значений параметров вида (4.3), не поддерживаемых процессором, и может сообщить об этом;
5) в нем есть возможности для обнаружения в представленной программной единице исходного текста или символов, не разрешенных в главе 3, и он может сообщить об этом;
6) он может обнаружить (и сообщить об этом) использование имен в представленной программе, не соответствующее правилам области действия для имен, меток, символов операций и присваивания из главы 14;
7) он может выявить в представленной программной единице встроенные процедуры, имена которых не определены в главе 13, и сообщить об этом;
8) он имеет возможность выявить причину сбоя представленной программы и сообщить об этом.

Однако от процессора не требуется выявлять или отмечать использование исключенных, нерекомендуемых или устаревших средств или использование дополнительных форм и отношений, встречающихся в спецификации формата, которая не является частью *оператора-FORMAT* (10.1.1).

Соответствующий стандарту процессор может разрешить дополнительные формы и отношения при условии, что они не будут противоречить стандартным. Однако соответствующий стандарту процессор может разрешить дополнительные встроенные процедуры, даже если это приведет к конфликту с именем процедуры в соответствующей стандарту программе. Если такой конфликт возник и включает имя внешней процедуры, соответствующий стандарту процессор должен позволять использование имени встроенной процедуры только в том случае, когда это имя не определяет тело интерфейса и не имеет атрибута EXTERNAL в той же контекстной области (14). Соответствующая стандарту программа не должна использовать нестандартные встроенные процедуры, добавленные процессором.

Заметим, что соответствующая стандарту программа не должна использовать любые формы и отношения, запрещенные стандартом, тогда как соответствующий стандарту процессор может их разрешить, если они не меняют правильной

интерпретации соответствующей стандарту программы. Например, соответствующий стандарту процессор может разрешить нестандартный тип данных.

Поскольку соответствующая стандарту программа может поместить запросы к процессору, которые не входят в область действия данного стандарта, или может содержать стандартные элементы, не обладающие свойством мобильности, как например, внешние процедуры, определяемые не средствами Фортрана, все это приводит к тому, что соответствие данному стандарту еще не дает гарантии в том, что соответствующая стандарту программа будет выполняться одинаково на всех или на некоторых соответствующих стандарту процессорах.

В некоторых случаях стандарт разрешает использовать средства, не полностью описанные в стандарте. Такие средства идентифицируются как **процессорно-зависимые**, они должны обеспечиваться процессором вместе с определяемыми процессором методами или семантикой.

1.4.1. Совместимость с ФОРТРАНом 77

С учетом отмеченных в этой главе исключений данный стандарт является совместимым расширением предыдущего стандарта Фортрана, ISO 1539:1980, обычно именуемого ФОРТРАНом 77, и любой соответствующий данному Международному стандарту процессор является стандартным и для ФОРТРАНа 77. Любая программа, соответствующая стандарту с точки зрения ФОРТРАНа 77, остается таковой и для данного стандарта, сохраняя при этом свою интерпретацию; однако см. ниже пункт 4 относительно встроенных процедур. Данный Международный стандарт ограничивает режимы для некоторых средств, которые были оставлены в ФОРТРАНе 77 как процессорно-зависимые. Таким образом, стандартная с точки зрения ФОРТРАНа 77 программа, использующая такие процессорно-зависимые средства, может иметь в данном Международном стандарте другую интерпретацию, оставаясь при этом программой, соответствующей стандарту. Другую интерпретацию в данном Международном стандарте имеют следующие средства ФОРТРАНа 77.

1. При инициализации объекта двойной точности вещественной константой в операторе DATA, ФОРТРАН 77 разрешает процессору извлекать из константы большую точность, чем может вместить вещественное данное. Данный стандарт запрещает процессору такой режим.
2. Если именованная переменная, не входящая в общий блок, инициализируется в операторе DATA, но не включена в оператор SAVE, ФОРТРАН 77 оставляет ее атрибут SAVE процессорно-зависимым. Данный стандарт описывает ее с атрибутом SAVE (5.2.9).
3. ФОРТРАН 77 требует, чтобы количество символов, затребованное списком ввода при форматном вводе, было меньше или равно количеству символов в записи. В данном Международном стандарте говорится, что если в операторе OPEN не задан режим PAD=NO (9.4.4.4.2) и если не хватает значений в списке ввода, входная запись логически заполняется пробелами.
4. В данном Международном стандарте больше встроенных функций, чем в ФОРТРАНе 77, и добавлено несколько встроенных подпрограмм. Таким образом, стандартная для ФОРТРАНа 77 программа может иметь другую интерпретацию в данном стандарте, если она вызывает процедуру, имя которой совпадает с именем одной из новых встроенных процедур, и если эта процедура не включена в оператор EXTERNAL, как это рекомендовано для невстроенных функций в приложении к стандарту ФОРТРАН 77.

1.5. Условные обозначения

В данном Международном стандарте слово "должен" трактуется как требование, и наоборот, "не должен" ("не может") трактуется как запрет.

1.5.1. Синтаксические правила

Для удобства описания тех форм, которые могут принимать операторы и конструкции Фортрана, используются **синтаксические правила**. Они представляют собой разновидность форм Бекуса-Наура (BNF), в которых используются следующие правила.

1. Символы из набора символов Фортрана (3.1) должны быть записаны так, как показано, исключая специально оговоренные случаи.
2. Прописные выделенные курсивом буквы и слова (часто написанные через дефис и сокращенные) представляют обобщенные синтаксические классы, для которых конкретные синтаксические объекты должны быть заменены реальными операторами.

   Некоторые сокращения, наиболее часто используемые в синтаксических терминах:

   специф - спецификатор *оп* - знак операции

   нцкл - неявного-цикла *вв* - ввода/вывода
3. Используемые синтаксические метасимволы:

   есть вводит определение синтаксического класса;

   или вводит альтернативу синтаксического класса;

   [] обрамляет необязательный элемент;

   []... обрамляет необязательный повторяемый элемент, который может появиться нуль или более раз;

   ♦ продолжает синтаксическое правило.
4. Каждое синтаксическое правило задается уникальным идентифицирующим номером вида R*sNN*, где *s* - одна или две цифры номера главы, а *NN* - порядковый номер внутри этой главы. Синтаксические правила расположены в соответствующих разделах текста и в случае надобности на них можно ссылаться по номеру. Некоторые правила из глав 2 и 3 более полно описаны в последующих главах; в таких случаях *s* является номером той последующей главы, где правило повторяется. Кроме того, все правила собраны в приложении D.
5. Синтаксические правила не являются полным и точным описанием синтаксиса Фортрана и не могут быть использованы для автоматического построения Фортрановского анализатора; в тех местах, где синтаксическое правило является неполным, оно сопровождается соответствующими ограничениями и текстом.
6. Устаревшие (архаичные) (1.6) средства выделены шрифтом другого размера (пример шрифта для архаичных средств).

Пример использования синтаксических правил:

строка-цифр **есть** *цифра* [*цифра*]...

Этим правилом разрешаются следующие формы строки цифр:

цифра
цифра цифра
цифра цифра цифра цифра
цифра цифра цифра цифра цифра цифра цифра цифра

Когда вместо цифры подставляется конкретное ее содержание, реальные строки цифр могут быть такими:

4
67
1999
10243852

1.5.2. Подразумеваемые синтаксические правила

Чтобы уменьшить число дополнительных синтаксических правил и передать соответствующую информацию об ограничениях, предполагаются следующие правила. Буквы *xyz* подставляются для любой правильной фразы синтаксического класса:

список-xyz	**есть** *xyz* [,*xyz*]...
имя-xyz	**есть** *имя*
скалярное-xyz	**есть** *xyz*

Ограничение: *скалярное-xyz* должно быть скаляром.

1.5.3. Синтаксические соглашения и характеристики

1. Любое имя синтаксического класса, которое начинается с префикса "*оператор-*", соответствует правилам для операторов исходного текста: оно должно быть ограничено концом строки или точкой с запятой и может быть помечено, если не является частью другого оператора (например, оператора IF или WHERE). И наоборот, все, что рассматривается как оператор исходного текста, определяется в синтаксических правилах, начинающихся с префикса "*оператор-*".
2. Правила упорядочения операторов точно описаны в определении программной единицы (R202-R216). Иерархия выражений точно описана в определении *выражения* (R723).
3. Префикс "*специф-*" используется соответственно для спецификаторов, таких как фактические аргументы с ключевыми словами и спецификаторы операторов ввода/вывода. Он также используется для спецификации атрибутов описания типа (например, "*специф-массива*" в R512) и в нескольких других случаях.
4. Когда производится ссылка на параметр типа вместе с окружающими его скобками, используется термин "селектор", например, "*селектор-длины*" (R507) и "*селектор-вида*" (R505).
5. Термин "*индекс*" (например, R617, R618 и R619) используется в определениях массива.

1.5.4. Текстовые обозначения

В описательном тексте используются русские слова, которые эквивалентны синтаксическим терминам BNF. Конкретные операторы и атрибуты идентифицируются в тексте заглавными английскими буквами, как, например, "оператор END". При первом употреблении специального термина используется выделение нужных слов текста.

1.6. Исключенные и устаревшие средства

Этот Международный стандарт сохраняет для пользователя все накопленное в программном обеспечении за счет сохранения всех элементов языка ФОРТРАН 77, которые не являются процессорно-зависимыми. Данный документ идентифицирует две категории отживших средств. Первую категорию - **исключенные средства**, образуют те элементы языка, которые считались избыточными уже в ФОРТРАНе 77 и, в основном, уже тогда не использовались. Вторая категория, **устаревшие средства** (архаизмы), были избыточными в Фортране, но использовались довольно широко.

1.6.1. Природа исключенных средств:

1) в ФОРТРАНе 77 существуют более совершенные методы;
2) эти средства не включены в данную ревизию Фортрана.

1.6.2. Природа устаревших средств:

1) в ФОРТРАНе 77 есть более совершенные методы;
2) именно эти лучшие средства рекомендуется использовать программистам для написания новых программ на Фортране и переделки старых программ;
3) устаревшие средства выделяются в тексте шрифтом другого размера (1.5.1);
4) если использование этих средств в Фортран-программах стало незначительным, последующим комитетам по стандарту Фортрана рекомендуется исключить их при следующей ревизии;
5) будущим комитетам по стандарту Фортрана рекомендуется при следующей ревизии Фортрана обсудить удаление только тех языковых средств, которые включены в список устаревших средств;
6) рекомендуется, чтобы процессоры, поддерживающие язык Фортран, продолжали поддержку этих средств до тех пор, пока они продолжают широко использоваться в Фортран-программах.

1.7. Модули

Данный Международный стандарт предоставляет средства, способствующие разработке и использованию модульного и повторно используемого программного обеспечения. Описания данных и процедур могут быть организованы в невыполняемые программные единицы, называемые модулями, и эти описания становятся доступными для любой другой программной единицы. Помимо возможностей создания глобальных данных и библиотеки процедур, предоставляется механизм для определения абстракций данных и некоторых языковых расширений. Модули описываются в 11.3.

Модуль может быть стандартизован как отдельный сопутствующий стандарт. **Стандартный модуль** не должен использовать исключенных и устаревших средств или нестандартных форм и отношений.

1.8. Нормативные ссылки

Перечисленные ниже стандарты содержат положения, которые посредством ссылок на них в данном тексте образуют положения данного Международного стандарта. Во время публикации указанные редакции были действующими. Все стандарты подвергаются ревизиям, и части соглашений, базирующиеся на данном Международном стандарте, поддерживаются, чтобы сохранить возможность применения их в более поздних редакциях перечисленных ниже стандартов. Члены МЭК и ИСО поддерживают списки действующих в данный момент Международных стандартов.

ISO 8601:1988, *Data Elements and Interchange Formats - Information interchange - Representation of dates and times.*

ISO 646:1983, *Information processing - ISO 7-bit coded character set for information interchange.*

CCIR Recommendation 460-2.

Глава 2. ТЕРМИНЫ И ПОНЯТИЯ ФОРТРАНА

2.1. Синтаксис верхнего уровня

В данной главе вводятся термины Фортрана, связанные с программными единицами и другими понятиями более высокого уровня, чем конструкции, операторы и выражения, и иллюстрируются их взаимосвязи. Правила записи синтаксиса описаны в 1.5.1. Заметим, что некоторые из синтаксических правил этой главы подчинены ограничениям, приведенным только в соответствующих местах более поздних глав.

R201 *выполняемая-программа* **есть** *программная-единица*
[*программная-единица*]...

R202 *программная-единица* **есть** *главная-программная-единица*
или *внешняя-процедура*
или *модуль*
или *блок-данных*

R1101 *главная-программная-единица* **есть** [*оператор-PROGRAM*]
[*раздел-спецификаций*]
[*раздел-выполняемых-операторов*]
[*раздел-внутренних-процедур*]
оператор-конца-программы

R203 *внешняя-процедура* **есть** *процедура-функция*
или *процедура-подпрограмма*

R1215 *процедура-функция* **есть** *оператор-FUNCTION*
[*раздел-спецификаций*]
[*раздел-выполняемых-операторов*]
[*раздел-внутренних-процедур*]
оператор-конца-функции

R1219 *процедура-подпрограмма* **есть** *оператор-SUBROUTINE*
[*раздел-спецификаций*]
[*раздел-выполняемых-операторов*]
[*раздел-внутренних-процедур*]
оператор-конца-подпрограммы

R1104 *модуль* **есть** *оператор-MODULE*
[*раздел-спецификаций*]
[*раздел-модульных-процедур*]
оператор-конца-модуля

R1110 *блок-данных* **есть** *оператор-BLOCK-DATA*
[*раздел-спецификаций*]
оператор-конца-блока-данных

R204 *раздел-спецификаций* **есть** [*оператор-USE*]...
[*раздел-IMPLICIT*]
[*конструкция-объявления*]...

R205 *раздел-IMPLICIT* **есть** [*оператор-раздела-IMPLICIT*]...
оператор-IMPLICIT

R206 *оператор-раздела-IMPLICIT* **есть** *оператор-IMPLICIT*
или *оператор-PARAMETER*
или *оператор-FORMAT*
или *оператор-ENTRY*

R207 *конструкция-объявления* **есть** *описание-производного-типа*
или *блок-интерфейса*
или *оператор-объявления-типа*
или *оператор-спецификации*
или *оператор-PARAMETER*
или *оператор-FORMAT*

или *оператор-ENTRY*
или *описание-операторной-функции*

R208 *раздел-выполняемых-операторов* **есть** *выполняемая-конструкция*
[*конструкция-раздела-выполняемых-операторов*]...

R209 *конструкция-раздела-выполняемых-операторов* ♦
♦ **есть** *выполняемая-конструкция*
или *оператор-FORMAT*
или *оператор-DATA*
или *оператор-ENTRY*

R210 *раздел-внутренних-процедур* **есть** *оператор-CONTAINS*
внутренняя-процедура
[*внутренняя-процедура*]...

R211 *внутренняя-процедура* **есть** *процедура-функция*
или *процедура-подпрограмма*

R212 *раздел-модульных-процедур* **есть** *оператор-CONTAINS*
модульная-процедура
[*модульная-процедура*]...

R213 *модульная-процедура* **есть** *процедура-функция*
или *процедура-подпрограмма*

R214 *оператор-спецификации* **есть** *оператор-доступности*
или *оператор-ALLOCATABLE*
или *оператор-COMMON*
или *оператор-DATA*
или *оператор-DIMENSION*
или *оператор-EQUIVALENCE*
или *оператор-EXTERNAL*
или *оператор-INTENT*
или *оператор-INTRINSIC*
или *оператор-NAMELIST*
или *оператор-OPTIONAL*
или *оператор-POINTER*
или *оператор-SAVE*
или *оператор-TARGET*

R215 *выполняемая-конструкция* **есть** *оператор-действия*
или *конструкция-CASE*
или *конструкция-DO*
или *конструкция-IF*
или *конструкция-WHERE*

R216 *оператор-действия* **есть** *оператор-ALLOCATE*
или *оператор-BACKSPACE*
или *оператор-CALL*
или *оператор-CLOSE*
или *оператор-CONTINUE*
или *оператор-CYCLE*
или *оператор-DEALLOCATE*
или *оператор-ENDFILE*
или *оператор-EXIT*
или *оператор-GOTO*
или *оператор-вычисляемый-GOTO*
или *оператор-IF*
или *оператор-INQUIRE*
или *оператор-NULLIFY*
или *оператор-OPEN*
или *оператор-PRINT*
или *оператор-READ*

или *оператор-RETURN*
или *оператор-REWIND*
или *оператор-STOP*
или *оператор-WHERE*
или *оператор-WRITE*
или *оператор-конца-программы*
или *оператор-конца-подпрограммы*
или *оператор-конца-функции*
или *оператор-присваивания*
или *оператор-присваивания-указателю*
или *арифметический-оператор-IF*
или *оператор-ASSIGN*
или *оператор-GOTO-по-предписанию*
или *оператор-PAUSE*

Ограничение: *раздел-выполняемых-операторов* не должен содержать *оператор-конца-программы*, *оператор-конца-подпрограммы* или *оператор-конца-функции*. (Термины *раздел-выполняемых-операторов* и *раздел-выполняемых-конструкций* - идентичны.)

2.2. Понятия программной единицы

Программные единицы являются основными компонентами Фортран-программы. **Программная единица** может быть главной программной единицей, внешней процедурой, модулем или программной единицей-блоком данных. Внешняя процедура может быть функцией или подпрограммой. Модуль содержит описания, которые надо сделать доступными для других программных единиц. Программная единица - блок данных используется для задания начальных значений для элементов именованных общих блоков данных. Каждый из видов программных единиц описан позднее в главах 11 или 12. **Внешняя процедура** - это процедура, не входящая в главную программную единицу, в модуль или в другую процедуру. **Внутренняя процедура** - это процедура, входящая в главную программную единицу или в другую процедуру. **Модульная процедура** - это процедура, находящаяся в модуле, но не являющаяся внутренней процедурой.

Программная единица состоит из непересекающихся контекстных областей.

Контекстной областью является:

1) описание производного типа (4.4.1),
2) тело интерфейса процедуры, исключая все описания производного типа и тела интерфейсов процедур, содержащиеся в нем (12.3.2.1), или
3) программная единица или процедура, исключая содержащиеся в ней описания производного типа, тела интерфейсов процедур и процедуры.

Контекстная область, непосредственно объемлющая другую контекстную область, называется **объемлющей контекстной областью**.

2.2.1. Выполняемая программа

Выполняемая программа состоит из одной главной программной единицы и произвольного количества (включая нулевое) программных единиц других видов. Набор программных единиц в выполняемой программе может включать любую комбинацию разных видов программных единиц в произвольном порядке.

2.2.2. Главная программная единица

Главная программная единица описана в разделе 11.1.

2.2.3. Процедура

Процедура содержит произвольную последовательность вычислений, которую можно вызвать непосредственно во время выполнения программы. Процедура - это либо функция, либо подпрограмма. **Функция** - это процедура, которая вызывается из выражения; ее вызов приводит к вычислению значения, которое потом используется в вычислении выражения. Переменная, которая выдает значение функции, называется **результирующей переменной.** **Подпрограмма** - это процедура, которая вызывается с помощью оператора CALL или оператора определяемого присваивания (12.4.4, 7.5.1.3). Подпрограмму можно использовать для изменения состояния программы путем изменения значений некоторых объектов данных, доступных этой подпрограмме; функция может, помимо вычисления значения, делать то же самое.

Процедуры описаны далее в главе 12.

2.2.3.1. Внешняя процедура

Внешняя процедура - это процедура, определяемая либо программной единицей Фортрана, либо нефортрановскими средствами. Внешнюю процедуру можно вызвать из главной программной единицы или из любой процедуры выполняемой программы.

2.2.3.2. Модульная процедура

Модульная процедура - это процедура, описанная в модуле (R213). Модульная процедура может быть вызвана другой процедурой в модуле или любой другой программной единицей, использующей модуль. Модуль, содержащий процедуру, называется **объемлющим контекстом** модульной процедуры.

2.2.3.3. Внутренняя процедура

Внутренняя процедура - это процедура, которая описана как внутренняя (R211). Главная программная единица или процедура, содержащая внутреннюю процедуру, называется **объемлющим контекстом** для внутренней процедуры. Внутренняя процедура локальна внутри своей объемлющей в том смысле, что она доступна внутри контекстных областей объемлющей и всех ее других внутренних процедур, но недоступна вне объемлющей.

2.2.3.4. Блок интерфейса процедуры

Назначение **блока интерфейса процедуры** состоит в описании интерфейсов (12.3) для множества процедур. Он позволяет вызывать их посредством одного обобщенного имени, посредством определяемой операции или определяемого присваивания. Он определяет формы ссылок, посредством которых можно вызывать процедуры.

2.2.4. Модуль

Модуль содержит (или передает из других модулей) описания, которые должны стать доступными для других программных единиц. Эти описания включают объявления объектов данных, описания типов и процедур, блоки интерфейсов процедур. Назначение модуля - сделать содержащиеся в нем описания доступными для всех других программных единиц выполняемой программы, в которых есть необходимость такого доступа. Контекстная область другой программной единицы может запросить доступ к описаниям, содержащимся в модуле. Модули описаны в главе 11.

2.3. Концепции выполнения

Каждый оператор Фортрана классифицируется как **выполняемый оператор** или **невыполняемый оператор**. Имеются ограничения на порядок расположения операторов в программной единице, а некоторые выполняемые операторы могут находиться только в конкретных выполняемых конструкциях.

2.3.1. Выполняемые/невыполняемые операторы

Выполнение программы - это последовательность (во времени) вычислительных действий. Выполняемый оператор - это команда на выполнение одного (или нескольких) из таких действий. Таким образом, выполняемые операторы в программной единице определяют вычислительный процесс в программной единице. Выполняемые операторы - это все операторы, образующие синтаксический класс *выполняемая-конструкция*.

Невыполняемые операторы не специфицируют действий; они используются для формирования программной среды, в которой производятся вычислительные действия. Невыполняемые операторы - это те операторы, которые не относятся к выполняемым. Все операторы в программной единице-блоке данных должны быть невыполняемыми. Модуль может содержать выполняемые операторы только в находящихся в нем процедурах.

2.3.2. Порядок операторов

Синтаксические правила из раздела 2.1 определяют порядок операторов в программных единицах и процедурах. Эти правила проиллюстрированы на рис. 2.1 и в табл. 2.1. Рисунок относится ко всем программным единицам и процедурам и демонстрирует правила упорядочения операторов. Вертикальные линии разделяют множества операторов, которые могут пересекаться, а горизонтальные разделяют множества операторов, которые не могут пересекаться. Операторы USE, если они есть, должны следовать непосредственно за заголовком программной единицы. Внутренние или модульные процедуры должны следовать за оператором CONTAINS. Между операторами USE и CONTAINS невыполняемые операторы должны в общем случае предшествовать выполняемым, хотя невыполняемые операторы DATA, FORMAT и ENTRY могут находиться среди выполняемых операторов.

<table>
<tr><td colspan="3">Оператор PROGRAM, SUBROUTINE, FUNCTION,
MODULE или BLOCK DATA</td></tr>
<tr><td colspan="3">Операторы USE</td></tr>
<tr><td rowspan="4">Операторы
FORMAT
и
ENTRY</td><td colspan="2">IMPLICIT NONE</td></tr>
<tr><td>Операторы
PARAMETER</td><td>Операторы
IMPLICIT</td></tr>
<tr><td>Операторы
PARAMETER
и
DATA</td><td>Описания производного типа,
блоки интерфейса,
операторы объявления типа,
описания операторных функций,
операторы спецификации</td></tr>
<tr><td>Операторы
DATA</td><td>Выполняемые
операторы</td></tr>
<tr><td colspan="3">Оператор CONTAINS</td></tr>
<tr><td colspan="3">Процедуры внутренние или модульные</td></tr>
<tr><td colspan="3">Оператор END</td></tr>
</table>

Рис. 2.1.

Требования к порядку следования операторов

2.3.3. Оператор END

Оператор-конца-программы, оператор-конца-функции, оператор-конца-подпрограммы, оператор-конца-модуля и *оператор-конца-блока-данных* - это все **оператор END**. Каждая программная единица, модульная или внутренняя процедура должна иметь ровно один оператор END. *Оператор-конца-программы, оператор-конца-функции* и *оператор-конца-подпрограммы* - выполняемые операторы и могут быть адресатами операторов ветвления. Выполнение *оператора-конца-программы* вызывает завершение исполнения выполняемой программы. Выполнение *оператора-конца-функции* или *оператора-конца-подпрограммы* эквивалентно выполнению *оператора-RETURN* в процедуре.

Оператор-конца-модуля и *оператор-конца-блока-данных* - невыполняемые операторы.

Табл. 2.1 показывает, какие операторы разрешены в контекстной области.

Таблица 2.1

Операторы, разрешенные в контекстных областях

Вид контекстной области	Главная программная единица	Модуль	Блок данных	Внешняя процедура	Модульная процедура	Внутренняя процедура	Тело интерфейса
USE	да	да	да	да	да	да	да
ENTRY	нет	нет	нет	да	да	нет	нет
FORMAT	да	нет	нет	да	да	да	нет
Описания	да	да	да	да	да	да	да
DATA	да	да	да	да	да	да	нет
Описание производного типа	да	да	нет	да	да	да	да
Блоки интерфейса	да	да	нет	да	да	да	да
Операторная функция	да	нет	нет	да	да	да	нет
Выполняемый оператор	да	нет	нет	да	да	нет	нет
CONTAINS	да	да	нет	да	да	нет	нет

Примечание. Описания - это операторы PARAMETER, IMPLICIT, операторы объявления типа и операторы спецификации. Заметим, что в контекстную область модуля не включаются находящиеся в нем модульные процедуры.

2.3.4. Последовательность выполнения

Исполнение выполняемой программы начинается с первой выполняемой конструкции главной программной единицы. Выполнение главной программной единицы или процедуры включает в себя исполнение выполняемых конструкций ее контекстной области. При вызове процедуры выполнение начинается с первой выполняемой конструкции, следующей после вызываемой точки входа. Выполняемые конструкции исполняются в порядке их появления в главной программной единице или процедуре до тех пор, пока не выполнится оператор STOP, RETURN или оператор END. Из этого правила есть следующие исключения.

1) Выполнение операторов ветвления (8.2) изменяет последовательность исполнения. Эти операторы явно задают новую стартовую точку для последовательности исполнения.

2) Конструкции IF, DO и CASE содержат внутреннюю структуру операторов,и выполнение этих конструкций включает в себя неявное (т.е. автоматическое) внутреннее ветвление. Подробно семантику каждой из этих конструкций см. в главе 8.

3) Альтернативный возврат и спецификаторы END=, ERR= и EOR= могут вызвать ветвление.

Оператору END в главной программной единице или внешней процедуре могут предшествовать внутренние процедуры. При выполнении все такие описания пропускаются.

2.4. Концепции данных

Невыполняемые операторы используются для определения характеристик области данных. Это включает типизацию переменных, объявления массивов и описания новых типов данных.

2.4.1. Тип данных

Тип данных - это именованная категория данных, которая характеризуется множеством значений, способом обозначения этих значений и набором операций, которые интерпретируют эти значения и манипулируют ими. Эта основная концепция подробно рассматривается в разделе 4.1.

Есть две категории типов данных - встроенные типы и производные типы.

2.4.1.1. Встроенный тип

Встроенный тип - это тип, который вместе с операциями описывается неявно и всегда доступен. Встроенные типы - это INTEGER, REAL, COMPLEX, DOUBLE PRECISION, CHARACTER и LOGICAL. Свойства этих встроенных типов описаны в разделе 4.3. Встроенный тип может быть параметризован, и тогда множество значений данных зависит от значений параметров. Такой параметр называется **параметром типа** (4.3). Параметрами типа являются KIND (вид) и LEN (длина).

Параметр вида указывает диапазон десятичного порядка для целого типа (4.3.1.1); десятичную точность, диапазон порядка для вещественного и комплексного типов (4.3.1.2 и 4.3.1.3) и способ представления для символьного и логического типов (4.3.2.1 и 4.3.2.2). **Параметр длины** задает число символов для символьного типа.

2.4.1.2. Производный тип

Производный тип - это тип, который не относится к неявно описанным, а нуждается в описании типа с объявлением компонентов встроенных или других производных типов. Скалярный объект такого производного типа называется **структурой** (5.1.1.7). Единственной встроенной операцией для производных типов является присваивание для согласованных типов (4.4.5). Создание значений каждого производного типа обеспечивается конструкторами структуры (4.4.4). Кроме того, объекты производного типа могут использоваться в качестве аргументов процедуры или в качестве результата функции, могут находиться в списках ввода/вывода. Если для производных типов нужны дополнительные операции, они должны быть предусмотрены в виде описаний процедур.

Производные типы описаны далее в разделе 4.4.

2.4.2. Значения данных

Каждый встроенный тип имеет связанное с ним множество значений, которые могут принимать данные этого типа. Значения для каждого из встроенных типов описаны в разделе 4.3. Поскольку производные типы в конце концов специфицируются в терминах компонентов встроенного типа, значения этих объектов можно определить из описания производного типа и множеств значений встроенных типов.

2.4.3. Данное

Данное - это объект данных, результат вычисления выражения или результат вычисления ссылки на функцию (называемый результатом функции). Данное имеет тип (встроенный или производный) и имеет либо может иметь значение (за исключением неопределенной переменной). Каждое данное имеет размерность и является, таким образом, скаляром или массивом.

2.4.3.1. Объект данных

Объект данных (сокращенно - **объект**) - это константа (4.1.2), переменная (6) или подобъект константы. Тип именованного объекта данных может быть задан явно (5) или неявно (5.3).

Подобъекты - это части некоторых именованных объектов данных, на которые можно ссылаться и определять (только переменные) независимо от других частей. Это касается частей массивов (элементы массива и секции массива), частей символьных строк (подстроки) и частей структур (компоненты). Подобъекты сами по себе являются объектами данных, но ссылаться на них можно только через обозначение подобъекта. Подобъект переменной является переменной. Описаны подобъекты в главе 6.

Объекты, на которые ссылаются по имени:

именованный скаляр	(скалярный объект)
именованный массив	(объект-массив)

Подобъекты, на которые ссылаются по обозначению подобъекта:

элемент массива	(скалярный подобъект)
секция массива	(подобъект-массив)
компонент структуры	(подобъект - скаляр или массив)

2.4.3.1.1. Переменная

Переменная может быть определена или переопределена во время исполнения выполняемой программы.

2.4.3.1.2. Константа

Константа имеет значение и не может быть определена или переопределена во время исполнения выполняемой программы.

Константа с именем называется **именованной константой** и имеет атрибут PARAMETER (5.1.2.1). Константа без имени называется **литеральной константой** (4.3).

2.4.3.1.3. Подобъект константы

Подобъект константы - это часть константы. На какую именно часть ссылка - это может зависеть от значения переменной. Например:

```
CHARACTER (LEN=10),PARAMETER :: DIGITS='0123456789ABCDEF'
CHARACTER (LEN=1)            :: DIGIT
INTEGER :: I
    ...
DIGIT = DIGITS (I:I)
```

DIGITS - именованная константа, а DIGITS(I:I) обозначает подобъект константы DIGITS.

2.4.3.2. Выражение

Выражение во время вычисления вырабатывает данное. Выражение представляет собой либо ссылку на данное, либо вычисление и образуется из операндов, знаков операций и скобок. Тип, значение и размерность результата выражения определяются по правилам из главы 7.

2.4.3.3. Ссылка на функцию

Во время выполнения **ссылки на функцию** (12.4.2) при вычислении выражения вырабатывается данное. Тип и размерность результата функции определяются интерфейсом функции(12.2.2). Значение результата функции определяется при выполнении функции.

2.4.4. Скаляр

Скаляр - это данное, которое не является массивом. Скаляры могут быть любого из встроенных или производных типов. Заметим, что структура является скаляром, даже если имеет компоненты-массивы.

Размерность скаляра равна нулю. Конфигурация скаляра представляется одномерным массивом нулевого размера.

2.4.5. Массив

Массив - это множество данных одного типа и с одинаковыми параметрами типа, отдельные элементы которого размещены по прямоугольной схеме. **Элемент массива** - это один из отдельных элементов этого множества, который сам является скаляром. **Секция массива** - это подмножество элементов массива, которое само является массивом.

Массив может иметь до семи измерений и любую **протяженность** (число элементов) по любому измерению. **Размерность** массива - это число его измерений, его **размер** - это общее число элементов, равное произведению протяженностей. Массив может иметь нулевой размер. **Конфигурация** массива определяется его размерностью и протяженностью по каждому измерению; она может быть представлена одномерным массивом, элементами которого являются протяженности. Все именованные массивы должны быть объявлены, и размерность именованного массива задается в его объявлении. Однажды объявленная, размерность массива является постоянной, постоянными могут быть и протяженности измерений. Однако для массивов - формальных аргументов, автоматически размещаемых массивов, массивов-адресатов и динамически размещаемых массивов протяженности по измерениям могут изменяться во время выполнения.

Два массива **согласованы**, если они имеют одинаковую конфигурацию. Скаляр согласован с любым массивом. Любая операция, определенная для скалярных объектов, может быть применена к согласованным объектам. Такие операции выполняются поэлементно для получения результирующего массива, согласованного с массивами-операндами. Поэлементная операция означает, что соответствующие элементы массивов-операндов включены в "скаляроподобную" операцию для выработки соответствующего элемента результирующего массива и что все такие операции над элементами могут

выполняться в любом порядке или одновременно. Такие операции описываются как поэлементные.

Одномерный массив может быть собран из скаляров и других массивов и может быть преобразован в любую другую допустимую для массива конфигурацию (4.5).

Объекты-массивы могут быть любого встроенного или производного типа и описываются в разделе 6.2.

2.4.6. Указатель

Указатель - это переменная с атрибутом POINTER. Указатель **связан** с адресатом посредством размещения (6.3.1) или присваивания указателю (7.5.2). На него нельзя ссылаться и его нельзя определять, пока он не связан. Указатель становится **не связанным** вследствие выполнения операторов NULLIFY и DEALLOCATE, либо вследствие последующей связи с несвязанным указателем. Не связанный в данный момент указатель не связывается с адресатом (14.6.2). Если указатель объявлен как массив, его размерность объявляется, но протяженности по измерениям определяются в момент связи с адресатом.

2.4.7. Память

Большая часть средств этого Международного стандарта не содержит никаких предположений относительно физических характеристик памяти для объектов данных. Однако программные единицы, которые содержат средства, зависящие от связи по памяти, должны иметь в виду некоторые ограничения на память (14.6.3).

2.5. Основные термины

Ниже определяются некоторые основные термины, которые наиболее часто употребляются в данном Международном стандарте.

2.5.1. Имя и обозначение

Имя используется для идентификации элементов программы, таких, как программная единица, именованная переменная, именованная константа, формальный аргумент или производный тип. Правила образования имен приведены в разделе 3.2.2. **Обозначение подобъекта** - это имя, за которым следует один или несколько селекторов компонента, селекторов секции массива, селекторов элемента массива или селекторов подстроки.

2.5.2. Ключевое слово

Термин **ключевое слово** используется в данном стандарте в двух смыслах. Слова, которые являются частью синтаксиса оператора и могут быть использованы для идентификации оператора, являются **ключевыми словами операторов**. Примерами ключевых слов этого вида являются: IF, READ, WHERE, INTEGER и т.д.

Ключевое слово аргумента - это имя формального аргумента. Глава 13 определяет ключевые слова аргументов для всех встроенных процедур. Ключевые слова аргументов для внешних процедур можно задать в блоке интерфейса процедуры (12.3.2.1).

2.5.3. Объявление

Термин **объявление** относится к спецификации атрибутов различных объектов программы. Часто это включает спецификацию типа данных для именованных объектов данных или спецификацию конфигурации именованного объекта-массива.

2.5.4. Определение и описание[1)]

Термины **определение** и **описание** (definition) используются в двух случаях. Во-первых, когда объект данных получает во время выполнения программы правильное значение, говорят, что он становится **определенным**. Часто это происходит в результате выполнения оператора присваивания или оператора ввода. При некоторых обстоятельствах переменная не имеет предсказуемого значения, и тогда говорят, что она **не определена**. В главе 14 приводятся ситуации, когда переменные могут стать определенными или неопределенными. Второе использование термина definition - **описание** - относится к описанию производных типов и процедур.

2.5.5. Ссылка

Ссылка на объект данных - это появление имени объекта данных или обозначения подобъекта данных в точке контекста, где требуется его значение во время выполнения программы.

Ссылка на процедуру - это появление имени процедуры, знака ее операции или символа присваивания в контексте, который требует в данной точке исполнения этой процедуры.

Появление имени объекта данных, обозначения подобъекта или имени процедуры в списке фактических аргументов не является ссылкой на этот объект, подобъект или процедуру, кроме случаев, когда такая ссылка нужна для завершения спецификации фактического аргумента.

Ссылка на модуль - это появление имени модуля в операторе USE (11.3.1).

2.5.6. Связь

Связь может быть связью по имени (14.6.1), по указателю (14.6.2) или по памяти (14.6.3). Связь по имени может быть связью аргументов, связью по объемлющему контексту или по использованию. Связь по памяти приводит к тому, что разные объекты используют одну и ту же память. Любая связь позволяет один и тот же объект идентифицировать либо разными именами в одной контекстной области, либо разными или одинаковыми именами в разных контекстных областях.

2.5.7. Встроенный

Определение **встроенный** означает, что термин, к которому он относится, определен в данном Международном стандарте. Он применяется к типам данных, процедурам и операциям. Все они могут быть использованы в контекстной области без дальнейшего описания или спецификации.

2.5.8. Знак операции

Знак операции определяет конкретное вычисление, включающее одно (унарная операция) или два (бинарная операция) значения данных (операнды). Фортран содержит ряд встроенных операций (например, арифметические операции +, -, *, / и ** над числовыми операндами или логические операции .AND. и .OR. и т.д. над логическими операндами). В выполняемой программе могут быть определены и дополнительные операции (7.1.3).

---

1) В английском тексте для этих двух разных понятий используется один и тот же термин - definition (*Прим. перев. и ред.*).

2.5.9. Последовательность

Последовательность - это множество, упорядоченное во взаимно-однозначном соответствии с числами 1, 2, ..., N. Число элементов последовательности - N. Последовательность может быть пустой, тогда она не содержит элементов.

На элементы непустой последовательности ссылаются как на первый элемент, на второй элемент и т.д. Последним элементом называется N-й элемент, где N - число элементов в последовательности. В пустой последовательности нет ни первого, ни последнего элемента.

Глава 3. СИМВОЛЫ, ЛЕКСИЧЕСКИЕ ЭЛЕМЕНТЫ И ФОРМА ИСХОДНОГО ТЕКСТА

В этой главе описываются набор символов Фортрана и различные элементы лексики, такие, как имена и операции. Кроме того, в главе описываются правила построения Фортран-программы.

3.1. Набор символов процессора

Набор символов процессора зависит от процессора. Структуру такого набора составляют следующие символы:

1) **управляющие символы** (например, 'newline') и
2) **графические символы**:
   - буквы (3.1.1);
   - цифры (3.1.2);
   - символ подчеркивания (3.1.3);
   - специальные символы (3.1.4);
   - другие символы (3.1.5).

Буквы, специальные символы, цифры и символ подчеркивания образуют **набор символов Фортрана**.

R301 *символ* **есть** *буквенно-цифровой-символ*
или *специальный-символ*

R302 *буквенно-цифровой-символ* **есть** *буква*
или *цифра*
или *подчеркивание*

Изображение символов (кроме денежного символа) должно быть таким, как в 3.1.1, 3.1.2, 3.1.3 и 3.1.4. Однако шрифт для любого из символов не определяется.

3.1.1. Буквы

Двадцать шесть **букв** - это следующие символы:

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

Набор букв определяет синтаксический класс *буква*. Если процессор допускает еще и строчные буквы, они в программных единицах эквивалентны заглавным везде, кроме символьного контекста (3.3).

3.1.2. Цифры

Десять **цифр** - это

0 1 2 3 4 5 6 7 8 9

Десять цифр определяют синтаксический класс *цифра*.

3.1.3. Символ подчеркивания

R303 *подчеркивание* **есть** _

Символ подчеркивания может быть использован как значащий символ в имени.

3.1.4. Специальные символы

Двадцать один **специальный символ** приведен в табл. 3.1.

Таблица 3.1

Специальные символы

Символ	Название символа	Символ	Название символа
	пробел	:	двоеточие
=	равно	!	восклицательный знак
+	плюс	"	кавычки
-	минус	%	процент
*	звездочка	&	амперсанд
/	дробная черта	;	точка с запятой
(	левая скобка	<	меньше чем
)	правая скобка	>	больше чем
,	запятая	?	вопросительный знак
.	десятичная точка	$	денежный символ
'	апостроф		

Двадцать один специальный символ определяют синтаксический класс *специальный символ*. Специальные символы используются в знаках операций, для заключения в скобки и различных форм разделения и ограничения лексических элементов. Использование специальных символов $ и ? не специфицируется.

3.1.5. Другие символы

В процессоре могут быть представимы и дополнительные символы, но они могут находиться только в символьных константах, описателях редактирования "символьная строка", в комментариях и записях ввода/вывода (4.3.2.1, 10.2.1, 3.3.1.1, 3.3.2.1, 9.1.1).

Стандартный символьный тип должен поддерживать набор символов, включающий алфавит Фортрана. Другие наборы символов могут поддерживаться процессором для нестандартных символьных типов. Символы, допустимые в таких типах, не описываются, кроме одного символа в каждом нестандартном символьном типе, который обозначается как пробел, используемый для заполнения записей.

3.2. Синтаксис нижнего уровня

Синтаксис нижнего уровня описывает основные лексические элементы программной единицы. **Лексические элементы** - это последовательности символов с неделимой интерпретацией, из которых строится программа. Это ключевые слова, имена, литеральные константы (кроме комплексных литеральных констант), знаки операций, метки, ограничители, запятая, =, =>, :, ::, ; и %.

3.2.1. Ключевые слова

Ключевые слова в синтаксических правилах глав 4-12 изображаются в их английском написании заглавными буквами.

3.2.2. Имена

Имена используются для различных языковых объектов, таких, как переменные, программные единицы, формальные аргументы, именованные константы и производные типы.

R304 *имя* **есть** *буква* [*буквенно-цифровой-символ*]...

Ограничение: максимальная длина *имени* - 31 символ.

Примеры имен:

```
A1
NAME_LENGTH          (один символ подчеркивания)
S_P_R_E_A_D__O_U_T   (два последовательных подчеркивания)
TRAILER_             (подчеркивание в конце имени)
```

3.2.3. Константы

R305 *константа* **есть** *литеральная-константа*
или *именованная-константа*

R306 *литеральная-константа* **есть** *целая-литеральная-константа*
или *вещественная-литеральная-константа*
или *комплексная-литеральная-константа*
или *логическая-литеральная-константа*
или *символьная-литеральная-константа*
или *BOZ-литеральная-константа*

R307 *именованная-константа* **есть** *имя*

R308 *целая-константа* **есть** *константа*

Ограничение: *целая-константа* должна быть целого типа.

R309 *символьная-константа* **есть** *константа*

Ограничение: *символьная-константа* должна быть символьного типа.

3.2.4. Знаки операций

R310 *знак-встроенной-операции* **есть** *оп-степени*
или *оп-умножения*
или *оп-сложения*
или *оп-конкатенации*
или *оп-отношения*
или *оп-отрицания*
или *оп-конъюнкции*
или *оп-дизъюнкции*
или *оп-эквивалентности*

R708 *оп-степени* **есть** **

R709 *оп-умножения* **есть** *
или /

R710 *оп-сложения* **есть** +
или -

R712 *оп-конкатенации* **есть** //

R714 *оп-отношения* **есть** .EQ.
или .NE.
или .LT.
или .LE.
или .GT.
или .GE.
или ==
или /=
или <

или <=
или >
или >=

R719 *оп-отрицания* **есть** .NOT.

R720 *оп-конъюнкции* **есть** .AND.

R721 *оп-дизъюнкции* **есть** .OR.

R722 *оп-эквивалентности* **есть** .EQV.
или .NEQV.

R311 *знак-определяемой-операции* **есть** *знак-определяемой-унарной-операции*
или *знак-определяемой-бинарной-операции*
или *знак-расширенной-встроенной-операции*

R704 *знак-определяемой-унарной-операции* **есть** . *буква*[*буква*]... .

R724 *знак-определяемой-бинарной-операции* **есть** . *буква*[*буква*]... .

R312 *знак-расширенной-встроенной-операции* **есть** *знак-встроенной-операции*

Ограничение: *знак-определяемой-унарной-операции* и *знак-определяемой-бинарной-операции* не должны содержать больше 31 символа и не могут совпадать со знаком какой-либо встроенной операции или с *логической-литеральной-константой*.

3.2.5. Метки операторов

Метка оператора предоставляет возможность ссылаться на конкретный оператор.

R313 *метка* **есть** *цифра*[*цифра*[*цифра*[*цифра*[*цифра*]]]]

Ограничение: по крайней мере одна цифра в метке должна быть отличной от нуля.

Если оператор помечен, он должен содержать хотя бы один отличный от пробела символ. Нельзя одной и той же меткой пометить несколько операторов в одной контекстной области. При различении меток операторов не являются значащими стоящие впереди нули и пробелы. Например:

```
99999
 10
010
```

Это все метки оператора. Последние две эквивалентны.

3.2.6. Ограничители

Ограничители используются для выделения синтаксических списков. Следующие пары являются ограничителями:

```
 ( ... )
 / ... /
(/ ... /)
```

3.3. Форма исходного текста

Программная единица Фортрана - это последовательность из одного или нескольких операторов Фортрана, комментариев и строк INCLUDE. **Оператор** Фортрана - это последовательность из одной или нескольких полных или неполных строк. **Строка** - это упорядоченная последовательность из некоторого количества (в том числе, и нулевого) символов. Строки, следующие за оператором END в программной единице, не являются частью этой программной единицы.

Термин **символьный контекст** означает символы внутри символьной литеральной константы (4.3.2.1) или внутри описателей редактирования "символьная строка" (10.7).

Комментарий может содержать любой символ, который может находиться в любом символьном контексте.

Есть две **формы исходного текста** - свободная и фиксированная. В одной и той же программной единице нельзя перемежать свободную форму с фиксированной. Способ задания формы исходного текста программной единицы является процессорно-зависимым.

3.3.1. Свободная форма

В **свободной форме исходного текста** каждая строка может содержать от 0 до 132 символов, и нет никаких ограничений на положение оператора (или его части) внутри строки. Однако, если какой-либо оператор в строке содержит символ, параметр вида которого отличается от стандартного (4.3.2.1), число разрешенных в строке символов становится процессорно-зависимым.

В свободной форме символ пробела не может находиться внутри лексемы (кроме символьной константы или описателя редактирования "символьная строка"). Пробелы можно свободно вставлять между лексемами для повышения читабельности, например, пробелы могут находиться между частями комплексной литеральной константы. Вне символьного контекста последовательность пробелов эквивалентна одному пробелу.

Пробелы должны использоваться для отделения имен, констант или меток от соседних ключевых слов, имен, констант или меток. В примере

```
REAL X
READ 10
30 DO K=1,3
```

пробелы обязательны после REAL, READ, 30 и DO.

Между некоторыми из смежных ключевых слов один или несколько пробелов обязательны, а между некоторыми - не обязательны.

Необязательные пробелы	Обязательные пробелы
BLOCK DATA	CASE DEFAULT
DOUBLE PRECISION	DO WHILE
ELSE IF	IMPLICIT *спецификатор-типа*
END BLOCK DATA	IMPLICIT NONE
END DO	INTERFACE ASSIGNMENT
END FILE	INTERFACE OPERATOR
END FUNCTION	MODULE PROCEDURE
END IF	RECURSIVE FUNCTION
END INTERFACE	RECURSIVE SUBROUTINE
END MODULE	RECURSIVE *спецификатор-типа*
END PROGRAM	*спецификатор-типа* FUNCTION
END SUBROUTINE	
END TYPE	
END WHERE	
GO TO	
IN OUT	
SELECT CASE	

3.3.1.1. Комментарий в свободной форме

Вне символьного контекста символ "!" означает начало **комментария**. Комментарий продолжается до конца строки. Если "!" является первым отличным от пробела символом в строке, строка называется строкой комментария. Строки, содержащие только пробелы или вообще не содержащие символов, тоже являются строками комментария. Комментарии могут располагаться в любом месте программной

единицы и могут предшествовать ее первому оператору. На интерпретацию программной единицы комментарии не влияют.

3.3.1.2. Разделение операторов в свободной форме

Вне символьного контекста и комментария символ ";" разделяет операторы или части операторов, расположенные на одной строке исходного текста. Если после разделителя ";" следует какое-то число пробелов (в том числе, нулевое) и один или несколько разделителей ";", вся последовательность, начиная с первого ";" и кончая последним ";", рассматривается как один разделитель ";". Разделитель ";", который является последним отличным от пробела символом в строке или последним отличным от пробела символом перед комментарием, игнорируется.

3.3.1.3. Продолжение оператора в свободной форме

Символ "&" используется для обозначения того, что текущий оператор будет продолжен в следующей строке, которая не является строкой комментария. Строки комментария не могут быть продолжены; символ "&" в комментарии не оказывает действия. Комментарии могут находиться внутри продолженного оператора. Используемый для продолжения символ "&" не является частью оператора. Строка не может состоять из одного символа "&" как единственного отличного от пробела символа в строке или как единственного отличного от пробела символа перед "!".

3.3.1.3.1. Продолжение несимвольного контекста

Если "&", не входящий в комментарий, является последним отличным от пробела символом в строке или перед "!", оператор продолжится на ближайшей строке, которая не является строкой комментария. Если на этой строке первый отличный от пробела символ тоже "&", тогда оператор продолжится с символьной позиции, следующей за "&", в противном случае он продолжается с символьной позиции 1 на следующей отличной от комментария строке.

Если конец строки разрывает лексему, первым отличным от пробела символом в первой отличной от комментария строке должен быть "&", за которым непосредственно следует продолжение лексемы.

3.3.1.3.2. Продолжение символьного контекста

Если надо продолжить символьный контекст, "&" должен быть последним отличным от пробела символом в строке и не может быть продолжен комментарием. Символ "&" должен быть первым отличным от пробела символом в следующей строке, которая не является строкой комментария, а оператор продолжается с символа, следующего за "&".

3.3.1.4. Операторы в свободной форме

Метка может предшествовать любому оператору, который не является частью другого оператора. Заметим, что оператор Фортрана не может начинаться с цифры. Оператор в свободной форме может иметь не более 39 строк продолжения.

3.3.2. Фиксированная форма

В **фиксированной форме исходного текста** имеются ограничения на расположение оператора в строке. Если в исходной строке все символы относятся к стандартному символьному типу, исходная строка должна содержать ровно 72 символьные позиции; в противном случае максимальное число символов в строке зависит от процессора.

Исключая символьный контекст, пробелы являются незначащими и могут свободно располагаться в любом месте программы.

3.3.2.1. Комментарии в фиксированной форме

Символ "!", не входящий в символьный контекст и не расположенный в шестой позиции, обозначает начало комментария, который продолжается до конца строки. Если первый отличный от пробела символ в строке - "!", эта строка является строкой комментария. Строки с символами "C" или "*" в первой позиции или состоящие из одних пробелов тоже являются строками комментария. Комментарий может находиться в любом месте программной единицы, в том числе, перед ее первым оператором. На интерпретацию программной единицы комментарии не влияют.

3.3.2.2. Разделение операторов в фиксированной форме

Вне символьного контекста и комментария символ ";" разделяет операторы или части операторов, расположенные на одной строке исходного текста. Если после разделителя ";" следует какое-то число пробелов (в том числе, нулевое) и один или несколько разделителей ";", вся последовательность, начиная с первого ";" и кончая последним ";", рассматривается как один разделитель ";". Разделитель ";", который является последним отличным от пробела символом в строке или последним отличным от пробела символом перед комментарием, игнорируется.

3.3.2.3. Продолжение оператора в фиксированной форме

Позиция 6, если она не попадает в комментарий, используется для обозначения продолжения. Если она содержит нуль или пробел, данная строка является начальной строкой нового оператора, который начинается с позиции 7. Если в позиции 6 находится символ, отличный от нуля или пробела, позиции 7-72 данной строки образуют продолжение предыдущей строки (не комментария).

Заметим, что символы "!" и ";" в позиции 6 обозначают продолжение предыдущей отличной от комментария строки. Строки комментария нельзя продолжать. Строки комментария могут находиться внутри продолженного оператора.

3.3.2.4. Операторы в фиксированной форме

Если есть метка, она должна находиться только в символьных позициях 1-5 первой строки оператора, если метки нет, позиции 1-5 должны быть пусты. Внутри метки могут находиться пробелы. Заметим, что оператор, следующий за ";" в той же строке, не может быть помечен. Символьные позиции 1-5 строк продолжения должны быть пусты. Оператор в фиксированной форме не может иметь более 19 строк продолжения. Оператор END нельзя продолжать, и ни один оператор в программной единице не может иметь начальную строку, которая воспринималась бы как оператор END.

3.4. Включение исходного текста

При обработке исходного текста в него может быть включен дополнительный текст при помощи **строк INCLUDE** следующего вида:

INCLUDE *символьная-литеральная-константа*

Значением параметра вида для *символьной-литеральной-константы* не может быть *именованная-константа*.

Строка INCLUDE не является оператором Фортрана.

Строка INCLUDE должна находиться на отдельной строке входного текста, в которой может появиться оператор; на этой строке должен быть только отличный от пробелов текст, кроме, может быть, конечных пробелов. Таким образом, метка оператора не разрешается.

Действует строка INCLUDE, как если бы исходный текст, на который она ссылается, физически заменил саму строку до обработки программы. Включаемый текст может содержать любой текст, в том числе и строки INCLUDE. Максимальная глубина вложенности любых вложенных строк INCLUDE зависит от процессора. Включение исходного текста, на который ссылается строка INCLUDE, не должно (на любом уровне вложенности) приводить к включению этого же исходного текста.

Когда строка INCLUDE уже раскручена, первая включаемая операторная строка не может быть строкой продолжения, а последняя из включаемых операторных строк не может быть продолжена.

Интерпретация *символьной-литеральной-константы* зависит от процессора. Примером возможной правильной интерпретации может быть задание через *символьную-литеральную-константу* имени файла, содержащего включаемый текст.

Глава 4. ВСТРОЕННЫЕ И ПРОИЗВОДНЫЕ ТИПЫ ДАННЫХ

Фортран предоставляет некоторое абстрактное средство классификации данных безотносительно к конкретному физическому представлению. Этим абстрактным средством является концепция **типа данных**. Каждый тип данных имеет имя. Имена встроенных типов определяются языком, имена производных типов должны быть определены в описании типа (4.4.1). Тип данных характеризуется множеством значений, способом обозначения значений, множеством операций для действий над ними и интерпретацией значений.

Например, логический тип данных имеет множество из двух значений, обозначаемых лексемами .TRUE. и .FALSE., и действия над ними выполняются с помощью логических операций.

Примером менее ограниченного типа данных является целый тип данных. Этот тип данных имеет процессорно-зависимое множество целочисленных значений, каждое из которых обозначается необязательным знаком, за которым следует строка цифр; действия над ними выполняются с помощью операций целой арифметики и операций отношения.

Способ изображения значения указывает как на тип значения, так и на конкретного члена множества значений, характеризующих этот тип. Встроенные типы данных могут быть параметризованы. В этом случае множество значений ограничивается значением параметра или параметров. Например, символьный тип данных имеет параметр длины, который ограничивает множество символьных значений только теми, длина которых равна значению параметра.

Встроенный тип - это предопределенный языком тип. Встроенные типы - это целый, вещественный, комплексный, логический и символьный. Выражение "предопределен", используемое далее в этой главе, означает "предопределенность" именно в этом смысле.

Помимо встроенных типов, могут быть созданы специальные типы для приложений. Объекты производных типов имеют **компоненты**. Каждый из компонентов может быть встроенного или производного типа. Описание типа (4.4.1) требует задания имени типа, имен и типов его компонентов. Например, если бы комплексный тип не был встроенным и потребовалось бы его определить, надо было бы задать имя "complex" и описать два вещественных компонента.

Предоставляются способы для записи значений производного типа (4.4.4) и для определения операций, которые можно использовать для действий над объектами производного типа (4.4.5). Производный тип должен быть описан в выполняемой программе, тогда как встроенный тип предопределен. Использовать производный тип можно только там, где доступно его описание (4.4.1). Встроенный тип доступен всегда.

4.1. Концепция типа данных

Тип данных имеет:

1) имя;
2) множество правильных значений;
3) способ изображения таких значений (константы);
4) множество операций для действий над этими значениями.

4.1.1. Множество значений

Для каждого типа данных есть множество правильных значений. Оно может быть полностью определено, как в случае логического типа, или может быть определено каким-либо процессорно-зависимым способом, как в случае целого или вещественного типов. Для комплексного типа и производных типов множество правильных значений образуется из множества всех возможных комбинаций значений отдельных компонентов.

Для параметризованных типов множество правильных значений зависит от значений параметров.

4.1.2. Константы

Для каждого из встроенных типов данных синтаксис литеральных констант специфицирован в данном стандарте. Эти литеральные константы для каждого встроенного типа описываются в 4.3. В выполняемой программе все литеральные константы одинакового вида имеют одинаковое значение.

Константному значению можно дать имя (5.1.2.1, 5.2.10).

Константное значение производного типа можно сконструировать (4.4.4) из правильной последовательности константных выражений (7.1.6.1) с помощью конструктора структуры. Такое константное значение рассматривается как скаляр, несмотря на возможное наличие у него компонентов-массивов.

4.1.3. Операции

Для каждого из встроенных типов данных множество операций и соответствующих знаков операций заранее определено. Они описываются в главе 7. Встроенное множество может быть дополнено операциями и знаками операций, определяемыми функциями с интерфейсом OPERATOR (12.3.2.1). Описания операций приведены в главах 7 и 12.

Единственной встроенной операцией для производных типов данных является присваивание. Все остальные операции над объектами производного типа должны быть определены в выполняемой программе (4.4.5).

4.2. Взаимосвязь типов и значений объектов

Имя типа данных служит спецификатором типа и может быть использовано для объявления объектов этого типа. Объявление специфицирует тип именованного объекта. Объект данных может быть объявлен явно или неявно. Однажды описав производный тип, можно объявлять объекты этого типа. Объекты данных могут иметь, помимо типов, другие атрибуты. В главе 5 приведены способы объявления объектов и способы задания их типов и других атрибутов.

Массив - это набор подобъектов.

Скалярные данные любого встроенного или производного типа могут быть сгруппированы по прямоугольной схеме для образования массива элементов одинакового типа с одинаковыми параметрами типа. Массив является объектом и так же, как скалярные объекты, обладает типом и параметрами типа.

Скалярный объект производного типа называют **структурой**. Компоненты структуры являются подобъектами.

Переменные могут быть объектами или подобъектами. Тип переменной определяет, какие значения может она принимать. Одним из способов определения или переопределения переменной любого типа является присваивание. Присваивание предопределено для всех типов, когда тип, параметры типа и конфигурация переменной и присваиваемого ей значения идентичны. Присваивание объектов конкретных встроенных типов, отличающихся по типу, параметрам типа и конфигурации, описывается в главе 7. Например, присваивание целого значения вещественной переменной предопределено. Подпрограмма (7.5.1.3) и блок интерфейса ASSIGNMENT (12.3.2.1) описывают присваивание, которое не является предопределенным или переопределяет встроенное присваивание для производного типа.

Тип переменной определяет операции, которые можно использовать для действий над ней.

4.3. Встроенные типы данных

К встроенным типам данных относятся:

- числовые типы: целый, вещественный и комплексный;
- нечисловые типы: символьный и логический.

4.3.1. Числовые типы

Числовые типы предназначены для числовых вычислений. Для них предопределены обычные операции арифметики: сложение (+), вычитание (-), умножение (*), деление (/), возведение в степень (**), изменение знака (унарный -) и идентичность (унарный +).

Каждый числовой тип включает нулевое значение, которое рассматривается как не принадлежащее ни к отрицательным, ни к положительным. Значение нуля со знаком то же, что у нуля без знака. В данном стандарте неуточненный термин "литеральная константа" в применении к числовым типам означает "литеральная константа без знака".

4.3.1.1. Тип целый

Множество значений **целого типа** является подмножеством математических целых. Процессор должен обеспечивать один или несколько **способов представления**, которые определяют множество значений для данных целого типа. Каждый такой способ характеризуется значением параметра типа, называемым параметром **вида**. Этот параметр вида для конкретного способа представления выдается с помощью встроенной справочной функции KIND (13.13.51). С помощью встроенной функции SELECTED_INT_KIND (13.13.92) выдается значение, обеспечивающее наименьший диапазон среди значений вида, обеспечивающих данный диапазон. Диапазон десятичного порядка для какого-либо конкретного способа представления выдается с помощью встроенной справочной функции RANGE (13.13.85).

Спецификатором типа (R502) для целых является ключевое слово INTEGER.

Если параметр вида не задан, значением вида по умолчанию является KIND(0), и тогда объект данных относится к **стандартному целому** типу.

Любое целое значение может быть представлено как *целая-литеральная-константа-со-знаком*.

R401 строка-цифр-со-знаком **есть** [*знак*] *строка-цифр*

R402 строка-цифр **есть** *цифра* [*цифра*]...

R403 целая-литеральная-константа-со-знаком ♦

♦ **есть** [знак] целая-литеральная-константа

R404 целая-литеральная-константа **есть** *строка-цифр* [_*параметр-вида*]

R405 параметр-вида **есть** *строка-цифр*

или *имя-скалярной-целой-константы*

R406 знак **есть** +

или -

Ограничение: значение *параметра-вида* должно быть неотрицательным.

Ограничение: значение *параметра-вида* должно задавать существующий в процессоре способ представления.

Необязательный параметр вида, следующий за *строкой-цифр*, задает параметр вида для целой константы; если его нет, константа является константой стандартного целого типа.

Примеры целых литеральных констант со знаком и без знака:

```
 473
+56
-101
21_2
21_SHORT
1976354279568241_8
```

где SHORT - именованная целая скалярная константа, значение которой должно быть допустимым значением параметра вида для целого типа.

Целая константа интерпретируется как целое значение.

В операторе DATA (5.2.9) целой скалярной переменной может соответствовать двоичная, восьмеричная или шестнадцатеричная константа.

R407 *BOZ-литеральная-константа* **есть** *двоичная-константа*
или *восьмеричная-константа*
или *шестнадцатеричная-константа*

Ограничение: *BOZ-литеральная-константа* может находиться только в операторе DATA.

R408 *двоичная-константа* **есть** B'*цифра* [*цифра*]...'
или B"*цифра* [*цифра*]..."

Ограничение: *цифра* может иметь значения только 0 или 1.

R409 *восьмеричная-константа* **есть** O'*цифра* [*цифра*]...'
или O"*цифра* [*цифра*]..."

Ограничение: *цифры* могут иметь значения от 0 до 7.

R410 *шестнадцатеричная-константа* ♦
♦ **есть** Z'*шестнадцатеричная-цифра* [*шестнадцатеричная-цифра*]...'
или Z"*шестнадцатеричная-цифра* [*шестнадцатеричная-цифра*]..."

R411 *шестнадцатеричная-цифра* **есть** *цифра*
или A
или B
или C
или D
или E
или F

В этих константах двоичные, восьмеричные и шестнадцатеричные цифры интерпретируются в соответствии с нужной системой счисления. Если процессор поддерживает строчные буквы, в исходном тексте *шестнадцатеричные-цифры* от A до F могут быть представлены своими строчными эквивалентами.

4.3.1.2. Вещественный тип

Вещественный тип имеет значения, аппроксимирующие математические вещественные числа. Процессор должен обеспечивать не менее двух **способов аппроксимации**, которые определяют множества значений для данных вещественного типа. Каждый из этих способов имеет **способ представления** и характеризуется значением параметра типа, называемого параметром **вида**. Значение параметра вида для каждого способа аппроксимации выдается с помощью справочной встроенной функции KIND (13.13.51). С помощью встроенной функции SELECTED_REAL_KIND (13.13.93) выдается то значение, которое обеспечивает наименьшую десятичную точность среди значений вида, обеспечивающих нужную точность и нужный диапазон порядка. Десятичную точность и диапазон десятичного порядка для способа аппроксимации можно выдать при помощи встроенных функций PRECISION (13.13.79) и RANGE (13.13.85).

Спецификатором типа для вещественного типа является ключевое слово REAL, а для вещественных двойной точности - ключевое слово DOUBLE PRECISION.

Если специфицировано ключевое слово REAL, а параметр вида не задан, значение вида по умолчанию есть KIND (0.0), а объект данных относится к **стандартному**

вещественному типу. Если задано ключевое слово DOUBLE PRECISION, нельзя задавать параметр вида, а объект данных относится к **вещественным двойной точности**. Значением параметра вида для такого объекта является KIND (0.0D0). Десятичная точность способа аппроксимации для двойной точности должна быть больше, чем для стандартного вещественного типа.

R412 *вещественная-литеральная-константа-со-знаком* ♦

♦ **есть** [*знак*] *вещественная-литеральная-константа*

R413 *вещественная-литеральная-константа* ♦

♦ **есть** *значащая-часть* [*буква-экспоненты экспонента*] [_*параметр-вида*]
или *строка-цифр буква-экспоненты экспонента* [_*параметр-вида*]

R414 *значащая-часть* **есть** *строка-цифр* . [*строка-цифр*]
или . *строка-цифр*

R415 *буква-экспоненты* **есть** E
или D

R416 *экспонента* **есть** *строка-цифр-со-знаком*

Ограничение: если присутствует и *параметр-вида*, и *буква-экспоненты*, буквой экспоненты должна быть E.

Ограничение: значение *параметра-вида* должно задавать способ аппроксимации, существующий в процессоре.

Вещественные литеральные константы без параметра вида являются константами стандартного вещественного типа, если они записаны без экспонентной части или с буквой экспоненты E, и константами вещественными двойной точности, если с буквой экспоненты D. Вещественная литеральная константа с параметром вида является вещественной константой с заданным параметром вида.

Экспонента представляет степень десяти, на которую умножается число, стоящее перед буквой-экспоненты. Смысл этих констант такой же, как в научной десятичной нотации.

Значащая часть может быть записана с большим количеством цифр, чем используется процессором для аппроксимации значения константы.

Примеры вещественных литеральных констант со знаком:

```
-12.78
+1.6E3
2.1
-16.E4_8
0.45E-4
10.93E7_QUAD
.123
3E4
```

В 10.93E7_QUAD именованная константа QUAD должна быть определена, и ее значение должно быть правильным параметром вида для этого вещественного типа.

4.3.1.3. Комплексный тип

Комплексный тип имеет значения, аппроксимирующие математические комплексные числа. Значения комплексного типа являются упорядоченными парами вещественных значений. Первое вещественное значение называется **действительной частью**, второе - **мнимой частью**.

Каждый способ аппроксимации, используемый для представления объектов данных вещественного типа, должен быть доступен для представления действительной и мнимой частей объектов данных комплексного типа. Для комплексного объекта может быть задан параметр **вида**, который обеспечивает выбор способа аппроксимации для обеих частей комплексного объекта, действующий по тем же правилам, что и для вещественных чисел.

Спецификатором комплексного типа является ключевое слово COMPLEX. Для комплексных двойной точности нет ключевого слова. Если есть ключевое слово COMPLEX и не задан параметр вида, значение параметра вида то же, что для стандартных вещественных, обе части имеют стандартный вещественный тип, а комплексный объект данных относится к **стандартному комплексному** типу.

R417 *комплексная-литеральная-константа* **есть** (*действит-часть*, *мнимая-часть*)

R418 *действит-часть* **есть** *целая-литеральная-константа-со-знаком*
или *вещественная-литеральная-константа-со-знаком*

R419 *мнимая-часть* **есть** *целая-литеральная-константа-со-знаком*
или *вещественная-литеральная-константа-со-знаком*

Если действительная и мнимая части комплексной литеральной константы являются вещественными, параметром вида для всей константы является значение параметра вида той ее части, у которой больше десятичная точность; если точность одна и та же, берется значение одной из частей, как определено процессором. Если у одной из частей и у всей константы разные значения параметров вида, эта часть преобразуется к способу аппроксимации данной комплексной литеральной константы.

Если действительной и мнимой частями являются целые литеральные константы со знаком, они преобразуются к способу аппроксимации для стандартного вещественного типа, и константа относится к стандартному комплексному типу. Если только одна из частей является целой литеральной константой со знаком, она преобразуется к способу аппроксимации, выбранному для вещественной литеральной константы, и значением параметра вида для комплексной литеральной константы является значение параметра для вещественной литеральной константы со знаком.

Примеры комплексной литеральной константы:

```
(1.0, -1.0)
(3, 3.1E6)
(4.0_4, 3.6E7_8)
```

4.3.2. Нечисловые типы

Нечисловые типы используются в нечисловой обработке информации. Встроенные операции, определенные для каждого из этих типов, приведены ниже.

4.3.2.1. Символьный тип

Символьный тип имеет множество значений, образующих символьные строки. **Символьная строка** - это последовательность символов, пронумерованных слева направо числами 1,2,3,... до количества символов в строке. Количество символов в строке называется **длиной** строки. Длина - это параметр типа, его значение должно быть больше нуля, или равно нулю. Строки разной длины все относятся к символьному типу.

Процессор должен обеспечить один или несколько **способов представления**, определяющих множество значений символьного типа. Каждый такой способ характеризуется значением параметра типа, именуемого параметром **вида**. Значение параметра вида для какого-либо способа представления выдается с помощью встроенной справочной функции KIND (13.13.51). Любой символ конкретного способа представления, допустимого в процессоре, может находиться в символьной строке, имеющей тот же способ представления.

Если параметр вида не задан, значением вида по умолчанию является KIND('A'), а объект данных относится к **стандартному символьному** типу.

Спецификатором типа для символьного типа является ключевое слово CHARACTER.

Символьная литеральная константа записывается как последовательность символов, заключенных в апострофы или двойные кавычки.

R420 *символьная-литеральная-константа* ♦

- **есть** [*параметр-вида* _] ' [*представимый-символ*]... '
  или [*параметр-вида* _] " [*представимый-символ*]... "

Ограничение: значение *параметра-вида* должно задавать способ представления, существующий в процессоре.

Необязательный параметр вида, предшествующий первому ограничителю, задает параметр вида для символьной константы. Если его нет, константа относится к стандартному символьному типу.

Для символьного типа с заданным параметром вида или для стандартного символьного типа *представимый-символ* - это один из перечисленных ниже вариантов.

1. Любой символ из процессорно-зависимого набора символов в фиксированной форме исходного текста. Процессор может ограничить появление некоторых (или всех) управляющих символов.
2. Любой графический символ из процессорно-зависимого множества в свободной форме исходного текста.

Ограничивающие апостроф или кавычки не являются частью значения символьной литеральной константы.

Символ апостроф внутри символьной константы, ограниченной апострофами, представляется двумя последовательными апострофами (без промежуточных пробелов); в этом случае сдвоенные апострофы считаются как один символ. Аналогично символ двойные кавычки внутри символьной константы, ограниченной двойными кавычками, представляется сдвоенным символом двойные кавычки (без промежуточных пробелов); в этом случае сдвоенный символ считается как один.

Вне символьного контекста литеральная символьная константа нулевой длины представляется двумя смежными апострофами (без пробелов между ними) или сдвоенными двойными кавычками.

Для двух объектов данных символьного типа с одинаковым параметром вида определена встроенная операция **конкатенации** (//).

Примеры символьных литеральных констант:

```
"DON'T"
'DON''T'
```

обе они имеют значение DON'T, а

```
''
```

имеет значением строку символов нулевой длины. .

Примерами литеральных констант нестандартного символьного типа в процессоре, поддерживающем соответствующие алфавиты, могут быть следующие константы:

CYRILLIC_'БЕЗ НЕЕ НИЧТО НЕ ВОЗМОЖНО'

GREEK_'*αβχδεφγηιφκλμνοπθρστυϖωξψζ*'

где CYRILLIC и GREEK - скалярные целые именованные константы, значения которых являются параметрами вида для Cyrillic (русский и другие славянские языки) и Greek соответственно.

4.3.2.1.1. Упорядочивающая последовательность

Каждая реализация определяет упорядочивающую последовательность для набора символов каждого из видов символьных. **Упорядочивающая последовательность** - это взаимно-однозначное отображение символов на неотрицательные целые, так что каждый символ соответствует одному неотрицательному целому. Встроенные функции CHAR и ICHAR (см. главу 13) осуществляют перевод символов в целые и целых в символы в соответствии с этим отображением. Таким образом,

```
ICHAR ('CHARACTER')
```

выдает целое значение заданного символа в соответствии с упорядочивающей последовательностью процессора.

Для стандартного символьного типа ограничения на упорядочение только такие:

1) ICHAR ('A') < ICHAR ('B') < ... < ICHAR ('Z') для двадцати шести букв;
2) ICHAR ('0') < ICHAR ('1') < ... < ICHAR ('9') для десяти цифр;
3) ICHAR(' ') < ICHAR('0') < ICHAR('9') < ICHAR('A') или
   ICHAR(' ') < ICHAR('A') < ICHAR('Z') < ICHAR('0');
4) ICHAR('a') < ICHAR('b') < ... < ICHAR('z'), если процессор поддерживает строчные буквы;
5) ICHAR(' ') < ICHAR('0') < ICHAR('9') < ICHAR('a') или
   ICHAR(' ') < ICHAR('a') < ICHAR('z') < ICHAR('0'), если процессор поддерживает строчные буквы.

На упорядочение специальных символов и символа подчеркивания нет ограничений (кроме пробела). В стандарте не определено взаимное расположение в упорядочивающей последовательности прописных и строчных букв.

ISO 646:1983 (International Reference Version) устанавливает числовые коды для множества символов, включая буквы, цифры, символ подчеркивания и специальные символы; последовательность таких символов называется **упорядочивающей последовательностью ASCII**. Заметим, что ISO 646:1983 - это международный эквивалент ANSI X3.4-1986, который широко известен как ASCII. Встроенные функции ACHAR и IACHAR обеспечивают преобразование символов в целые в соответствии с упорядочивающей последовательностью ASCII и целых в символы. Встроенные функции LGT, LGE, LLE и LLT обеспечивают сравнение строк, базирующееся на упорядочивающей последовательности ASCII. Если множество символов ограничить буквами, цифрами, символом подчеркивания и специальными символами, международная мобильность гарантируется.

4.3.2.2. Логический тип

Логический тип имеет два значения: "истина" и "ложь".

Процессор должен обеспечивать один или несколько **способов представления**, определяющих множество значений логического типа. Каждый такой способ характеризуется значением параметра, называемого параметром **вида**. Параметр вида для конкретного способа представления выдается с помощью справочной встроенной функции KIND (13.13.51).

Если параметр вида не задан, значение вида по умолчанию есть KIND(.FALSE.), а объект данных относится к **стандартному логическому** типу.

R421 *логическая-литеральная-константа* **есть** .TRUE. [_*параметр-вида*]
или .FALSE. [_*параметр-вида*]

Ограничение: значение *параметра-вида* должно задавать способ представления, существующий в процессоре.

Необязательный параметр вида в конце константы задает параметр вида для логической константы. Если его нет, константа относится к стандартному логическому типу.

Встроенными операциями, определенными для объектов данных логического типа, являются: отрицание (.NOT.), конъюнкция (.AND.), включающая дизъюнкция (.OR.), логическая эквивалентность (.EQV.) и логическая неэквивалентность (.NEQV.). Кроме того, есть набор предопределенных операций отношения, которые производят сравнение значений объектов данных других типов и вырабатывают значения стандартного логического типа. Эти операции описаны в 7.2.3.

Спецификатором типа для логического типа является ключевое слово LOGICAL.

4.4. Производные типы

Из встроенных типов данных можно образовать дополнительные типы данных. Для того, чтобы определить имя типа и имена его компонентов, необходимо описание типа. В конечном счете производный тип распадается на **конечные компоненты**, которые либо относятся к одному из встроенных типов, либо являются указателями.

Производные типы, описанные в разделе спецификаций модуля, доступны по умолчанию (5.1.2.2, 5.2.3) в любой контекстной области, которая имеет доступ к модулю. Это умолчание можно изменить внутри самого модуля ограничением доступа к этим типам. Конкретное описание типа можно объявить как приватное или общедоступное независимо от доступа, объявленного для модуля по умолчанию. Кроме того, тип может быть доступен и когда его компоненты являются приватными.

По умолчанию не предполагается отведение памяти в соответствии с порядком описания компонентов. Однако, если в описании типа содержится ключевое слово SEQUENCE, тип является **последовательным типом** и порядок описания компонентов определяет последовательность памяти для объектов этого типа.

Спецификатором производного типа является ключевое слово TYPE и следующее за ним имя типа в скобках (R502).

4.4.1. Описание производного типа

R422 *описание-производного-типа* **есть** *оператор-заголовка-типа*
[*оператор-PRIVATE-SEQUENCE*]...
оператор-описания-компонента
[*оператор-описания-компонента*]...
оператор-конца-типа

R423 *оператор-PRIVATE-SEQUENCE* **есть** PRIVATE
или SEQUENCE

R424 *оператор-заголовка-типа* **есть** TYPE [,*специф-доступности*] [::] *имя-типа*

Ограничение: один и тот же *оператор-PRIVATE-SEQUENCE* не может появиться более одного раза в одном *описании-производного-типа*.

Ограничение: если есть ключевое слово SEQUENCE, все производные типы, задаваемые в описаниях компонентов, должны быть последовательными типами.

Ограничение: *специф-доступности* (5.1.2.2) или оператор PRIVATE в описании типа разрешены только в том случае, когда описание типа находится в разделе спецификаций модуля.

Ограничение: если компонент производного типа относится к типу, объявленному как приватный, то либо описание типа должно содержать оператор PRIVATE, либо производный тип должен быть приватным.

Ограничение: *имя-типа* не должно совпадать ни с одним из имен встроенных типов, ни с именем другого доступного производного типа.

R425 *оператор-конца-типа* **есть** END TYPE [*имя-типа*]

Ограничение: если за END TYPE следует *имя-типа*, оно должно совпадать с именем в соответствующем *операторе-заголовка-типа*.

R426 *оператор-описания-компонента* **есть** *специф-типа* ♦
♦ [[,*список-специф-атрибутов-компонентов*]::] ♦
♦ *список-объявлений-компонентов*

R427 *специф-атрибутов-компонента* **есть** POINTER
или DIMENSION (*специф-компонента:массива*)

Ограничение: ни один *специф-атрибутов-компонента* не может входить в один и тот же *оператор-описания-компонента* более одного раза.

Ограничение: если для компонента не задан атрибут POINTER, *специф-типа* в *операторе-описания-компонента* должен специфицировать встроенный или предварительно описанный производный тип.

Ограничение: если для компонента задан атрибут POINTER, *специф-типа* в *операторе-описания-компонента* должен специфицировать встроенный тип или любой доступный производный тип, включая тот, который описывается.

R428 *специф-компонента-массива* **есть** *список-специф-явной-конфигурации*
или *список-специф-определяемой-* ♦
♦ *конфигурации*

R429 *объявление-компонента* **есть** *имя-компонента* ♦
♦ [(*специф-компонента-массива*)][* *длина-символьного*]

Ограничение: если атрибут POINTER не специфицирован, каждый *специф-компонента-массива* должен быть *списком-специф-явной-конфигурации*.

Ограничение: если атрибут POINTER специфицирован, каждый *специф-компонента-массива* должен быть *списком-специф-определяемой-конфигурации*.

Ограничение: спецификация " * *длина-символьного* " разрешена, только если специфицируемый тип является символьным.

Ограничение: *длина-символьного* в *описании-компонента* должна быть константным выражением спецификации (7.1.6.2).

Ограничение: каждая граница в *специф-явной-конфигурации* (R428) должна быть константным выражением спецификации (7.1.6.2).

Если есть оператор SEQUENCE, тип является последовательным типом. Если все компоненты в конце концов являются объектами стандартного целого, стандартного вещественного, вещественного двойной точности, стандартного комплексного, стандартного логического типа, но при этом не являются указателями, тип является **числовым последовательным типом**. Если же все компоненты в итоге являются объектами стандартного символьного типа, но не указателями, тип является **символьным последовательным типом**.

Заметим, что разделитель :: обязателен только тогда, когда в *операторе-описания-компонента* заданы либо атрибут POINTER, либо атрибут DIMENSION, либо оба сразу; в остальных случаях он не обязателен.

Компонент является массивом, если его описание содержит *спецификатор-компонента-массива* или его *оператор-описания-компонента* содержит атрибут DIMENSION. Если *описание-компонента* содержит *спецификатор-компонента-массива*, он задает границы массива (5.1.2.4.1); иначе они задаются в *спецификаторе-компонента-массива* атрибута DIMENSION.

Атрибут доступности производного типа может быть специфицирован явно в *спецификаторе-доступности* своего *оператора-заголовка-типа* или в *операторе-доступности* (5.2.3). Если же явно он нигде не специфицирован , принимается его значение по умолчанию. Если тип описан с атрибутом PRIVATE, имя типа, конструктор значения структуры (4.4.4) для типа, все объекты этого типа и все процедуры с формальными аргументами или результатом этого типа доступны только внутри модуля, содержащего данное описание.

Если описание типа содержит оператор PRIVATE, имена компонентов доступны только внутри модуля, содержащего это описание, даже если сам тип является общедоступным (5.1.2.2). Имена компонентов, а, значит и внутренняя структура типа недоступны для любой контекстной области, имеющей доступ к модулю через оператор USE. Аналогично конструктор структуры для этого типа можно употреблять только внутри модуля, содержащего описание.

Пример описания типа:

```
TYPE PERSON
    INTEGER AGE
    CHARACTER (LEN=50) NAME
END TYPE PERSON
```

Пример объявления переменной CHAIRMAN типа PERSON:

```
TYPE (PERSON) :: CHAIRMAN
```

Описание типа может иметь компоненты-массивы, например:

```
TYPE LINE
    REAL,DIMENSION(2,2) :: COORD          !    X1, Y1,
                                          !    X2, Y2
    REAL                :: WIDTH          ! ширина строки в см
    INTEGER             :: PATTERN        ! 1 - для сплошной,
                                          ! 2 - для штриховой,
                                          ! 3 - для пунктирной
END TYPE LINE
```

Пример объявления переменной LINE_SEGMENT производного типа LINE:

```
TYPE (LINE) :: LINE_SEGMENT
```

Компонентом скалярной переменной LINE_SEGMENT является массив, т.е. массив является подобъектом скаляра. Заметим, что сдвоенное двоеточие в описании для COORD обязательно; в описании для WIDTH и PATTERN оно необязательно.

Пример типа с приватными компонентами:

```
MODULE DEFINITIONS
    TYPE POINT
       PRIVATE
       REAL :: X, Y
    END TYPE POINT
END MODULE
```

Такое описание типа доступно в любой контекстной области, имеющей доступ к модулю через оператор USE, однако компоненты X и Y доступны только внутри модуля.

Описание производного типа может иметь компонент тоже производного типа. Например:

```
TYPE TRIANGLE
   TYPE (POINT) :: A, B, C
END TYPE TRIANGLE
```

Пример объявления переменной T типа TRIANGLE:

```
TYPE (TRIANGLE) :: T
```

Пример приватного типа:

```
TYPE, PRIVATE                :: AUXILIARY
    LOGICAL                  :: DIAGNOSTIC
    CHARACTER (LEN=20)       :: MESSAGE
END TYPE AUXILIARY
```

Этот тип будет доступен только в модуле, в котором он описан.

Пример числового последовательного типа:

```
TYPE NUMERIC_SEQ
   SEQUENCE
   INTEGER  :: INT_VAL
   REAL     :: REAL_VAL
   LOGICAL  :: LOG_VAL
END TYPE NUMERIC_SEQ
```

Производный тип может иметь компонент-указатель:

```
TYPE REFERENCE
     INTEGER                                :: VOLUME, YEAR, PAGE
     CHARACTER (LEN=50)                     :: TITLE
     CHARACTER, DIMENSION(:), POINTER :: ABSTRACT
END TYPE REFERENCE
```

Любой объект типа REFERENCE будет иметь четыре компонента фиксированного размера: VOLUME, YEAR, PAGE, TITLE и ABSTRACT - указатель на массив символов. Размер массива-адресата для ABSTRACT будет определяться в процессе выполнения. Память для адресата может быть выделена либо при размещении (6.3.1), либо компонент-массив может быть связан с адресатом при помощи оператора присваивания указателю (7.5.2).

Компонент-указатель производного типа может иметь в качестве адресата объект того типа, компонентом которого он является. Например:

```
TYPE NODE
    INTEGER                 :: VALUE
    TYPE(NODE), POINTER :: NEXT_NODE
END TYPE
```

Такого рода тип можно использовать для построения связанных списков объектов типа NODE.

4.4.2. Определение производных типов

Конкретное имя типа может быть описано в контекстной области не более одного раза. Описания производного типа с одинаковым именем типа могут находиться в разных контекстных областях, в этом случае они независимы и либо описывают разные производные типы, либо могут описывать и один и тот же тип.

Два объекта данных имеют один тип, если они объявлены со ссылкой на одно и то же описание производного типа. Описание может быть доступно либо из модуля, либо из объемлющей контекстной области. Объекты данных из разных контекстных областей также относятся к одному типу, если они описаны со ссылкой на разные описания типа, но имеющие одно имя и характеристику SEQUENCE, а компоненты структуры не имеют

атрибута доступности PRIVATE и согласуются по порядку, имени и атрибутам. Иначе они относятся к разным производным типам. Объект данных, объявленный как имеющий тип с характеристикой SEQUENCE, не совпадает по типу с объектом, тип которого описан как PRIVATE или имеет компонент с атрибутом доступности PRIVATE.

Пример объявления двух объектов, ссылающихся на одно и то же описание производного типа:

```
TYPE POINT
    REAL X, Y
END TYPE POINT

TYPE (POINT) :: X1
CALL SUB (X1)
    ...
CONTAINS
SUBROUTINE SUB (A)
    TYPE (POINT) :: A
    ...
END SUBROUTINE SUB
```

Описание производного типа POINT известно в подпрограмме SUB, т.к. контекстная область SUB входит в контекстную область объемлющей программной единицы (связь по объемлющему контексту). Поскольку объявления X1 и A ссылаются на одно и то же описание производного типа, они оба имеют один и то же тип. X1 и A имели бы один и тот же тип и в том случае, если бы описание производного типа находилось в модуле, к которому имели бы доступ и SUB, и содержащая ее программная единица.

Пример объектов данных из разных контекстных областей, имеющих одинаковый тип:

```
PROGRAM PGM
    TYPE EMPLOYEE
    SEQUENCE
    INTEGER         ID_NUMBER
    CHARACTER (50) NAME
    END TYPE EMPLOYEE
    TYPE (EMPLOYEE) PROGRAMMER
    CALL SUB (PROGRAMMER)
      ...
END PROGRAM PGM

SUBROUTINE SUB (POSITION)
    TYPE EMPLOYEE
        SEQUENCE
        INTEGER         ID_NUMBER
        CHARACTER (50) NAME
    END TYPE
    TYPE (EMPLOYEE) POSITION
    ...
END SUBROUTINE SUB
```

Фактический аргумент PROGRAMMER и формальный аргумент POSITION имеют один тип, поскольку они объявлены со ссылкой на описание производного типа с одним

и тем же именем, с характеристикой SEQUENCE и с компонентами, которые согласованы по порядку, имени, конфигурации, типу и параметрам типа.

Предположим, что имя компонента ID_NUMBER в подпрограмме было ID_NUM. Поскольку не было бы идентичности всех имен компонентов именам компонентов в производном типе EMPLOYEE в главной программной единице, тип у фактического аргумента PROGRAMMER отличался бы от типа формального аргумента POSITION. Таким образом, программа не соответствовала бы стандарту.

4.4.3. Значения производного типа

Множество значений конкретного производного типа состоит из всех возможных последовательностей значений компонентов, приемлемых для данного производного типа.

4.4.4. Конструирование значений производного типа

Описание производного типа неявно описывает и соответствующий **конструктор структуры**, который позволяет образовывать скалярное значение производного типа из последовательности значений, по одному значению на каждый компонент производного типа.

R430 *конструктор-структуры* **есть** *имя-типа (список-выражений)*

Последовательность выражений в конструкторе структуры задает значения компонентов, которые должны согласовываться по количеству и порядку с компонентами производного типа. Если необходимо, каждое значение преобразуется в соответствии с правилами встроенного присваивания (7.5.1.4) к значению, которое по типу и параметрам типа согласуется с соответствующим компонентом производного типа. Для компонентов - не указателей конфигурация выражения должна согласовываться с конфигурацией компонента. Конструктор структуры, все значения в котором являются константными выражениями, представляет собой константное выражение производного типа. Конструктор структуры не должен появляться до описания производного типа, на который он ссылается.

Этот пример иллюстрирует константное выражение, использующее производный тип, описанный в 4.4.1:

```
PERSON (21,'JOHN SMITH')
```

Описание типа может иметь компонентом массив. Кроме того, объект может быть массивом производного типа. Такие массивы можно сконструировать с помощью конструктора массива (4.5).

Когда компонентом производного типа является указатель, результатом вычисления соответствующего выражения конструктора должен быть объект, допустимый как адресат для такого указателя в операторе присваивания указателю (7.5.2). Например, если переменная TEXT объявлена (5.1) следующим образом:

```
CHARACTER, DIMENSION (1:440), TARGET :: TEXT
```

а BIBLIO была объявлена с использованием описания производного типа REFERENCE из 4.4.1:

```
TYPE (REFERENCE) :: BIBLIO
```

то оператор

```
BIBLIO=REFERENCE ( 1, 1987, 1, "This is the title of the    &
                                & referenced paper", TEXT )
```

является правильным, и он отождествляет компонент ABSTRACT объекта BIBLIO с объектом-адресатом TEXT.

Заметим, что нельзя конструировать константные выражения для производного типа, содержащего компонент-указатель, т.к. константное значение не годится в качестве адресата для оператора присваивания указателю.

4.4.5. Операции и присваивание для производного типа

Любая операция или невстроенное присваивание для объектов производного типа должны быть явно определены с помощью функции или подпрограммы и блока процедурного интерфейса (12.3.2.1). Аргументы и значения функций могут быть любого встроенного или производного типа.

4.5. Конструирование значений массива

Конструктор массива определяется как последовательность специфицированных скалярных значений и интерпретируется как одномерный массив, элементами которого являются специфицированные в этой последовательности значения.

R431 *конструктор-массива* **есть** (/*список-значений-констр-массива*/)

R432 *значение-констр-массива* **есть** *выражение*
или *неявный-цикл-констр-массива*

R433 *неявный-цикл-констр-массива* **есть** (*список-значений-констр-массива*,*управление-нцкл-констр-массива*)

R434 *управление-нцкл-констр-массива* **есть** *переменная-нцкл-констр-массива* = *скалярное-целое-выражение*, *скалярное-целое-выражение* [,*скалярное-целое-выражение*]

R435 *переменная-нцкл-констр-массива* **есть** *скалярная-целая-переменная*

Ограничение: *переменная-нцкл-констр-массива* должна быть именованной переменной.

Ограничение: все выражения *значения-констр-массива* в конструкторе массива должны иметь одинаковый тип и одинаковые параметры типа.

Если *значение-констр-массива* - скалярное выражение, его значение задает элемент конструктора массива. Если *значение-констр-массива* - выражение-массив, значения элементов выражения в принятом для массивов порядке (6.2.2.2) задают соответствующую последовательность элементов конструктора массива. Если же это неявный цикл конструктора массива, он разворачивается под управлением *переменной-нцкл-конструктора-массива* в последовательность выражений, как в конструкции DO (8.1.1.4).

Инициализация и исполнение неявного цикла такие же, как в конструкции DO. Для вложенных неявных циклов управляющие переменные не должны совпадать.

Пустая последовательность образует одномерный массив нулевого размера.

Тип и параметры типа у конструктора массива те же, что у *значения-конструктора-массива*.

Если каждое выражение в конструкторе массива - константное, сам конструктор массива является константным выражением. Например:

```
REAL X(3)
X=(/3.2, 4.01, 6.5/)
```

Одномерный массив можно преобразовать в массив любой допустимой конфигурации, используя встроенную функцию RESHAPE (13.13.38). Например,

```
Y = RESHAPE ( SOURCE= (/2.0,(/4.5,4.5/),X/), SHAPE= (/3,2/) )
```

В результате получится Y - массив 3*2 значений:

2.0	3.2
4.5	4.01
4.5	6.5

Примеры конструкторов массива, содержащих неявный цикл:

```
(/ (I, I = 1, 1075) /)
```

и

```
(/ 3.6, (3.6/I, I= 1, N) /)
```

Используя описания типа для PERSON и LINE из 4.4.1, можно сконструировать значения массива производного типа:

```
(/ PERSON (20,'SMITH'), PERSON (20,'JONES') /)
```

Пример конструирования скалярного значения производного типа с компонентом-массивом:

```
LINE (RESHAPE ((/ 2,2/), (/ 0.0, 1.0, 0.0, 2.0 /)), 0.1, 1)
```

В последнем примере используется встроенная функция RESHAPE для конструирования значения, представляющего сплошную линию от (0,0) до (1,2) шириной 0.1 см.

Глава 5. ОБЪЯВЛЕНИЯ И СПЕЦИФИКАЦИИ ОБЪЕКТОВ ДАННЫХ

Каждый объект данных обладает рядом свойств (например, тип, размерность и конфигурация), которые определяют характеристики данного и правила использования объекта. Все вместе эти свойства называются **атрибутами** объекта данных. Именованный объект данных не может быть явно специфицирован в контекстной области более одного раза с конкретным атрибутом. Тип именованного объекта данных либо определяется неявно по первой букве имени (5.3), либо явно специфицируется в **операторе объявления типа**. Дополнительные атрибуты можно специфицировать с помощью отдельных операторов спецификации: в оператор объявления типа можно включить все атрибуты.

Примеры:

```
INTEGER INCOME, EXPENDITURE
```

Объявляются два объекта данных с именами INCOME и EXPENDITURE как объекты целого типа.

```
REAL, DIMENSION (-5:+5) :: X, Y, Z
```

Объявляются три объекта данных с именами X, Y, Z. Все они имеют стандартный вещественный тип и являются массивами явной конфигурации с размерностью, равной 1, нижней границей -5, верхней границей +5 и размером 11.

5.1. Операторы объявления типа

R501 *оператор-объявления-типа* **есть** *специф-типа* ♦
♦ [[,*специф-атрибута*] ... ::] *список-объявлений-объектов*

R502 *специф-типа* **есть** INTEGER [*селектор-вида*]
или REAL [*селектор-вида*]
или DOUBLE PRECISION
или COMPLEX [*селектор-вида*]
или CHARACTER [*селектор-символьного*]
или LOGICAL [*селектор-вида*]
или TYPE (*имя-типа*)

R503 *специф-атрибута* **есть** PARAMETER
или *специф-доступности*
или ALLOCATABLE
или DIMENSION (*специф-массива*)
или EXTERNAL
или INTENT (*специф-назначения*)
или INTRINSIC
или OPTIONAL
или POINTER
или SAVE
или TARGET

R504 *объявление-объекта* **есть** *имя-объекта* [(*специф-массива*)] ♦
♦ [* *длина-символьного*] [= *выражение-инициализации*]
или *имя-функции* [(*специф-массива*)] [* *длина-символьного*]

R505 *селектор-вида* **есть** ([KIND=] ♦
♦ *скалярное-целое-выражение-инициализации*)

Ограничение: ни один *специф-атрибута* не должен входить более одного раза в один и тот же *оператор-объявления-типа*.

Ограничение: *имя-функции* должно быть именем внешней, встроенной, формальной или операторной функции.

Ограничение: конструкция "= *выражение-инициализации*" должна присутствовать, когда оператор содержит атрибут PARAMETER (5.1.2.1).

Ограничение: если присутствует "= *выражение-инициализации*", то перед *списком-объявлений-объектов* должен присутствовать разделитель ::.

Ограничение: если *имя-объекта* - формальный аргумент, результат функции, объект из именованного общего блока (при условии, что объявление типа данных не находится в программной единице-блоке данных), объект из неименованного блока данных, динамически размещаемый массив, указатель, имя внешней или встроенной процедуры, автоматически размещаемый объект, - то конструкция "= *выражение-инициализации*" должна отсутствовать.

Ограничение: спецификация "* *длина-символьного*" разрешена только при объявлении данных символьного типа.

Ограничение: атрибут ALLOCATABLE можно использовать только при условии, что объявляемый массив не является формальным массивом или результатом функции.

Ограничение: для массива, объявленного с атрибутом POINTER или ALLOCATABLE, *специф-массива* должен иметь вид: *список-специф-определяемой-конфигурации* (5.1.2.4.3).

Ограничение: *специф-массива* для *имени-функции*, не имеющего атрибута POINTER, должен иметь вид: *список-специф-явной-конфигурации*.

Ограничение: *специф-массива* для *имени-функции*, имеющего атрибут POINTER, должен иметь вид: *список-специф-определяемой-конфигурации*.

Ограничение: если присутствует атрибут POINTER, нельзя задавать атрибуты TARGET, INTENT, EXTERNAL и INTRINSIC.

Ограничение: если присутствует атрибут TARGET, нельзя задавать атрибуты POINTER, EXTERNAL, INTRINSIC и PARAMETER.

Ограничение: атрибут PARAMETER нельзя задавать для формальных аргументов, указателей, функций, динамически размещаемых массивов и объектов из общего блока.

Ограничение: атрибуты INTENT и OPTIONAL можно задавать только для формальных аргументов.

Ограничение: объект не должен иметь атрибут PUBLIC, если его тип имеет атрибут PRIVATE.

Ограничение: атрибут SAVE нельзя задавать для объекта из общего блока, формального аргумента, процедуры, результата функции и автоматически размещаемого объекта.

Ограничение: объект не должен иметь одновременно и атрибут EXTERNAL, и атрибут INTRINSIC.

Ограничение: объект в *операторе-объявления-типа* не может иметь ни атрибута EXTERNAL, ни атрибута INTRINSIC, если он не является функцией.

Ограничение: массив не должен одновременно иметь и атрибут ALLOCATABLE, и атрибут POINTER.

Ограничение: ни один из атрибутов объекта нельзя объявить более одного раза в одной контекстной области.

Ограничение: значение, указанное в *скалярном-целом-выражении-инициализации*, должно быть неотрицательным и должно специфицировать способ представления, существующий в процессоре.

Заметим, что разделитель :: обязателен только в случае, если в операторе есть *специф-атрибута* или "= *выражение-инициализации*", в остальных случаях он не обязателен.

Имя, идентифицирующее в контекстной области специфическую встроенную функцию, имеет тип, приведенный в 13.12. Явный оператор объявления типа для этих функций не обязателен, но разрешен. Спецификации типа для обобщенного имени встроенной функции в операторе объявления типа еще недостаточно для утраты этой функцией ее родовых свойств.

Выражение-спецификации (7.1.6.2) в *значении-параметра-типа* (5.1.1.5) или в *специф-массива* (5.1.2.4) может быть неконстантным выражением при условии, что это выражение располагается в теле интерфейса (12.3.2.1) или в разделе спецификаций процедуры. Если объект данных объявлен с такими неконстантными выражениями, не являясь при этом формальным аргументом, такой объект называется **автоматически**

размещаемым объектом данных. Автоматически размещаемый объект не может входить в операторы SAVE и DATA, не может иметь атрибута SAVE и не может быть начально определен посредством конструкции "= *выражение-инициализации*".

Если *селектор-длины* является неконстантным выражением, длина объявляется при входе в процедуру и не зависит от определения или переопределения переменных (входящих в выражение спецификации) во время выполнения процедуры.

Если в *объявлении-объекта* есть "= *выражение-инициализации*" и объект не имеет атрибута PARAMETER, то *имя-объекта* - это переменная, значение которой начально определено. *Имя-объекта* становится определенным и получает значение из *выражения-инициализации* в соответствии с правилами встроенного присваивания (7.5.1.4). Нельзя более одного раза инициализировать переменную или ее часть.

Присутствие конструкции "= *выражение-инициализации*" подразумевает наличие атрибута SAVE для всех объектов, кроме *имен-объектов* из именованного общего блока. Неявный атрибут SAVE может быть подтвержден явным использованием атрибута SAVE в операторе объявления типа или включением имени объекта в оператор SAVE (5.2.4).

Примеры операторов объявления типа:

```
REAL A(10)
LOGICAL, DIMENSION (5,5)  ::  MASK1, MASK2
COMPLEX :: CUBE_ROOT = (-0.5, 0.866)
INTEGER, PARAMETER :: SHORT = SELECTED_INT_KIND (4)
REAL (KIND (0.0D0)) A
REAL (KIND = 2) B
COMPLEX (KIND = KIND(0.0D0)) :: C
INTEGER (SHORT) K                          ! Диапазон от - 9999 до 9999
TYPE (PERSON) :: CHAIRMAN
```

5.1.1. Спецификаторы типа

Спецификатор типа специфицирует тип всех объектов, перечисленных в списке объявлений объектов. Он может отменять или подтверждать тип объекта, определяемый по первой букве его имени в соответствии с действующими правилами неявной типизации (5.3).

5.1.1.1. INTEGER

Спецификатор типа - INTEGER означает, что все объекты, имена которых объявлены в этом операторе, относятся к встроенному типу - целый (4.3.1.1). Если есть селектор вида, он задает способ представления целых. Если его нет, значением параметра вида является KIND(0) и объявленные объекты имеют стандартный целый тип. Объект, объявленный со спецификатором типа INTEGER (KIND(0)), относится к тому же типу, что и объект, объявленный со спецификатором типа INTEGER.

5.1.1.2. REAL

Спецификатор типа REAL означает, что все объекты, имена которых объявлены в этом операторе, относятся к встроенному типу - вещественный (4.3.1.2). Если есть селектор вида, он задает способ аппроксимации вещественных. Если его нет, значением параметра вида является KIND(0.0) и объявляемые объекты имеют тип стандартный вещественный. Объект, объявленный со спецификатором типа REAL (KIND(0.0)), относится к тому же типу, что и объект, объявленный со спецификатором типа REAL.

5.1.1.3. DOUBLE PRECISION

Спецификатор типа - DOUBLE PRECISION означает, что все объекты, имена которых объявлены в этом операторе, относятся к встроенному типу - вещественный двойной

точности (4.3.1.3). Значением параметра вида является KIND(0.0D0). Объект, объявленный как REAL (KIND (0.0D0)), относится к тому же типу, что объект, объявленный как DOUBLE PRECISION.

5.1.1.4. COMPLEX

Спецификатор типа COMPLEX означает, что все объекты, имена которых объявлены в этом операторе, относятся к встроенному типу - комплексный (4.3.1.1). Если есть селектор вида, он задает способ аппроксимации вещественных для двух вещественных значений, образующих действительную и мнимую части комплексного значения. Если его нет, значением параметра вида является KIND(0.0) и объявленные объекты имеют стандартный комплексный тип. Объект, объявленный со спецификатором типа - COMPLEX (KIND(0.0)), относится к тому же типу, что и объект, объявленный со спецификатором типа - COMPLEX.

5.1.1.5. CHARACTER

Спецификатор типа - CHARACTER означает, что все объекты, имена которых объявлены в этом операторе, относятся к встроенному типу - символьный (4.3.2.1).

R506 *селектор-символьного* **есть** *селектор-длины*
или (LEN = *значение-параметра-типа*, ♦
♦ KIND = *скалярное-целое-выражение-инициализации*)
или (*значение-параметра-типа*, ♦
♦ KIND = *скалярное-целое-выражение-инициализации*)
или (KIND = *скалярное-целое-выражение-инициализации*) ♦
♦ [,LEN = *значение-параметра-типа*]

R507 *селектор-длины* **есть** ([LEN=] *значение-параметра-типа*)
или * *длина-символьного* [,]

R508 *длина-символьного* **есть** (*значение-параметра-типа*)
или *скалярная-целая-литеральная-константа*

Ограничение: необязательная запятая в ***селекторе-длины*** разрешена только в *специф-типа* в *операторе-объявления-типа*.

Ограничение: необязательная запятая в *селекторе-длины* может появиться только тогда, когда в *операторе-объявления-типа* отсутствует разделитель :: .

Ограничение: значение *скалярного-целого-выражения-инициализации* должно быть неотрицательным и должно специфицировать способ представления, существующий в процессоре.

Ограничение: *скалярная-целая-литеральная-константа* не должна включать *параметр-вида*.

Селектор-символьного в *спецификаторе-типа* CHARACTER и спецификация "* *длина-символьного*" в объявлении-объекта определяют длину символьных объектов. Спецификация "* *длина-символьного*" в *объявлении-объекта* специфицирует длину для отдельного объекта и отменяет для него длину, указанную в *селекторе-символьного*. Если в *объявлении-объекта* не указана спецификация "* *длина-символьного*", длина объекта определяется *селектором-длины* или *значением-параметра-типа* в *селекторе-символьного*. Если же длина объекта не указана ни в *селекторе-символьного*, ни при помощи спецификации "* *длина-символьного*", длина объекта считается равной 1.

R509 *значение-параметра-типа* **есть** *выражение-спецификации*
или *

Ограничение: имя функции нельзя объявлять со *значением-параметра-типа* в виде звездочки, если функция является внутренней, модульной, рекурсивной или ее значением является массив или указатель.

Неопределенный-параметр-типа - это параметр типа для формального аргумента, для которого в качестве *значения-параметра-типа* используется символ "звездочка".

Если параметр типа длина получает в результате вычисления отрицательное значение, длина символьного объекта объявляется равной нулю. Звездочку в качестве значения параметра типа можно использовать только в следующих случаях.

1. Она может быть использована при объявлении формального аргумента процедуры - тогда длиной формального аргумента предполагается длина фактического аргумента, связанного с ним при вызове процедуры.
2. Она может быть использована при объявлении именованной константы - тогда длина определяется по значению константы.
3. Во внешней функции имя результата функции может быть задано с параметром типа - длиной в виде звездочки; тогда каждая контекстная область, вызывающая функцию, должна либо объявить имя этой функции со значением параметра типа, отличным от звездочки, либо иметь доступ к такому определению через связь по использованию или по объемлющему контексту. Когда функция вызвана, в качестве длины результирующей переменной этой функции берется значение этого параметра типа.

Длина, заданная для символьной операторной функции или для символьного аргумента операторной функции, должна быть целым константным выражением.

Селектор вида, если он есть, задает способ представления символьных. Если он отсутствует, параметром вида является KIND ('A'), а объект имеет тип - стандартный символьный.

Примеры операторов объявления символьного типа:

CHARACTER (LEN=10, KIND=2) A

CHARACTER *10 B, C*20

5.1.1.6. LOGICAL

Спецификатор типа LOGICAL означает, что все объекты, имена которых объявлены в этом операторе, относятся к встроенному типу - логический (4.3.2.2).

Если есть селектор-вида, он задает способ представления. Если его нет, значением параметра вида является KIND (.FALSE.) и объявляемые объекты имеют тип - стандартный логический.

5.1.1.7. Производный тип

Спецификатор типа - TYPE означает, что все объекты, имена которых объявлены в этом операторе, относятся к одному производному типу, заданному *именем-типа*. Типы компонентов каждого такого объекта тоже объявляются в соответствующем операторе *объявления-компонента* в *описании-производного-типа* (4.4.1). Производный тип, специфицированный для объектов данных, должен быть предварительно описан в контекстной области или должен быть доступен посредством связи по объемлющему контексту или по использованию.

Скалярным объектом производного типа является **структура**.

Если производный тип имеет характеристику SEQUENCE, скалярный объект такого типа является **последовательной структурой**. Скалярный объект числового последовательного типа (4.4.1) является **числовой последовательной структурой**. Скалярный объект символьного последовательного типа (4.4.1) является **символьной последовательной структурой**.

Описание для формального аргумента непоследовательного производного типа должно задавать производный тип, доступный посредством связи по объемлющему контексту или по использованию, поскольку для того, чтобы и формальный,и фактический аргументы относились к одному производному типу, надо, чтобы использовалось одно и то же объявление типа. Это ограничение не относится к аргументам последовательного типа (4.4.2).

5.1.2. Атрибуты

Дополнительные атрибуты, которые могут появиться в операторе объявления типа, специфицируют либо природу объявляемых объектов, либо ограничение на их использование в программе.

5.1.2.1. Атрибут PARAMETER

Атрибут PARAMETER означает, что объекты, имена которых объявлены в операторе, являются именованными константами. *Имя-объекта* становится определенным и получает значение *выражения-инициализации*, находящегося справа от знака равенства, в соответствии с правилами встроенного присваивания (7.5.1.4). Появление в спецификации атрибута PARAMETER требует, чтобы для всех объектов, входящих в *список-объявлений-объектов*, было объявлено "=*выражение-инициализации*".

Любая именованная константа, входящая в выражение инициализации, должна быть предварительно определена в том же операторе объявления типа или в другом операторе объявления типа с атрибутом PARAMETER, или в операторе PARAMETER, либо должна быть доступна посредством связи по объемлющему контексту или по использованию. На именованную константу нельзя ссылаться в каком-либо другом контексте, если она не была определена в предшествующем операторе PARAMETER или в операторе объявления типа, использующем атрибут PARAMETER, либо не была сделана доступной посредством связи по объемлющему контексту или по использованию.

Именованная константа не должна появляться в спецификации формата (10.1.1).

Примеры объявлений с атрибутом PARAMETER:

```
REAL, PARAMETER :: ONE = 1.0, Y = 4.1 / 3.0
INTEGER, DIMENSION(3), PARAMETER :: ORDER = (/ 1, 2, 3 /)
```

5.1.2.2. Атрибут доступности

Атрибут доступности специфицирует доступность объектов и описаний производных типов.

R510 *специф-доступности* **есть** PUBLIC
или PRIVATE

Ограничение: атрибут *специф-доступности* может находиться только в контекстной области модуля.

Объекты, объявленные с атрибутом PRIVATE, недоступны вне модуля. Объекты, объявленные с атрибутом PUBLIC, могут быть доступны для других программных единиц с помощью оператора USE. Для объектов без явно заданного *спецификатора-доступности* по умолчанию значение атрибута доступности - PUBLIC, если оно не изменено с помощью оператора PRIVATE (5.2.3).

Пример спецификации атрибута доступности:

```
REAL, PRIVATE :: X, Y, Z
```

5.1.2.3. Атрибут INTENT

Атрибут INTENT специфицирует допустимое использование формального аргумента.

R511 *специф-назначения* **есть** IN
или OUT
или INOUT

Ограничение: нельзя специфицировать атрибут INTENT для формального аргумента, который является формальной процедурой или формальным указателем.

Атрибут INTENT(IN) означает, что формальный аргумент не может быть переопределен или лишен определенности во время выполнения процедуры.

Атрибут INTENT(OUT) означает, что формальный аргумент должен быть определен до выполнения любой ссылки на него внутри процедуры, а любой связанный с ним фактический аргумент должен быть определимым. При вызове процедуры такой формальный аргумент становится неопределенным.

Атрибут INTENT(INOUT) означает, что формальный аргумент предназначен как для получения данных из вызывающей контекстной области, так и для возврата данных в нее. Любой связанный с ним фактический аргумент должен быть определимым.

Если для формального аргумента не специфицирован атрибут INTENT, это значит, что он подчиняется всем ограничениям, налагаемым на связанный с ним фактический аргумент (12.5.2.1, 12.5.2.2, 12.5.2.3).

Пример задания спецификации INTENT:

```
SUBROUTINE MOVE (FROM, TO)
    USE PERSON_MODULE
    TYPE(PERSON), INTENT(IN) :: FROM
    TYPE(PERSON), INTENT(OUT):: TO
```

5.1.2.4. Атрибут DIMENSION

Атрибут DIMENSION означает, что все объекты, имена которых объявлены в этом операторе, являются массивами. Размерность или размерность и конфигурация специфицируются с помощью *спецификатора-массива* (если он есть) в *объявлении-объекта*, либо с помощью *спецификатора-массива* в атрибуте DIMENSION в противном случае. *Спецификатор-массива* в *объявлении-объекта* специфицирует либо размерность, либо размерность и конфигурацию для отдельного массива и отменяет для данного массива действие *спецификатора-массива* в атрибуте DIMENSION. Если атрибут DIMENSION опущен, *спецификатор-массива* должен быть задан в *объявлении-объекта* для того, чтобы объявить массив в этом операторе.

R512 *специф-массива* **есть** *список-специф-явной-конфигурации*
или *список-специф-неопределенной-конфигурации*
или *список-специф-определяемой-конфигурации*
или *специф-неопределенного-размера*

Ограничение: максимальная размерность равна семи.

Примеры спецификации атрибута DIMENSION:

```
SUBROUTINE EX (N, A, B, S)
   REAL, DIMENSION(N,10) :: W              ! Автоматически размещаемый
                                           ! массив явной конфигурации
   REAL A(:), B(0:)                        ! Массив неопределенной
                                           ! конфигурации
   REAL, POINTER :: D(:, :)                ! Указатель-на-массив
   REAL, DIMENSION(:), POINTER :: P        ! Указатель-на-массив
   REAL, ALLOCATABLE, DIMENSION(:)::E      ! Динамически размещаемый
                                           ! массив
   REAL :: S (N,*)                         ! Массив неопределенного
                                           ! размера
```

5.1.2.4.1. Массив явной конфигурации

Массив явной конфигурации - это именованный массив, объявленный со *списком-спецификаторов-явной-конфигурации*, специфицирующим явные значения для границ измерений массива.

R513 *специф-явной-конфигурации* **есть** [*нижняя-граница* :] *верхняя-граница*

R514 *нижняя-граница* **есть** *выражение-спецификации*

R515 *верхняя-граница* **есть** *выражение-спецификации*

Ограничение: массив явной конфигурации, границы которого зависят от значений неконстантных выражений, должен быть либо формальным аргументом, либо результатом функции, либо массивом, автоматически размещаемым в процедуре.

Автоматически размещаемый массив - это объявленный в процедуре массив явной конфигурации, который не является формальным аргументом, но имеет в качестве границ неконстантные выражения спецификации.

Если у массива явной конфигурации границами являются неконстантные выражения спецификации, границы, а значит и конфигурация определяются при входе в процедуру путем вычисления значений выражений границ. Границы такого массива не зависят от переопределения или разопределения (во время исполнения процедуры) переменных из выражения спецификации.

Значения каждой *верхней-границы* и *нижней-границы* определяют границы массива по конкретному измерению, а значит и протяженность по этому измерению. Значение нижней или верхней границы может быть положительным, отрицательным или нулевым. Диапазон индексов массива в этом измерении - это множество целых значений, заключенных между нижней и верхней границами (включая их) при условии, что верхняя граница не меньше нижней. Если верхняя граница меньше нижней, то диапазон пуст, протяженность этого измерения равна нулю, а массив имеет нулевой размер. Если *нижняя-граница* опущена, подразумевается по умолчанию ее значение, равное 1. Количество наборов границ является размерностью.

5.1.2.4.2. Массив неопределенной конфигурации

Массив неопределенной конфигурации - это формальный массив, который не является указателем и наследует свою конфигурацию от связанного с ним фактического аргумента-массива.

R516 *специф-неопределенной-конфигурации* **есть** [*нижняя-граница*] :

Размерность равна числу двоеточий в *списке-спецификаторов-неопределенной-конфигурации*.

Протяженность измерения массива неопределенной конфигурации - это протяженность соответствующего измерения связанного с ним фактического аргумента-массива. Если значение нижней границы d, а протяженность соответствующего измерения связанного фактического аргумента-массива s, тогда значение верхней границы равно $s+d-1$. Нижняя граница равна *нижней-границе*, если она задана, иначе она равна 1.

5.1.2.4.3. Массив определяемой конфигурации

Массив определяемой конфигурации - это либо указатель-на-массив, либо динамически размещаемый массив.

Динамически размещаемый массив - это именованный массив, имеющий атрибут ALLOCATABLE и заданную размерность, но его границы, а следовательно, и конфигурация определяются при размещении массива в памяти при выполнении оператора ALLOCATE (6.3.1).

Атрибут ALLOCATABLE может быть задан для массива в операторе объявления типа или в операторе ALLOCATABLE (5.2.6). Массив с атрибутом ALLOCATABLE должен быть объявлен со *списком-спецификаторов-определяемой-конфигурации* в операторе объявления типа, в операторе ALLOCATABLE, в операторе DIMENSION (5.2.5) или в операторе TARGET (5.2.8). Тип и параметры типа можно специфицировать в операторе объявления типа.

Указатель-на-массив - это именованный массив с атрибутом POINTER и заданной размерностью. Его тип, параметры типа и размерность специфицируются в операторе объявления типа или в операторе описания компонента, а границы и,

следовательно, конфигурация определяются при установлении его связи с адресатом при помощи оператора присваивания указателю (7.5.2) или при выполнении оператора ALLOCATE (6.3.1). Атрибут POINTER может быть задан для массива либо в операторе объявления типа, либо в операторе POINTER (5.2.7). Массив с атрибутом POINTER должен быть объявлен со *списком-спецификаторов-определяемой-конфигурации* в операторе объявления типа, в операторе POINTER или в операторе DIMENSION (5.2.5).

R517 *специф-определяемой-конфигурации* **есть** :

Размерность равна числу двоеточий в *списке-спецификаторов-определяемой-конфигурации*.

Размер, границы и конфигурация динамически размещаемого, но еще не размещенного, массива не определены. Нельзя ни определять какую-либо его часть, ни ссылаться на нее, кроме как в качестве аргумента справочных встроенных функций для запроса о статусе размещенности аргумента, о его типе или параметрах типа. Нижняя и верхняя границы каждого измерения - такие, как указано в операторе ALLOCATE при размещении массива.

Размер, границы и конфигурация адресата для несвязанного указателя-на-массив не определены. Нельзя ни определять какую-либо часть такого массива, ни ссылаться на нее, кроме как в качестве аргумента справочных встроенных функций для запроса о присутствии аргумента, о его типе и параметрах типа, о статусе связанности. Границы каждого измерения указателя-на-массив можно определить двумя способами.

1. Они задаются в операторе ALLOCATE (6.3.1), когда размещается адресат.
2. Они задаются в операторе присваивания указателю. Нижняя граница есть результат функции LBOUND (13.13.52) применительно к соответствующему измерению адресата. Верхняя граница каждого измерения есть результат функции UBOUND (13.13.111) применительно к соответствующему измерению адресата.

На границы массива-адресата или динамически размещаемого массива не влияет последующее переопределение или разопределение переменных, входящих в границы.

Формальный аргумент-указатель может быть связан только с фактическим аргументом-указателем, тогда как указатель - фактический аргумент может быть связан и не с указателем в качестве формального аргумента.

Результат функции может быть объявлен с атрибутом POINTER.

5.1.2.4.4. Массив неопределенного размера

Массив неопределенного размера - это формальный массив, размер которого наследуется им от связанного с ним фактического массива. Размерность и протяженности измерений у формального и фактического массивов могут быть разными; формальному массиву передается только размер фактического массива.

R518 *специф-неопределенного-размера* **есть** [*список-специф-явной-* ♦
♦ *конфигурации*,] [*нижняя-граница* :] *

Ограничение: имя функции, значением которой является массив, не должно быть объявлено как массив неопределенного размера.

Размер массива неопределенного размера определяется следующим образом:

1) если фактическим аргументом, связанным с формальным массивом неопределенного размера, является имя массива любого типа, кроме символьного, размером формального массива становится размер фактического массива;
2) если фактическим аргументом, связанным с формальным массивом неопределенного размера, является элемент массива любого типа, кроме символьного, с порядковым значением индекса r (6.2.2.2) в массиве размера x, размер формального массива равен $x-r+1$;
3) если фактический аргумент - массив стандартного символьного типа, элемент массива стандартного символьного типа или подстрока элемента

массива стандартного символьного типа (6.1.1), которые начинаются с символьной единицы памяти *r* в массиве из *c* символьных единиц памяти, то размер формального массива равен MAX(INT((*c*-*r*+1)/*e*, 0), где *e* - длина элемента формального символьного массива.

Размерность массива равна единице плюс число *спецификаторов-явной-конфигурации*.

Массив неопределенного размера не имеет верхней границы в последнем измерении, и, следовательно, у него нет протяженности последнего измерения и нет конфигурации. Имя такого массива нельзя записать в ссылке на полный массив, кроме случая использования его в качестве фактического аргумента в ссылке на процедуру, когда конфигурация не требуется, или в ссылке на встроенную функцию LBOUND.

Границы первых *n*-1 измерений задаются *списком-спецификаторов-явной-конфигурации* (если он есть) в *спецификаторе-неопределенного-размера*. Нижней границей последнего измерения является либо *нижняя-граница*, если она задана, либо 1. Можно задавать элементы и секции (6.2.2.3) массива неопределенного размера. Нельзя опускать верхнюю границу в индексном триплете последнего измерения.

Если массив неопределенного размера имеет границы с неконстантными выражениями спецификации, границы объявляются при входе в процедуру. На границы такого массива не влияет любое переопределение или разопределение (во время исполнения процедуры) переменных из выражений спецификации.

5.1.2.5. Атрибут SAVE

Атрибут SAVE означает, что все объекты, объявление которых содержит этот атрибут, сохраняют свой статус связанности, статус размещенности, статус определенности и значение после выполнения оператора RETURN или END в контекстной области, содержащей это объявление. Такой объект называется **сохраняемым объектом**.

Объекты в контекстной области модуля могут быть объявлены с атрибутом SAVE. Такие объекты сохраняют свой статус определенности, статус связанности, статус размещенности и значение при выполнении операторов RETURN или END в любой процедуре, имеющей доступ к модулю через оператор USE.

Объекты, объявленные с атрибутом SAVE в контекстной области процедуры, используются совместно всеми экземплярами (12.5.2.4) процедуры.

Атрибут SAVE нельзя специфицировать для объектов из общих блоков, для формальных аргументов, процедур, результата функции или автоматически размещаемого объекта.

Атрибут SAVE может находиться и в объявлениях главной программной единицы, это ни на что не влияет.

5.1.2.6. Атрибут OPTIONAL

Атрибут OPTIONAL может быть специфицирован только в контекстной области процедуры или блока интерфейса и только для формальных аргументов. Он означает, что формальный аргумент не обязательно должен быть связан с фактическим при ссылке на процедуру (12.5.2.8). Для того, чтобы определить, связан ли формальный аргумент, имеющий атрибут OPTIONAL, с фактическим аргументом, можно использовать встроенную функцию PRESENT (13.13.80).

5.1.2.7. Атрибут POINTER

Атрибут POINTER означает, что на объект нельзя ссылаться и нельзя его определять до тех пор, пока он не будет посредством оператора присваивания указателю (7.5.2) или с помощью оператора ALLOCATE (6.3.1) связан с адресатом, на который можно ссылаться и который можно определять. Если указатель объявлен как

массив, он должен быть объявлен со *списком-спецификаторов-определяемой-конфигурации*. Объект с атрибутом POINTER занимает неспецифицированную единицу памяти (14.6.3.1).

Примеры спецификации атрибута POINTER:

```
TYPE (NODE), POINTER :: CURRENT, TAIL
REAL, DIMENSION (:,:), POINTER :: IN, OUT, SWAP
```

5.1.2.8. Атрибут TARGET

Атрибут TARGET означает, что объект может быть связан с указателем (7.5.2). Объект без атрибутов POINTER или TARGET не может быть связан с указателем.

Примеры спецификации атрибута TARGET:

```
TYPE (NODE), TARGET :: HEAD
REAL, DIMENSION (1000, 1000), TARGET :: A, B
```

5.1.2.9. Атрибут ALLOCATABLE

Атрибут ALLOCATABLE означает, что все объекты, объявленные в этом операторе, являются динамически размещаемыми массивами. Это должны быть массивы определяемой конфигурации, которая определяется при размещении каждого из массивов в памяти во время выполнения оператора ALLOCATE (6.3.1).

5.1.2.10. Атрибут EXTERNAL

Атрибут EXTERNAL означает, что имя объекта в объявлении, содержащем этот атрибут, является внешней или формальной функцией и позволяет использовать это имя в качестве фактического аргумента. Этот атрибут может быть объявлен и в операторе EXTERNAL (12.3.2.2).

5.1.2.11. Атрибут INTRINSIC

Атрибут INTRINSIC означает, что имя объекта в объявлении, содержащем этот атрибут, должно быть специфическим или обобщенным именем встроенной функции, и позволяет использовать это имя в качестве фактического аргумента, если это специфическое имя встроенной функции (13.12). Этот атрибут может быть объявлен и в операторе INTRINSIC (12.3.2.3).

5.2. Операторы спецификации атрибутов

Все атрибуты объекта (кроме типа) могут быть специфицированы отдельно от объявления типа с помощью операторов спецификации одного атрибута. На комбинацию атрибутов, которые можно задать для конкретного объекта, накладываются те же ограничения, что для операторов объявления типа, независимо от способа задания. Это также применимо и к операторам EXTERNAL и INTRINSIC.

5.2.1. Оператор INTENT

R519 *оператор-INTENT* **есть** INTENT(*специф-назначения*) [::] ♦
♦ *список-имен-формальных-аргументов*

Ограничение: *оператор-INTENT* может находиться только в *разделе-спецификаций* процедуры или тела интерфейса (12.3.2.1).

Ограничение: *имя-формального-аргумента* не должно быть именем формальной процедуры или формального аргумента-указателя.

Этот оператор специфицирует предполагаемое использование указанных в нем формальных аргументов. Каждый из перечисленных в нем формальных аргументов имеет атрибут INTENT.

Пример оператора INTENT:

```
SUBROUTINE EX (A,B)
    INTENT (INOUT) :: A, B
```

5.2.2. Оператор OPTIONAL

R520 *оператор-OPTIONAL* **есть** OPTIONAL [::] *список-имен-формальных-аргументов*

Ограничение: *оператор-OPTIONAL* может находиться только в контекстной области процедуры или тела интерфейса.

Этот оператор включает формальные аргументы, которые не обязательно должны быть связаны с фактическими при вызове процедуры (12.5.2.8). Каждый из перечисленных в нем формальных аргументов имеет атрибут OPTIONAL.

Пример оператора OPTIONAL:

```
SUBROUTINE EX (A,B)
    OPTIONAL :: A
```

5.2.3. Операторы PRIVATE и PUBLIC

R521 *оператор-доступности* **есть** *специф-доступности* [[::] ♦
♦ *список-идентификаторов-доступности*]

R522 *идентификатор-доступности* **есть** *имя-для-USE*
или *обобщенный-специф*

Ограничение: *оператор-доступности* может находиться только в контекстной области модуля. Только один оператор доступности без *списка-идентификаторов-доступности* разрешен в контекстной области модуля.

Ограничение: каждое *имя-для-USE* должно быть именем переменной, процедуры, производного типа, группы NAMELIST или именованной константы.

Ограничение: модульная процедура, формальный аргумент или результат которой принадлежит к типу с атрибутом доступности PRIVATE, должна иметь атрибут доступности PRIVATE и не должна иметь обобщенный идентификатор с атрибутом доступности PUBLIC.

Этот оператор объявляет атрибут доступности объектов (5.1.2.2) - **PUBLIC** или **PRIVATE**. Процедура, имеющая общедоступный обобщенный идентификатор (12.3.2.1), доступна через этот идентификатор, даже если ее специфическое имя является приватным.

Если *оператор-доступности* без *списка-идентификаторов-доступности* находится в контекстной области модуля, оператор устанавливает атрибут доступности по умолчанию для всех потенциально доступных объектов в контекстной области модуля. Например, оператор

PUBLIC

устанавливает атрибут доступности по умолчанию - PUBLIC. Оператор

PRIVATE

устанавливает атрибут доступности по умолчанию - PRIVATE. Если же в модуле нет этих операторов, атрибут доступности по умолчанию - PUBLIC.

Примеры операторов спецификации атрибута доступности:

```
MODULE EX
    PRIVATE
    PUBLIC :: A,B,C, ASSIGNMENT (=), OPERATOR (+)
```

5.2.4. Оператор SAVE

R523 *оператор-SAVE* **есть** SAVE [[::] список-сохраняемых-объектов]

R524 *сохраняемый-объект* **есть** *имя-объекта*
или /*имя-общего-блока*/

Ограничение: *имя-объекта* не должно быть именем формального аргумента, процедуры, результата функции, автоматически размещаемого объекта или именем объекта из общего блока.

Ограничение: если в контекстной области есть оператор SAVE без списка сохраняемых объектов, все другие вхождения оператора или атрибута SAVE в этой контекстной области запрещены.

Все объекты, указанные явно или включенные в явно указанный общий блок, имеют атрибут SAVE (5.1.2.5). Если конкретное имя общего блока присутствует в операторе SAVE какой-либо контекстной области выполняемой программы (кроме главной программной единицы), оно должно присутствовать в операторе SAVE каждой контекстной области (кроме главной программной единицы), в которой есть этот общий блок. Если в операторе SAVE объявлено имя общего блока, текущие значения объектов в последовательности памяти общего блока (5.5.2.1) во время выполнения операторов END или RETURN становятся доступными для следующей контекстной области в последовательности исполнения выполняемой программы, которая специфицирует имя этого общего блока или доступ к нему. Если именованный общий блок специфицирован в контекстной области главной программной единицы, текущие значения из последовательности памяти общего блока становятся доступными для каждой контекстной области, в которой специфицировано это имя общего блока. Статус определенности каждого объекта в последовательности памяти именованного общего блока зависит от связи, установленной для последовательности памяти общего блока.

Оператор SAVE с пустым списком сохраняемых объектов трактуется так, как если бы он содержал имена всех допустимых объектов в этой контекстной области.

Оператор SAVE может появиться в разделе спецификаций главной программной единицы, но не оказывает при этом никакого влияния.

Пример оператора SAVE:

```
SAVE A, B, C, /BLOCKA/, D
```

5.2.5. Оператор DIMENSION

R525 *оператор-DIMENSION* **есть** DIMENSION [::] *имя-массива* ♦
♦ (*специф-массива*) [,*имя-массива* (*специф-массива*)]...

Оператор специфицирует список имен объектов, которым приписывается атрибут DIMENSION (5.1.2.4), и характеристики для каждого из перечисленных массивов.

Пример оператора DIMENSION:

```
DIMENSION A(10), B(10, 70), C(-3:12, *)
```

5.2.6. Оператор ALLOCATABLE

R526 *оператор-ALLOCATABLE* **есть** ALLOCATABLE [::] ♦
♦ *имя-массива* [(*список-специф-определяемой-конфигурации*)] ♦
♦ [,*имя-массива*[(*список-специф-определяемой-конфигурации*)]]...

Ограничение: *имя-массива* не должно быть формальным аргументом или результатом функции.

Ограничение: если где-нибудь в контекстной области специфицируется атрибут DIMENSION для *имени-массива*, *спецификатор-массива* должен быть *списком-спецификаторов-определяемой-конфигурации*.

Этот оператор описывает список имен массивов с атрибутом ALLOCATABLE (5.1.2.9). Конфигурация для динамически размещаемого массива определяется во время размещения массива в памяти с помощью оператора ALLOCATE (6.3.1).

Пример оператора ALLOCATABLE:

```
REAL A, B(:)
ALLOCATABLE :: A (:, :), B
```

5.2.7. Оператор POINTER

R527 *оператор-POINTER* **есть** POINTER [::] ♦
♦ *имя-объекта* [(*список-специф-определяемой-конфигурации*)] ♦
♦ [,*имя-объекта*[(*список-специф-определяемой-конфигурации*)]]...

Ограничение: для *имени-объекта* не должен быть специфицирован атрибут INTENT.

Ограничение: если для *имени-объекта* где-нибудь в контекстной области специфицируется атрибут DIMENSION, *спецификатор-массива* должен быть *списком-спецификаторов-определяемой-конфигурации*.

Ограничение: для *имени-объекта* не должен быть специфицирован атрибут PARAMETER.

Этот оператор специфицирует список имен объектов, которые имеют атрибут POINTER (5.1.2.7). На такой объект нельзя ссылаться и нельзя его определять до тех пор, пока в результате выполнения оператора присваивания указателю (7.5.2) или оператора ALLOCATE (6.3.1) он не будет связан с адресатом, на который можно ссылаться и который можно определять.

Пример оператора POINTER:

```
TYPE (NODE)     :: CURRENT
POINTER         :: CURRENT, A(:,:)
```

5.2.8. Оператор TARGET

R528 *оператор-TARGET* **есть** TARGET [::] ♦
♦ *имя-объекта* [(*специф-массива*)] ♦
♦ [,*имя-объекта* [(*специф-массива*)]]...

Ограничение: для *имени-объекта* не должен быть специфицирован атрибут PARAMETER.

Этот оператор специфицирует список объектов, которые имеют атрибут TARGET и поэтому могут быть связаны с указателями.

Пример оператора TARGET:

```
TARGET :: A(1000,1000), B
```

5.2.9. Оператор DATA

Оператор DATA используется для задания начальных значений переменных.

R529 *оператор-DATA* **есть** DATA *множество-оператора-DATA* ♦
♦ [[,] *множество-оператора-DATA*]...

Ни переменная, ни какая-либо ее часть не могут быть начально определены в выполняемой программе более чем один раз.

Если тип переменной, входящей в оператор DATA, не был предварительно объявлен, эта переменная может появиться в последующих объявлениях типа только при условии, что эти объявления согласуются с неявным объявлением ее типа. Имя массива, секции массива или элемента массива должно быть определено как имя массива в одном из предыдущих операторов спецификации.

Исключая переменные из именованных общих блоков, если переменная или какая-либо ее часть инициализируется в операторе DATA, переменная имеет атрибут SAVE, что может быть подтверждено включением ее имени в оператор SAVE или в оператор объявления типа, содержащий атрибут SAVE.

R530 *множество-оператора-DATA* **есть** *список-объектов-оператора-DATA* /*список-значений-оператора-DATA*/

R531 *объект-оператора-DATA* **есть** *переменная*
или *неявный-цикл-DATA*

R532 *значение-оператора-DATA* **есть** [*повторитель-оператора-DATA* *] *константа-оператора-DATA*

R533 *константа-оператора-DATA* **есть** *скалярная-константа*
или *целая-литеральная-константа-со-знаком*
или *вещественная-литеральная-константа-со-знаком*
или *конструктор-структуры*
или *BOZ-литеральная-константа*

R534 *повторитель-оператора-DATA* **есть** *скалярная-целая-константа*

R535 *неявный-цикл-DATA* **есть** (*список-объектов-нцкл-DATA*, *переменная-нцкл-DATA* = *скалярное-целое-выражение*, *скалярное-целое-выражение* [,*скалярное-целое-выражение*])

R536 *объект-нцкл-DATA* **есть** *элемент-массива*
или *скалярный-компонент-структуры*
или *неявный-цикл-DATA*

Ограничение: *элемент-массива* не должен иметь в качестве основы константу.

Ограничение: *скалярный-компонент-структуры* не должен иметь в качестве основы константу.

R537 *переменная-нцкл-DATA* **есть** *скалярная-целая-переменная*

Ограничение: *переменная-нцкл-DATA* должна быть именованной переменной.

Ограничение: повторитель оператора DATA должен быть положительным или равным нулю. Если повторитель - именованная константа, он должен быть предварительно объявлен в этой контекстной области или сделан доступным через связь по объемлющему контексту или использованию.

Ограничение: если *константа-оператора-DATA* является *конструктором-структуры*, каждый его компонент должен быть выражением инициализации.

Ограничение: в переменной, которая является *объектом-оператора-DATA*, все индексы, индексы секции, начальные и конечные точки подстроки должны быть выражениями инициализации.

Ограничение: переменная, имя или обозначение которой включены в *список-объектов-оператора-DATA* или *список-объектов-нцкл-DATA*, не может быть формальным аргументом, объектом, доступ к которому реализован через связь по использованию или по объемлющему контексту, объектом из именованного общего блока (если только оператор DATA не находится в программной единице-блоке данных), объектом из неименованного общего блока, именем результата функции, именем функции, автоматически размещаемым объектом, указателем или динамически размещаемым массивом.

Ограничение: все индексы в *элементе-массива* или *скалярном-компоненте-структуры*, которые являются *объектами-нцкл-DATA*, должны быть выражениями, в которых все первичные - только константы или переменные объемлющих неявных циклов, а все операции - встроенные.

Ограничение: *скалярное-целое-выражение* в неявном цикле должно содержать в качестве первичных только константы или переменные объемлющих *неявных-циклов-DATA*, а каждая операция должна быть встроенной.

Список-объектов-оператора-DATA разворачивается в последовательность скалярных переменных. Если в *списке-объектов-оператора-DATA* появляется имя массива без индексов, оно эквивалентно полной последовательности элементов массива в принятом для массива порядке (6.2.2.2). Секция массива эквивалентна последо-

вательности элементов массива в принятом для массивов порядке. *Неявный-цикл-DATA* разворачивается в последовательность элементов массива и компонентов структуры под управлением переменной неявного цикла аналогично оператору DO (8.1.4.4).

Заметим, что массивы нулевого размера и списки неявного цикла с нулевым счетчиком числа итераций не требуют переменных, чтобы развернуть последовательность скалярных переменных, тогда как символьная переменная нулевой длины требует переменную в списке.

Список-значений-оператора-DATA разворачивается в последовательность скалярных константных значений. Каждое значение должно быть константой, либо предварительно определенной, либо доступной через связь по использованию или по объемлющему контексту. Повторитель указывает, сколько раз последующая константа должна быть включена в последовательность; отсутствие повторителя эквивалентно значению 1. Заметим, что значения с нулевым повторителем не требуют значений в развернутой последовательности константных скалярных значений.

Развернутые последовательности скалярных переменных и константных значений находятся во взаимно-однозначном соответствии. Каждая константа определяет начальное значение соответствующей переменной. Длины этих последовательностей должны совпадать.

Тип объекта и тип соответствующей константы должны совпадать, если это тип символьный, производный или логический.

Если объект вещественного или комплексного типа, соответствующая константа должна быть целой, вещественной или комплексной. Если объект целого типа, соответствующая константа должна быть либо целого, вещественного или комплексного типа, либо двоичной, восьмеричной или шестнадцатеричной. Если объект производного типа, соответствующая константа должна быть того же типа.

Значение константы должно согласовываться с соответствующей переменной по правилам встроенного присваивания (7.5.1.4), и по этим правилам константа определяет начальное значение переменной.

Примеры операторов DATA:

```
CHARACTER (LEN=10) NAME
INTEGER, DIMENSION (0:9) :: MILES
REAL, DIMENSION (100,100) :: SKEW
TYPE (PERSON) MYNAME, YOURNAME
DATA NAME /'JOHN DOE'/, MILES /10 * 0/
DATA ((SKEW (K,J), J = 1, K), K=1, 100) /5050 * 0.0/
DATA ((SKEW (K,J), J = K+1, 100), K=1, 99) /4950 * 1.0/
DATA MYNAME /PERSON (21, 'JOHN SMITH') /
DATA YOURNAME % AGE, YOURNAME % NAME /35,'FRED BROWN'/
```

Символьная переменная NAME инициализируется значением 'JOHN DOE' с добавлением пробелов справа, поскольку длина константы меньше длины переменной. Все десять элементов целого массива MILES получают нулевые значения, а двумерный массив SKEW инициализируется таким образом, что нижний треугольник SKEW становится нулевым, а верхний - единичным. Структуры MYNAME и YOURNAME объявлены с типом PERSON из 4.4.1. MYNAME инициализируется при помощи конструктора структуры, а YOURNAME - посредством задания отдельных значений для каждого компонента.

5.2.10. Оператор PARAMETER

Оператор PARAMETER предоставляет средство описания именованных констант. Именованные константы, описанные в операторе PARAMETER, имеют точно те же свойства и ограничения, как те, что описываются в операторе, специфицирующем атрибут PARAMETER (5.1.2.1).

R538 *оператор-PARAMETER* **есть** PARAMETER ♦
♦ (*список-описаний-именованных-констант*)

R539 *описание-именованной-константы* **есть** *именованная-константа* ♦
♦ = *выражение-инициализации*

Тип, параметры типа и конфигурация именованной константы должны быть либо специфицированы предварительным вхождением в оператор объявления типа в этой же контекстной области, либо определены по правилам неявной типизации, действующим в контекстной области в данный момент. Если тип именованной константы определяется по правилам неявной типизации, ее вхождение в любой последующий оператор объявления типа должно согласовываться с этим неявно заданным типом и значениями любых неявно заданных параметров типа.

Каждая именованная константа становится определенной, и ее значение определяется выражением инициализации, находящимся справа от знака равенства в соответствии с правилами встроенного присваивания (7.5.1.4).

Именованная константа, входящая в выражение инициализации, должна быть либо предварительно определена в этом же операторе PARAMETER или в предыдущих операторах PARAMETER или в операторах объявления типа, включающих атрибут PARAMETER, либо быть доступна через связь по использованию или по объемлющему контексту.

Именованная константа не может использоваться как часть спецификации формата (10.1.1).

Каждая именованная константа имеет атрибут PARAMETER.

Пример оператора PARAMETER:

```
PARAMETER (MODULUS = MOD (28, 3), NUMBER_OF_SENATORS = 100)
```

5.3. Оператор IMPLICIT

В контекстной области **оператор IMPLICIT** специфицирует тип и, возможно, параметры типа для всех неявно типизированных объектов данных, имена которых начинаются с одной из букв, указанных в операторе. Он же может указывать, что в данной контекстной области правила неявной типизации не применимы.

R540 *оператор-IMPLICIT* **есть** IMPLICIT *список-специф-IMPLICIT*
или IMPLICIT NONE

R541 *специф-IMPLICIT* **есть** *специф-типа* (*список-специф-буквы*)

R542 *специф-буквы* **есть** *буква* [- *буква*]

Ограничение: если в контекстной области есть оператор IMPLICIT NONE, он должен предшествовать всем операторам PARAMETER, которые находятся в этой контекстной области, и в этой контекстной области больше не должно быть других операторов IMPLICIT.

Ограничение: если есть минус и вторая буква, вторая буква должна следовать за первой в алфавитном порядке.

Спецификатор-буквы, состоящий из двух букв, разделенных минусом, эквивалентен написанию всех букв в алфавитном порядке, начиная с первой буквы и кончая второй буквой. Например, A-C эквивалентно A, B, C. Одна и та же буква, либо взятая отдельно, либо включенная в диапазон, не может появиться в операторах IMPLICIT в данной контекстной области более одного раза.

В каждой контекстной области существует отображение (оно может быть нулевым) между каждой из букв A, B, ..., Z и типом (и параметрами типа). Оператор IMPLICIT специфицирует такое отображение для букв из *списка-спецификаторов-буквы*. Оператор IMPLICIT NONE обозначает нулевое отображение для всех букв. Если для буквы не специфицировано отображение, оно принимается по умолчанию из объемлющей контекстной области. Программная единица обрабатывается так, как если бы она имела объемлющий контекст с объявлением:

```
IMPLICIT INTEGER (I-N), REAL (A-H, O-Z)
```

Любой объект данных, который не объявлен явно в операторе объявления типа, не является встроенной функцией и не стал доступным через связь по использованию или по объемлющему контексту, является объявленным неявно, причем его тип (и параметры типа) отображаются по первой букве его имени (при условии, что отображение не нулевое). Такой объект данных обрабатывается так, как если бы он был объявлен в явном объявлении типа в контекстной области, внешней по отношению к той, где он находится. Явная спецификация типа в операторе FUNCTION отменяет действие оператора IMPLICIT на имя этой функции.

Примеры использования операторов IMPLICIT:

```
MODULE EXAMPLE_MODULE
     IMPLICIT NONE
     ...
     INTERFACE
         FUNCTION FUN (I)          ! Все объекты данных
            INTEGER FUN, I         ! должны быть объявлены явно
         END FUNCTION FUN
     END INTERFACE
     CONTAINS
     FUNCTION JFUN(J)              ! Все объекты данных
         INTEGER JFUN, J           ! должны быть объявлены явно
         ...
     END FUNCTION JFUN
END MODULE EXAMPLE_MODULE

SUBROUTINE SUB
     IMPLICIT COMPLEX(C)
     C=(3.0, 2.0)                  ! C неявно объявлено как COMPLEX
     ...
     CONTAINS
     SUBROUTINE SUB1
         IMPLICIT INTEGER(A, C)
         C=(0.0, 0.0)              ! C через связь по объемлющему контексту
                                   ! объявлено как COMPLEX
         Z=1.0                     ! Z неявно объявлено как REAL
         A=2                       ! A неявно объявлено как INTEGER
         CC=1                      ! CC неявно объявлено как INTEGER
     END SUBROUTUNE SUB1

     SUBROUTINE SUB2
         Z=2.0                     ! Z неявно объявлено как REAL и
                                   ! отличается от одноименной
                                   ! переменной из SUB1
         ...
     END SUBROUTINE SUB2

     SUBROUTINE SUB3
         USE EXAMPLE_MODUL         ! Доступ к целой функции FUN через
                                   ! связь по использованию
         Q=FUN(K)                  ! Q неявно объявлено как REAL,
         ...                       ! а K - как INTEGER
     END SUBROUTINE SUB3
END SUBROUTINE SUB
```

Оператор IMPLICIT может специфицировать *спецификатор-типа* для производного типа. Например, задан оператор IMPLICIT, и тип описан следующим образом:

```
IMPLICIT TYPE (POSN) (A-B, W-Z), INTEGER (C-V)
```

```
TYPE POSN
      REAL X, Y
      INTEGER Z
END TYPE POSN
```

Переменные, начинающиеся с букв A, B, W, X, Y и Z, неявно объявляются как имеющие тип POSN, а остальные переменные неявно объявляются, как INTEGER.

5.4. Оператор NAMELIST

Оператор NAMELIST специфицирует группу именованных объектов данных, которые можно объединить в операторах передачи данных под одним именем (9.4, 10.9).

R543 *оператор-NAMELIST* **есть** NAMELIST ♦
♦ /*имя-группы-NAMELIST*/ *список-объектов-группы* ♦
♦ [[,]/*имя-группы-NAMELIST*/ *список-объектов-группы*]...

R544 *объект-группы* **есть** *имя-переменной*

Ограничение: *объект-группы* не должен быть формальным аргументом-массивом с неконстантной границей, переменной с неконстантной длиной (для символьных), автоматически размещаемым объектом, указателем, переменной, тип которой имеет указатель в качестве конечного компонента, или динамически размещаемым массивом.

Ограничение: если *имя-группы-NAMELIST* имеет атрибут PUBLIC, ни один из элементов *списка-объектов-группы* не может иметь атрибут PRIVATE.

Последовательность перечисления объектов (переменных) в операторе NAMELIST определяет порядок, в котором значения появляются при выводе.

Любое *имя-группы-NAMELIST* может находиться в более чем одном операторе NAMELIST в контекстной области. *Список-объектов-группы*, следующий за каждым вхождением одного и того же *имени-группы-NAMELIST* в данной контекстной области, воспринимается как продолжение списка для этого *имени-группы-NAMELIST*.

Объект группы может быть членом нескольких таких групп.

Объект группы должен либо быть доступен через связь по использованию или по объемлющему контексту, либо иметь тип, параметры типа и конфигурацию, предварительно объявленные в этой же контекстной области или определенные по правилам неявной типизации, действующим в этой контекстной области в данный момент. Если тип определен по правилам неявной типизации, любое последующее вхождение имени в оператор объявления типа должно согласовываться с неявно определенным типом.

Пример оператора NAMELIST:

```
NAMELIST /NLIST/A, B, C
```

5.5. Связь объектов данных по памяти

В общем случае физические единицы памяти и их порядок для объектов данных не специфицированы. Однако есть операторы EQUIVALENCE, COMMON и SEQUENCE, предназначенные для управления порядком и расположением единиц памяти. Основной механизм связи по памяти описан в 14.6.3.

5.5.1. Оператор EQUIVALENCE

Оператор EQUIVALENCE используется для задания совместного использования единиц памяти двумя или более объектами в контекстной области, что приводит к связи этих объектов по памяти.

Если эквивалентные объекты имеют разные типы или параметры типа, оператор EQUIVALENCE не вызывает преобразования типов и не подразумевает математической эквивалентности.

Если эквивалентны скаляр и массив, скаляр не приобретает свойства массива и наоборот.

R545 *оператор-EQUIVALENCE* **есть** EQUIVALENCE ♦
♦ *список-множеств-эквивалентности*

R546 *множество-эквивалентности* **есть** (*эквивалентный-объект*, ♦
♦ *список-эквивалентных-объектов*)

R547 *эквивалентный-объект* **есть** *имя-переменной*
или *элемент-массива*
или *подстрока*

Ограничение: *эквивалентный-объект* не может быть формальным аргументом, указателем, динамически размещаемым массивом, объектом непоследовательного производного типа или последовательного производного типа, содержащего указатель на любом уровне отбора компонентов, автоматически размещаемым объектом, именем функции, именем входа или результата, именованной константой, компонентом структуры, либо подобъектом любого из перечисленных выше объектов.

Ограничение: каждое индексное выражение или выражение диапазона подстроки в *эквивалентном-объекте* должно быть целым выражением инициализации (7.1.6.1).

Ограничение: если *эквивалентный-объект* относится к стандартному целому, стандартному вещественному, вещественному двойной точности, стандартному комплексному, стандартному логическому или к числовому последовательному типу, все объекты в множестве эквивалентности должны относиться к тем же типам.

Ограничение: если *эквивалентный-объект* относится к стандартному символьному или к символьному последовательному типу, все объекты в множестве эквивалентности должны принадлежать к тем же типам.

Ограничение: если *эквивалентный-объект* принадлежит к производному типу, который не является ни числовым последовательным, ни символьным последовательным, все объекты в множестве эквивалентности должны относиться к этому же типу.

Ограничение: если *эквивалентный-объект* относится к встроенному типу, который не является ни стандартным целым, ни стандартным вещественным, ни вещественным двойной точности, ни стандартным комплексным, ни стандартным логическим, ни стандартным символьным, все объекты в множестве эквивалентности должны быть того же типа и с теми же значениями параметров вида.

5.5.1.1. Связь по эквивалентности

Оператор EQUIVALENCE означает, что последовательности памяти (14.6.3.1) объектов, специфицированных в *множестве-эквивалентности*, связываются по памяти. Все последовательности ненулевого размера (если они есть) имеют одну и ту же первую единицу памяти, а все последовательности нулевого размера (если они есть) связываются по памяти друг с другом и с первой единицей памяти последовательностей ненулевого размера. Это приводит к связи по памяти объектов из *множества-эквивалентности* и, может быть, других объектов данных.

5.5.1.2. Эквивалентность объектов стандартного символьного типа

Объекты данных стандартного символьного типа могут быть эквивалентны другим объектам только стандартного символьного типа. Длины объектов не обязательно должны быть равными.

Оператор EQUIVALENCE означает, что последовательности памяти всех символьных объектов данных, специфицированных в *множестве-эквивалентности*, связаны по памяти. Все последовательности ненулевого размера (если они есть) в *множестве-эквивалентности* имеют общую первую символьную единицу памяти, а все последовательности нулевого размера (если они есть) в *множестве-эквивалентности* связываются по памяти друг с другом и с первой символьной единицей памяти ненулевых последовательностей. Это вызывает связь по памяти объектов данных из *множества-эквивалентности* и, может быть, других объектов данных. Например, используя объявления:

```
CHARACTER (LEN=4) :: A,B
CHARACTER (LEN=3) :: C(2)
EQUIVALENCE (A,C(1)), (B,C(2))
```

можно проиллюстрировать графически связь A, B и C следующим образом:

```
|     |     |     |     |     |     |     |
0     1     2     3     4     5     6     7
|-------- A --------|
                    |-------- B --------|
|---- C(1) ----|---- C(2) ----|
```

5.5.1.3. Имена массивов и обозначение элементов массива

Использование в операторе EQUIVALENCE имени массива ненулевого размера без списка индексов эквивалентно использованию обозначения элемента массива, идентифицирующего первый его элемент.

5.5.1.4. Ограничения на оператор EQUIVALENCE

Оператор EQUIVALENCE не должен специфицировать отведение одной и той же единицы памяти более чем одной последовательности памяти. Например, последовательность операторов:

```
REAL, DIMENSION (2) :: A
REAL :: B
EQUIVALENCE (A(1),B),(A(2),B)          ! Не соответствует стандарту
```

запрещена, поскольку одна и та же единица памяти должна отводиться и для A(1), и для A(2).

Оператор EQUIVALENCE не должен разъединять последовательные единицы памяти. Запрещено, например, следующее:

```
REAL A(2)
DOUBLE PRECISION D(2)
EQUIVALENCE (A(1),D(1)), (A(2),D(2))          ! Не соответствует
                                              ! стандарту
```

5.5.2. Оператор COMMON

Оператор COMMON описывает блоки физической памяти, называемые **общими блоками**, которые могут быть доступны для любой контекстной области в выполняемой программе. Таким образом, оператор COMMON обеспечивает средство глобализации данных, основанное на связи по памяти (14.6.3). Общий блок, специфицированный оператором COMMON, может иметь имя и называется тогда **именованным общим блоком** или не иметь его, тогда это **неименованный общий блок**.

R548 *оператор-COMMON* **есть** COMMON [/[*имя-общего блока*]/] ♦
♦ *список-объектов-общего-блока* ♦
♦ [[,]/[*имя-общего-блока*] /*список-объектов-общего-блока*]...

R549 *объект-общего-блока* **есть** *имя-переменной* ♦
♦ [(*список-специф-явной-конфигурации*)]

Ограничение: внутри контекстной области разрешено только одно вхождение *имени-переменной* в *списки-объектов-общего-блока*.

Ограничение: *объект-общего-блока* не должен быть формальным аргументом, автоматически размещаемым объектом, динамически размещаемым массивом, именем функции, именем входа или именем результата.

Ограничение: каждая граница в *спецификаторе-явной-конфигурации* должна быть константным выражением спецификации (7.1.6.2).

Ограничение: если *объект-общего-блока* - производного типа, тип должен быть последовательным (4.4.1).

Ограничение: если *имя-переменной* сопровождается *списком-спецификаторов-явной-конфигурации*, оно не должно иметь атрибута POINTER.

В каждом операторе COMMON объекты данных, имена которых входят в список объектов общего блока, следующий за именем общего блока, объявляются входящими в этот общий блок. Если первое имя общего блока опущено, все объекты данных, имена которых входят в первый список объектов общего блока, объявляются входящими в неименованный общий блок. Два символа "/" без имени общего блока между ними означают, что все объекты данных, имена которых включены в последующий список объектов общего блока, входят в неименованный общий блок.

Любое имя общего блока, в том числе опущенное для неименованного общего блока, может входить в контекстной области несколько раз в один оператор COMMON и в несколько операторов COMMON. Список объектов общего блока, следующий за каждым очередным появлением одного и того же имени общего блока в контекстной области, рассматривается как продолжение списка для этого общего блока. Аналогично каждый список объектов неименованного общего блока в контекстной области трактуется как продолжение предыдущего списка для неименованного общего блока.

Форма *имя-переменной* (*список-спецификаторов-явной-конфигурации*) объявляет, что *имя-переменной* имеет атрибут DIMENSION и специфицирует характеристики массива. Если в общем блоке находятся объекты числового последовательного типа (4.4.1) или символьного последовательного типа (4.4.1), это эквивалентно прямому перечислению отдельных компонентов в списке общего блока.

Примеры операторов COMMON:

```
COMMON /BLOCKA/ A, B, D(10,30)
COMMON I, J, K
```

5.5.2.1. Последовательность памяти общего блока

Для каждого общего блока последовательность памяти общего блока формируется следующим образом.

1. Последовательность памяти образуется как последовательность единиц памяти, содержащихся в последовательности памяти (14.6.3.1) всех объектов данных в списках объектов общего блока для данного общего блока. Порядок следования последовательностей памяти совпадает с порядком появления объектов в списках общего блока в контекстной области.
2. Последовательность памяти, полученная по описанному выше принципу, расширяется за счет включения единиц из других последовательностей памяти, связанных с данной последовательностью с помощью связи по эквивалентности. Последовательность может быть расширена только путем добавления после последней единицы памяти. Объекты данных, связанные с объектом из общего блока, тоже считаются входящими в этот общий блок.

5.5.2.2. Размер общего блока

Размер общего блока - это размер его последовательности памяти с учетом всех расширений за счет связи по эквивалентности.

5.5.2.3. Связь по общему блоку

Внутри выполняемой программы последовательности памяти всех общих блоков ненулевого размера с одинаковым именем начинаются с одной и той же единицы памяти, а все последовательности памяти одноименных общих блоков нулевого размера связаны друг с другом. Внутри выполняемой программы последовательности памяти всех неименованных общих блоков ненулевого размера начинаются с одной единицы памяти, а последовательности всех неименованных общих блоков нулевого размера связаны друг с другом и с первой единицей памяти неименованных общих блоков ненулевого размера. Это приводит к связи объектов из разных контекстных областей.

Объект (не указатель) стандартного целого, стандартного вещественного, вещественного двойной точности, стандартного комплексного, стандартного логического или числового последовательного типа может быть связан только с объектами (не указателями) тех же типов.

Объект (не указатель) стандартного символьного или символьного последовательного типа может быть связан только с объектами (не указателями) этих же типов.

Объект (не указатель) производного типа, который не является ни числовым последовательным типом, ни символьным последовательным типом, может быть связан только с объектами (не указателями) этого же типа.

Объект (не указатель) встроенного, но не стандартного целого, не стандартного вещественного, не вещественного двойной точности, не стандартного комплексного, не стандартного логического и не стандартного символьного типа может быть связан только с объектами того же типа и с теми же параметрами типа.

Указатель может связываться по памяти только с указателями того же типа, с теми же параметрами типа и с той же размерностью.

5.5.2.4. Разница между именованным и неименованным общими блоками

Неименованный общий блок имеет все те же свойства, что именованный, исключая следующее:

1) выполнение оператора RETURN или END может привести к тому, что становятся неопределенными все объекты именованных общих блоков, кроме тех, для которых имя общего блока входит в оператор SAVE, тогда как объекты из неименованных общих блоков никогда не становятся неопределенными (14.7.6);
2) одноименные общие блоки должны иметь одинаковый размер во всех контекстных областях выполняемой программы, в которых они есть, тогда как неименованные общие блоки могут быть разных размеров;
3) объект данных из именованного общего блока может быть инициализирован в операторе DATA или в операторе объявления типа в программной единице-блоке данных, тогда как объекты из неименованного общего блока нельзя инициализировать (11.4).

5.5.2.5. Ограничения на COMMON и EQUIVALENCE

Оператор EQUIVALENCE не должен приводить к связи последовательностей памяти двух разных общих блоков. Связь по эквивалентности не должна вызывать расширения последовательности памяти общего блока путем добавления единиц памяти перед первой единицей памяти первого объекта, специфицированного в операторе COMMON для общего блока.

Например, запрещено следующее:

```
COMMON /X/ A
REAL B(2)
EQUIVALENCE (A, B(2))          ! Не соответствует стандарту
```

Общий блок может быть объявлен в контекстной области модуля (11.3). Если это так, его нельзя объявлять в другой контекстной области, имеющей доступ к объектам из модуля через связь по использованию.

Глава 6. ИСПОЛЬЗОВАНИЕ ОБЪЕКТОВ ДАННЫХ

Появление имени объекта данных или обозначения подобъекта в контексте, где требуется его значение, называется ссылкой. Ссылка разрешена только если объект данных определен. Ссылка на указатель разрешена только тогда, когда он связан с объектом-адресатом, который уже определен. Объект данных становится определенным и получает значение, когда имя объекта или обозначение подобъекта появляется в конкретном контексте или происходят конкретные события (14.7).

R601 *переменная* **есть** *имя-скалярной-переменной*
или *имя-переменной-массива*
или *подобъект*

Ограничение: *имя-переменной-массива* должно быть именем объекта данных, являющегося массивом.

Ограничение: *имя-переменной-массива* не должно иметь атрибута PARAMETER.

Ограничение: *имя-скалярной-переменной* не должно иметь атрибута PARAMETER.

Ограничение: *подобъект* не должен быть обозначением подобъекта (например, подстроки), основой которого является константа.

R602 *подобъект* **есть** *элемент-массива*
или *секция-массива*
или *компонент-структуры*
или *подстрока*

R603 *логическая-переменная* **есть** *переменная*

Ограничение: *логическая-переменная* должна быть логического типа.

R604 *переменная-станд-логич-типа* **есть** *переменная*

Ограничение: *переменная-станд-логич-типа* должна быть переменной стандартного логического типа.

R605 *символьная-переменная* **есть** *переменная*

Ограничение: *символьная-переменная* должна быть символьного типа.

R606 *переменная-станд-симв-типа* **есть** *переменная*

Ограничение: *переменная-станд-симв-типа* должна быть переменной стандартного символьного типа.

R607 *целая-переменная* **есть** *переменная*

Ограничение: *целая-переменная* должна быть целого типа.

R608 *переменная-станд-целого-типа* **есть** *переменная*

Ограничение: *переменная-станд-целого-типа* должна быть переменной стандартного целого типа.

Указатели и динамически размещаемые массивы не могут быть определены при обстоятельствах, приведенных в 5.1.2.4.3. Формальные аргументы и связанные с ними переменные нельзя определять при обстоятельствах, приведенных в 12.5.2.1 и 12.5.2.8.

Литеральная константа - это скаляр, обозначаемый синтаксической формой, которая указывает его тип, параметры типа и значение. Именованная константа - это константа, связанная с именем при помощи атрибута PARAMETER (5.1.2.1, 5.2.10). Ссылка на константу разрешена всегда, переопределение константы никогда не разрешается.

Пусть, например, имеются объявления:

```
CHARACTER (10) A, B(10)
TYPE (PERSON) P                    ! См.4.4.1.
```

тогда A, B, B(1), B(1:5), P%AGE и A(1:1) - переменные.

6.1. Скаляры

Скаляр (2.4.4) - это объект данных, который может быть представлен отдельным значением, и не является массивом (6.2). Его значение, если оно определено, является отдельным элементом из множества значений, характеризующих его тип.

Скаляр имеет размерность равную нулю.

6.1.1. Подстроки

Подстрока - это непрерывная часть символьной строки (4.3.2.1). Форму подстроки определяют следующие правила:

R609 *подстрока*	**есть** *строка-основа* (*диапазон-подстроки*)
R610 *строка-основа*	**есть** *имя-скалярной-переменной* **или** *элемент-массива* **или** *скалярный-компонент-структуры* **или** *скалярная-константа*
R611 *диапазон-подстроки*	**есть** [*скалярное-целое-выражение*] : ♦ ♦ [*скалярное-целое-выражение*]

Ограничение: *строка-основа* должна быть символьного типа.

Первое *скалярное-целое-выражение*, входящее в *диапазон-подстроки*, называется **начальной точкой**, второе - **конечной точкой**. Длина подстроки - это число символов в подстроке и равна она MAX(l-f+1, 0), где f и l соответственно, начальная и конечная точки.

Предположим, что символы в строке-основе пронумерованы 1,2,3,...,N, где N - длина строки. Тогда в подстроку входят символы строки-основы, начиная с начальной точки и последовательно до конечной точки (включая ее). И начальная, и конечная точки должны находиться в диапазоне 1,2,...,N, если только начальная точка не больше конечной, в этом случае подстрока имеет нулевую длину. Если начальная точка не указана, подразумевается значение, равное 1; если не указана конечная, - подразумевается значение, равное N.

Если основой является переменная, подстрока тоже является переменной.

Примеры символьных подстрок:

`B(1)(1:5)`	строка-основа - элемент массива
`P%NAME(1:1)`	строка-основа - компонент структуры
`ID(4:9)`	имя скалярной переменной в качестве строки-основы
`'0123456789'(N:N)`	символьная константа в качестве строки-основы

6.1.2. Компоненты структуры

Компонент-структуры - это один из компонентов структуры (4.4) или массив, элементами которого являются компоненты элементов массива производного типа.

R612 *ссылка-на-данное*	**есть** *ссылка-на-часть* [% *ссылка-на-часть*]...
R613 *ссылка-на-часть*	**есть** *имя-части*[(*список-индексов-секции*)]

Ограничение: в *ссылке-на-данное* каждое *имя-части*, кроме самого правого, должно быть производного типа.

Ограничение: в *ссылке-на-данное* каждое *имя-части*, кроме самого левого, должно быть именем компонента из описания производного типа для предшествующего *имени-части*.

Ограничение: в *ссылке-на-часть*, содержащей *список-индексов-секции*, число *индексов-секции* должно быть равно размерности *имени-части*.

Размерность *ссылки-на-часть*, если она имеет вид - *имя-части*, равна размерности *имени-части*. Размерность *ссылки-на-часть*, имеющей список индексов секции, равна числу индексных триплетов и векторных индексов в списке.

Ограничение: в *ссылке-на-данное* должно быть не более одной *ссылки-на-часть* с ненулевой размерностью. *Имя-части*, стоящее справа в *ссылке-на-часть* с ненулевой размерностью, не должно иметь атрибута POINTER.

Размерность *ссылки-на-данное* - это размерность *ссылки-на-часть* с ненулевой размерностью, если таковая имеется, иначе размерность равна нулю. **Объект-основа** для *ссылки-на-данное* - это объект данных, имя которого является именем самой левой из частей.

R614 *компонент-структуры* **есть** *ссылка-на-данное*

Ограничение: в *компоненте-структуры* должно быть более одной *ссылки-на-часть* и самая правая из них должна иметь вид: *имя-части*.

Тип и параметры типа (если они есть) компонента структуры - те же, что у имени самой правой части. Нельзя ссылаться на компонент структуры и определять его до объявления объекта-основы. Компонент структуры имеет атрибут INTENT, TARGET или PARAMETER, если такой атрибут есть у объекта-основы. Компонент структуры является указателем, только если имя самой правой из частей было объявлено с атрибутом POINTER.

Примеры компонентов структуры:

`SCALAR_PARENT % SCALAR_FIELD`	скалярный компонент скалярной основы
`ARRAY_PARENT(J) % SCALAR_FIELD`	компонент основы - элемента массива
`ARRAY_PARENT(1:N) % SCALAR_FIELD`	компонент основы - секции массива

6.2. Массивы

Массив - это множество скалярных данных одного типа с одинаковыми параметрами типа, элементы которого упорядочены по прямоугольной схеме. Скалярные объекты, образующие массив, являются **элементами массива**.

При ссылках на элементы массива, определяемые вхождением имени массива в обозначения, возможен единственный порядок элементов массива, приведенный в (6.2.2).

6.2.1. Полный массив

Полный массив (массив целиком) - это именованный массив.

Полный массив - это либо переменная, либо именованная константа. **Полный массив-именованная константа** - это имя константного выражения (5.1.2.1 и 5.2.10), которое является массивом. **Полный массив-переменная** - это имя переменной, которая является массивом. Это имя не должно дополняться списком индексов.

Появление имени полного массива в выполняемом операторе специфицирует все элементы массива (2.4.5). Массив неопределенного размера в виде полного массива допускается в выполняемых конструкциях только как фактический аргумент в ссылке на процедуру, в которой не требуется конфигурация массива.

Появление имени полного массива в невыполняемой конструкции специфицирует весь массив.

6.2.2. Элементы и секции массива

R615 *элемент-массива* **есть** *ссылка-на-данное*

Ограничение: в *элементе-массива* каждая *ссылка-на-часть* должна иметь нулевую размерность, а последняя *ссылка-на-часть* должна содержать *список-индексов*.

R616 *секция-массива* **есть** *ссылка-на-данное* [(*диапазон-подстроки*)]

Ограничение: в *секции-массива* ровно одна *ссылка-на-часть* должна иметь ненулевую размерность и либо последняя *ссылка-на-часть* имеет *список-индексов-секции* с ненулевой размерностью, либо другая *ссылка-на-часть* имеет ненулевую размерность.

Ограничение: в *секции-массива* с *диапазоном-подстроки* самое правое *имя-части* должно быть символьного типа.

R617 *индекс* **есть** *скалярное-целое-выражение*

R618 *индекс-секции* **есть** *индекс*
или *индексный-триплет*
или *векторный-индекс*

R619 *индексный-триплет* **есть** [*индекс*] : [*индекс*] [: *шаг*]

R620 *шаг* **есть** *скалярное-целое-выражение*

R621 *векторный-индекс* **есть** *целое-выражение*

Ограничение: *векторный-индекс* должен быть выражением-массивом размерности 1.

Ограничение: нельзя опускать второй *индекс* в *индексном-триплете* последнего измерения массива неопределенного размера.

Элемент массива - это скаляр. Секция массива является массивом. Если в *секции-массива* есть *диапазон-подстроки*, каждый элемент является обозначенной подстрокой соответствующего элемента секции массива.

Например, если объявлено:

```
REAL A(10,10)
CHARACTER (LEN=10) B(5,5,5)
```

то A(1,2) - элемент массива, A(1:N:2,M) - одномерная секция массива, а B(:, :, :)(2:3) - массив конфигурации (5,5,5), элементами которого являются подстроки длины 2 соответствующих элементов массива B. Элемент массива или секция массива имеют атрибуты INTENT, TARGET или PARAMETER, если их имеет основа, но они никогда не имеют атрибута POINTER.

Примеры элементов и секций массива:

```
ARRAY_A(1:N:2)% ARRAY_B (I,J)% STRING(K)(:)      - секция массива
SCALAR_PARENT % ARRAY-FIELD (J)                  - элемент массива
SCALAR-PARENT % ARRAY-FIELD(1:N)                 - секция массива
SCALAR_PARENT %ARRAY_FIELD(1:N)%SCALAR_FIELD     - секция массива
```

6.2.2.1. Элементы массива

Значение индекса в элементе массива должно находиться в границах этого измерения.

6.2.2.2. Последовательность элементов массива

Элементы массива образуют последовательность, известную как **последовательность элементов массива**. Позиция элемента массива в этой последовательности определяется порядковым значением индекса из списка индексов, обозначающего данный элемент. Порядковое значение индекса вычисляется по формулам из табл. 6.1.

Таблица 6.1

Порядковое значение индекса

Размерность	Границы индексов	Список индексов	Порядковое значение индекса
1	$j_1 : k_1$	s_1	$1 + (s_1 - j_1)$
2	$j_1:k_1, j_2:k_2$	s_1,s_2	$1 + (s_1 - j_1)$ $+ (s_2 - j_2)*d_1$
3	$j_1:k_1 j_2:k_2 j_3:k_3$	s_1,s_2,s_3	$1 + (s_1-j_1)$ $+ (s_2-j_2)*d_1$ $+ (s_3-j_3)*d_2*d_1$
- - -	- - -	- - -	- - -
7	$j_1:k_1,...,j_7:k_7$	$s_1,...,s_7$	$1 + (s_1-j_1)$ $+ (s_2-j_2)*d_1$ $+ (s_3-j_3)*d_2*d_1$ $+ ...$ $+ (s_7-j_7)*d_6*d_5*...*d_1$

Примечания к табл. 6.1:

1) d_i = max $(k_i-j_i+1, 0)$ - размер i-го измерения;
2) если размер массива не нулевой, $j_i <= s_i <= k_i$ для всех $i = 1,2,...,7$.

6.2.2.3. Секции массива

Секция массива - это подобъект массива, за которым может следовать диапазон подстроки.

В секции массива, имеющей *список-индексов-секции*, каждый *индексный-триплет* или *векторный-индекс* в списке индексов секции обозначает последовательность индексов, которая может быть и пустой (6.2.2). Если последовательность не пуста, каждый индекс в ней должен находиться внутри границ своего измерения. Секция массива - это множество элементов массива, определяемое всеми возможными списками индексов, полученными из отдельных индексов или последовательностей индексов, специфицированных каждым индексом секции.

В секции массива без *списка-индексов-секции* размерность и конфигурация массива такие же, как у *ссылки-на-часть* с ненулевой размерностью, в противном случае размерность секции массива - это число индексных триплетов и векторных индексов в списке индексов секции. Конфигурация - это одномерный массив, i-м элементом которого является число целых значений в последовательности, определяемой i-м индексным триплетом или векторным индексом. Если какая-либо из этих последовательностей пуста, секция массива имеет нулевой размер. Индексный порядок элементов секции массива такой же, как у объекта-массива, т.к. секция массива представляет собой массив.

6.2.2.3.1. Индексный триплет

Индексный триплет обозначает регулярную последовательность индексов, включающую нуль или более индексных значений. Третье выражение в индексном триплете - это приращение индексного значения, называемое **шагом**.

Индексы и шаг в индексном триплете являются не обязательными. Если первый индекс в индексном триплете опущен, это эквивалентно тому, что значением индекса является нижняя граница для данного массива, а если опущен второй индекс, это эквивалентно верхней границе. Если опущен шаг, это эквивалентно шагу, равному единице.

Нельзя опускать второй индекс в последнем измерении массива неопределенного размера.

Когда шаг положительный, индексы, определяемые триплетом, представляют регулярную последовательность целых значений, начиная с первого индекса и с приращением на шаг до наибольшего целого, не превышающего второго индекса; если первый индекс больше второго - последовательность пуста.

Шаг не должен быть равным нулю.

Предположим, например, что массив объявлен как A(5,4,3). Секция A(3:5,2,1:2) - это массив конфигурации (3,2), приведенный ниже:

A(3,2,1)	A(3,2,2)
A(4,2,1)	A(4,2,2)
A(5,2,1)	A(5,2,2)

Когда шаг отрицательный, последовательность начинается с первого индекса и продолжается с приращением на шаг в сторону уменьшения индекса до такого наименьшего целого, которое больше второго индекса или равно ему; если второй индекс больше первого,- последовательность пуста. Например, если массив объявлен как B(10), секция B(9:1:-2) - это массив конфигурации (5), элементами которого являются B(9), B(7), B(5), B(3) и B(1), именно в таком порядке.

Заметим, что индекс в индексном триплете не обязательно должен находиться в объявленных для этого измерения границах, если все значения, используемые при выборе элементов массива, попадают в объявленные границы. Например, если массив объявлен как B(10), то секция массива B(3:11:7) - это массив конфигурации (2), состоящий из элементов B(3) и B(10) именно в этом порядке.

6.2.2.3.2. Векторный индекс

Векторный индекс обозначает последовательность индексов, соответствующих значениям элементов выражения. Каждый элемент выражения должен быть определен. **Многозначная секция** - это секция массива, векторный индекс которой имеет два элемента или более с одинаковым значением. Такая секция не может находиться слева от знака равенства в операторе присваивания или как элемент ввода в операторе READ.

Предположим, например, что Z - двумерный массив конфигурации (5,7), а U и V - одномерные массивы, конфигурации которых соответственно (3) и (4). Предположим, что значения U и V таковы:

```
U = (/ 1,3,2 /)
V = (/ 2,1,1,3 /)
```

Тогда секция Z(3,V) состоит из элементов третьей строки Z в следующем порядке:

```
Z(3,2)  Z(3,1)  Z(3,1)  Z(3,3)
```

Z(U,2) состоит из элементов столбца:

```
Z(1,2)  Z(3,2)  Z(2,2)
```

а Z(U,V) состоит из элементов:

```
Z(1,2)  Z(1,1)  Z(1,1)  Z(1,3)
Z(3,2)  Z(3,1)  Z(3,1)  Z(3,3)
Z(2,2)  Z(2,1)  Z(2,1)  Z(2,3)
```

Поскольку Z(3,V) и Z(U,V) содержат дублирующие элементы из Z, эти секции нельзя переопределять как секции.

Внутренний файл не должен быть секцией с векторным индексом. Секция массива с векторным индексом не должна быть аргументом, связанным с формальным массивом, который определяется или переопределяется в процедуре. Секция массива с векторным индексом не должна быть адресатом в операторе присваивания указателю.

6.3. Динамическая связь

Динамическое управление созданием, связыванием и удалением из памяти адресатов для указателей осуществляется с помощью операторов ALLOCATE, NULLIFY, DEALLOCATE и оператора присваивания указателю. ALLOCATE (6.3.1) создает адресаты для указателей; присваивание указателю связывает указатели с существующими адресатами; NULLIFY (6.3.2) разрывает связь указателей с адресатами; DEALLOCATE (6.3.3) удаляет адресаты из памяти. Динамическая связь применима к скалярам и массивам любого типа.

Операторы ALLOCATE и DEALLOCATE используются также для создания и удаления из памяти массивов с атрибутом ALLOCATABLE.

6.3.1. Оператор ALLOCATE

Оператор ALLOCATE динамически создает адресаты для указателей и динамически размещаемые массивы.

R622 *оператор-ALLOCATE*	**есть** ALLOCATE (*список-размещений* ♦ ♦ [,STAT=*переменная-состояния*])
R623 *переменная-состояния*	**есть** *скалярная-целая-переменная*
R624 *размещение*	**есть** *размещаемый-объект* ♦ ♦ [(*список-спецификаций-размещения*)]
R625 *размещаемый-объект*	**есть** *имя-переменной* **или** *компонент-структуры*
R626 *спецификация-размещения*	**есть** [*нижняя-граница-размещения* :] ♦ ♦ *верхняя-граница-размещения*
R627 *нижняя-граница-размещения*	**есть** *скалярное-целое-выражение*
R628 *верхняя-граница-размещения*	**есть** *скалярное-целое-выражение*

Ограничение: каждый *размещаемый-объект* должен быть указателем или динамически размещаемым массивом.

Ограничение: число *спецификаций-размещения* в *списке-спецификаций-размещения* должно совпадать с размерностью указателя или динамически размещаемого массива.

Граница в *спецификации-размещения* не должна быть выражением, содержащим в качестве первичного справочную функцию запроса о массиве (13.10.15), аргументом которой является любой *размещаемый-объект* из этого же оператора ALLOCATE.

Пример оператора ALLOCATE:

```
ALLOCATE (X(N), B(-3:M, 0:9), STAT = IERR_ALLOC)
```

Во время исполнения оператора ALLOCATE, выполняемого для массива, значения выражений верхней и нижней границ определяют границы массива. Последующие переопределения и разопределения каких-либо объектов, входящих в выражения границ, не влияют на значения границ массива. Если нижняя граница опущена, по умолчанию ее значение равно 1. Если верхняя граница меньше нижней, протяженность по этому измерению равна нулю и весь массив имеет нулевой размер. Заметим, что *размещаемый-объект* может быть символьного типа с нулевой длиной.

Если присутствует спецификатор STAT=, успешное выполнение оператора ALLOCATE приводит к тому, что *переменная-состояния* становится определенной и получает значение 0. Если во время выполнения оператора ALLOCATE возникла ситуация ошибки, *переменная-состояния* становится определенной и ее значение - положительное целое число, зависящее от процессора.

Если же ситуация ошибки возникла во время выполнения оператора ALLOCATE, не содержащего спецификатор STAT=, выполнение программы прекращается.

6.3.1.1. Размещение динамически размещаемых массивов

Динамически размещаемый массив, которому с помощью оператора ALLOCATE была отведена память и она затем не освобождалась (6.3.3), является **размещенным в данный момент**[1] и его можно определять. Попытка размещения массива, уже размещенного в данный момент, вызывает ситуацию ошибки в операторе ALLOCATE. В начале выполнения программы динамически размещаемые массивы имеют статус размещенности "не размещен в данный момент" и они не могут быть определены. Для того, чтобы определить, размещен ли динамически размещаемый массив в данный момент, можно использовать встроенную функцию ALLOCATED (13.13.9).

6.3.1.2. Размещение адресатов указателей

После успешного выполнения оператора ALLOCATE для указателя указатель связывается с адресатом, и его можно использовать для ссылок на адресат или для определения адресата. Размещение указателя создает объект, который неявно имеет атрибут TARGET. С помощью присваивания указателю можно связать дополнительные указатели с адресатом указателя или с частью адресата указателя. Не является ошибкой размещение указателя, который в данный момент связан с адресатом. В этом случае создается новый адресат указателя в соответствии с атрибутами указателя и границами массива, специфицированными в операторе ALLOCATE. Затем указатель связывается с новым адресатом. Любая предыдущая связь с адресатом разрывается. Если предыдущий адресат создавался путем размещения, он становится недоступным до тех пор, пока на него не будет ссылки с помощью других указателей, связанных с ним в данный момент. Чтобы проверить, связан ли в данный момент указатель, можно воспользоваться встроенной функцией ASSOCIATED (13.13.13).

В начале выполнения функции, результатом которой является указатель, статус связанности указателя-результата не определен. Перед возвратом из такой функции она должна либо связать адресат с этим указателем, либо определить статус связанности этого указателя как не связанный.

6.3.2. Оператор NULLIFY

Оператор NULLIFY прерывает связь указателей.

R629 *оператор-NULLIFY* **есть** NULLIFY (*список-объектов-указателей*)

R630 *объект-указатель* **есть** *имя-переменной*
или *компонент-структуры*

Ограничение: каждый *объект-указатель* должен иметь атрибут POINTER

6.3.3. Оператор DEALLOCATE

Оператор DEALLOCATE делает неразмещенными динамически размещаемые массивы и адресаты указателей, а указатели делает не связанными с адресатами.

R631 *оператор-DEALLOCATE* **есть** DEALLOCATE ♦
♦ (*список-размещаемых-объектов* [,STAT=*переменная-состояния*])

---

1) В дальнейшем слова "в данный момент" иногда будут опускаться.

Ограничение: каждый *размещаемый-объект* должен быть указателем или динамически размещаемым массивом.

Переменная-состояния не может быть удалена из памяти в этом же операторе DEALLOCATE.

Если спецификатор STAT= присутствует, успешное выполнение оператора DEALLOCATE приводит к тому, что *переменная-состояния* становится определенной и получает значение 0. Если во время выполнения оператора DEALLOCATE возникла ситуация ошибки, *переменная-состояния* становится определенной и ее значение - положительное целое число, зависящее от процессора. Если же ситуация ошибки возникла во время исполнения оператора DEALLOCATE без спецификатора STAT=, исполнение программы заканчивается.

Пример оператора DEALLOCATE:

```
DEALLOCATE (X,B)
```

6.3.3.1. Удаление из памяти динамически размещаемых массивов

Удаление динамически размещаемого массива, не размещенного в данный момент, вызывает ситуацию ошибки в операторе DEALLOCATE. Динамически размещаемый массив, описанный с атрибутом TARGET, нельзя удалять посредством связанного с ним указателя. Удаление динамически размещаемого массива с атрибутом TARGET делает неопределенным статус связанности любого связанного с ним указателя.

Когда выполнение процедуры заканчивается выполнением оператора RETURN или END, свой статус размещенности и определенности сохраняют следующие динамически размещаемые массивы:

1) динамически размещаемый массив с атрибутом SAVE;
2) динамически размещаемый массив из контекстной области модуля, если этот модуль доступен и другой контекстной области, находящейся в исполнении в данный момент, или
3) динамически размещаемый массив, доступный посредством связи по объемлющему контексту.

Все другие динамически размещаемые массивы, которые размещены в данный момент, при выполнени операторов RETURN и END становятся неопределенными, и статус их размещенности тоже становится неопределенным.

Если статус размещенности динамически размещаемого массива не определен, на такой массив впоследствии нельзя ссылаться, нельзя его определять, размещать или удалять из памяти.

6.3.3.2. Удаление из памяти адресатов указателей

Если в операторе DEALLOCATE имеется указатель, его статус связанности должен быть определен. Попытка удалить из памяти указатель, не связанный с адресатом или связанный с адресатом, который не был создан оператором ALLOCATE, вызывает ситуацию ошибки в операторе DEALLOCATE. Если указатель связан в данный момент с динамически размещаемым массивом, указатель нельзя удалять.

Указатель, который не связан в данный момент с полным размещенным объектом-адресатом, нельзя удалять. Если указатель в данный момент связан с частью (2.4.3.1) объекта-адресата, независимой от любой другой его части, его нельзя удалять из памяти. Удаление из памяти адресата указателя делает неопределенным статус связанности всех других указателей, связанных с этим адресатом или его частью.

Когда выполнение процедуры заканчивается исполнением операторов RETURN или END, статус связанности указателя, объявленного или доступного в этой процедуре, становится неопределенным, кроме следующих случаев:

1) указатель с атрибутом SAVE;
2) указатель из неименованного общего блока;
3) указатель из именованного общего блока, появляющийся хотя бы в одной из других контекстных областей, находящихся в исполнении в данный момент;
4) указатель, объявленный в контекстной области модуля, если этот модуль доступен и другой контекстной области, находящейся в исполнении в данный момент;
5) указатель, доступный через связь по объемлющему контексту;
6) указатель, выдающий значение функции, объявленной с атрибутом POINTER.

Когда адресат указателя становится неопределенным при выполнении оператора END или RETURN, статус связанности указателя (14.6.2.1) становится неопределенным.

Глава 7. ВЫРАЖЕНИЯ И ПРИСВАИВАНИЕ

Эта глава описывает выражения, правила их образования, интерпретации и вычисления и оператор присваивания.

7.1.Выражения

Выражение представляет либо ссылку на данное, либо вычисление, и его значением является скаляр или массив. Образуется выражение из операндов, операций и скобок. Простейшими формами операндов являются константы или переменные, такие как

```
3.0
.FALSE.
A
B(I)
C(I:J)
```

Операнд - это скаляр или массив. Операция - либо встроенная (7.2), либо определяемая (7.3). Более сложные выражения можно образовывать из операндов, которые сами являются выражениями.

Примеры встроенных операций:

```
+
>
.AND.
```

7.1.1. Форма выражения

В результате вычисления выражения вырабатывается значение, которое имеет тип, параметры типа (если нужно) и конфигурацию (7.1.4).

Примеры выражений:

```
A + B
(A-B) * C
A ** B
C .AND. D
F // G
```

Выражение определяется в терминах следующих категорий: первичное, выражение-1-уровня, выражение-2-уровня, выражение-3-уровня, выражение-4-уровня и выражение-5-уровня.

Эти категории относятся к разным уровням предшествования операций и в общем случае определяются с помощью других категорий. Простейшей формой каждой категории выражения является *первичное*. Приведенные ниже правила определяют синтаксис выражения. Семантика описана в 7.2 и 7.3.

7.1.1.1. Первичное

R701 *первичное* **есть** *константа*
или *константа-подобъект*
или *переменная*
или *конструктор-массива*
или *конструктор-структуры*
или *ссылка-на-функцию*
или *(выражение)*

R702 *константа-подобъект* **есть** *подобъект*

Ограничение: *подобъект* должен быть обозначением подобъекта, основой которого является константа.

Ограничение: *переменная* в качестве *первичного* не должна быть массивом неопределенного размера.

Примеры *первичного*:

Пример	Синтаксический класс
1.0	*константа*
'ABCDEFGHIJKLMNOPQRSTUVXYZ'(I:I)	*константа-подобъект*
A	*переменная*
(/1.0,2.0/)	*конструктор-массива*
PERSON (12, 'Jones')	*конструктор-структуры*
F(X,Y)	*ссылка-на-функцию*
(S+T)	*(выражение)*

7.1.1.2. Выражения уровня-1

Определяемые унарные операции имеют самый высокий приоритет среди операций (см. табл. 7.7). Выражение уровня-1 - это первичные, возможно, со знаками определяемых унарных операций.

R703 *выражение-1-уровня* **есть** ♦
♦ [*знак-определяемой-унарной-операции*] *первичное*

R704 *знак-определяемой-унарной-операции* **есть** . *буква*[*буква*]... .

Ограничение: *знак-определяемой-унарной-операции* должен содержать не более 31 буквы и не должен совпадать с каким-либо из *знаков-встроенных-операций* или с одной из *логических-литеральных-констант*.

Простые примеры выражения уровня-1:

Пример	Синтаксический класс
A	*первичное* (R701)
.INVERSE. B	*выражение-1-уровня* (R703)

Более сложный пример выражения уровня-1:

```
.INVERSE. (A+B)
```

7.1.1.3. Выражения уровня-2

Выражения уровня-2 - это выражения уровня-1, возможно, соединенные знаками числовых операций: *оп-степени*, *оп-умножения* и *оп-сложения*.

R705 *множитель* **есть** *выражение-1-уровня* [*оп-степени множитель*]

R706 *слагаемое* **есть** [*слагаемое оп-умножения*] *множитель*

R707 *выражение-2-уровня* **есть** [[*выражение-2-уровня*] *оп-сложения*] *слагаемое*

R708 *оп-степени* **есть** **

R709 *оп-умножения* **есть** *
или /

R710 *оп-сложения* **есть** +
или -

Простые примеры выражений уровня-2:

Пример	Синтаксический класс	Примечание
A	*первичное*	A - *первичное* (R703)
B**C	*множитель*	B - *выражение-1-уровня*, ** - *оп-степени* и C - *множитель* (R705)
D*E	*слагаемое*	D - *слагаемое*, * - *оп-умножения* и E - *множитель* (R706)
+1	*выражение-2-уровня*	+ - *оп-сложения* и 1 - *слагаемое*(R707)

F-I *выражение-2-уровня* F - *выражение-2-уровня*,
- обозначает *оп-сложения*
и I - *слагаемое* (R707)

Более сложный пример выражения уровня-2:

```
-A + D * E + B ** C
```

7.1.1.4. Выражения уровня-3

Выражения уровня-3 - это выражения уровня-2, возможно, включающие символьную операцию *оп-конкатенации*.

R711 *выражение-3-уровня* **есть** [*выражение-3-уровня* ♦
♦ *оп-конкатенации*] *выражение-2-уровня*

R712 *оп-конкатенации* **есть** //

Простые примеры выражения уровня-3:

Пример	Синтаксический класс
A	*выражение-2-уровня* (R707)
B // C	*выражение-3-уровня* (R711)

Более сложный пример выражения уровня-3:

```
X // Y // 'ABCD'
```

7.1.1.5. Выражения уровня-4

Выражения уровня-4 - это выражения уровня-3, возможно, включающие операции отношения *оп-отношения*.

R713 *выражение-4-уровня* **есть** [*выражение-3-уровня* *оп-отношения*] ♦
♦ *выражение-3-уровня*

R714 *оп-отношения* **есть** .EQ. **или** ==
или .NE. **или** /=
или .LT. **или** <
или .LE. **или** <=
или .GT. **или** >
или .GE. **или** >=

Простые примеры выражения уровня-4:

Пример	Синтаксический класс
A	*выражение-3-уровня* (R711)
B .EQ. C	*выражение-4-уровня* (R713)
D < E	*выражение-4-уровня* (R713)

Более сложный пример выражения уровня-4:

```
(A+B) .NE. C
```

7.1.1.6. Выражения уровня-5

Выражения уровня-5 - это выражения уровня-4, возможно, включающие логические операции: *оп-отрицания*, *оп-конъюнкции*, *оп-дизъюнкции*, *оп-эквивалентности*.

R715 *логический-множитель* **есть** [*оп-отрицания*] *выражение-4-уровня*

R716 *логическое-слагаемое* **есть** [*логическое-слагаемое* ♦
♦ *оп-конъюнкции*] *логический-множитель*

R717 *операнд-эквивалентности* **есть** [*операнд-эквивалентности* ♦
♦ *оп-дизъюнкции*] *логическое-слагаемое*

R718 *выражение-5-уровня* **есть** [*выражение-5-уровня* ♦
♦ *оп-эквивалентности*] *операнд-эквивалентности*

R719 *оп-отрицания* **есть** .NOT.

R720 *оп-конъюнкции* **есть** .AND.

R721 *оп-дизъюнкции* **есть** .OR.

R722 *оп-эквивалентности* **есть** .EQV.
или .NEQV.

Простые примеры выражения уровня-5:

Пример	Синтаксический класс
A	*выражение-4-уровня* (R713)
.NOT. B	*логический-множитель* (R715)
C.AND.D	*логическое-слагаемое* (R716)
E.OR.F	*операнд-эквивалентности* (R717)
G.EQV.H	*выражение-5-уровня* (R718)
S.NEQV.T	*выражение-5-уровня* (R718)

Более сложный пример выражения уровня-5:

```
A .AND. B .EQV. .NOT. C
```

7.1.1.7. Общая форма выражения

Выражения есть выражения уровня-5, возможно, включающие знаки определяемых бинарных операций. Знаки определяемых бинарных операций имеют самый низкий приоритет (табл. 7.7).

R723 *выражение* **есть** [*выражение* ♦ ♦ *знак-определяемой-бинарной-операции*] *выражение-5-уровня*

R724 *знак-определяемой-бинарной-операции* **есть** . *буква*[*буква*]... .

Ограничение: *знак-определяемой-бинарной-операции* должен содержать не более 31 буквы и не должен совпадать с каким-либо из *знаков-встроенной-операции* или с одной из *логических-литеральных-констант*.

Простые примеры выражений:

Пример	Синтаксический класс
A	*выражение-5-уровня* (R718)
B .UNION. C	*выражение* (R723)

Более сложные примеры выражений:

```
(B .INTERSECT. C) .UNION. (X-Y)
A + B .EQ. C * D
.INVERSE. (A + B)
A + B .AND. C * D
E // G .EQ. H (1:10)
```

7.1.2. Встроенные операции

Встроенная операция - это либо встроенная унарная, либо встроенная бинарная операция. **Встроенная унарная операция** - это операция вида: *знак-встроенной-операции* x_1 где x_2 имеет один из встроенных типов (4.3), перечисленных в табл. 7.1 для этой встроенной унарной операции.

Встроенная бинарная операция - это операция вида: x_1 *знак-встроенной-операции* x_2, где x_1 и x_2 имеют встроенные типы (4.3), перечисленные в табл. 7.1 для этой встроенной операции, и согласованы по конфигурации (7.1.5).

Таблица 7.1

Типы операндов и результатов для встроенной операции $[x_1]$ *оп* x_2

Знак встроенной операции	Тип x_1	Тип x_2	Тип $[x_1]$ *оп* x_2
унарный + , -		I, R, Z	I, R, Z
бинарный +, - , * , / , **	I R Z	I, R, Z I, R, Z I, R, Z	I, R, Z R R, Z Z, Z, Z
//	C	C	C
.EQ., .NE. == , /=	I R Z C	I, R, Z I, R, Z I, R, Z C	L, L, L L, L, L L, L, L L
.GT., .GE., .LT., .LE. > , >= , < , <=	I R C	I, R I, R C	L, L L, L L
.NOT.		L	L
.AND., .OR., .EQV., .NEQV.	L	L	L

Примечание . Символы I, R, Z, C и L обозначают целый, вещественный, комплексный, символьный и логический типы. Там, где для x_2 указано несколько типов, тип результата берется в той же позиции следующего столбца. Для встроенных операций, требующих операнды символьного типа, параметры вида у операндов должны быть одинаковыми.

Числовая встроенная операция - это встроенная операция, содержащая один из знаков числовой операции (+, -, *, /, **). **Знак числовой встроенной операции** - это знак операции в числовой встроенной операции.

Для числовых встроенных бинарных операций два операнда могут иметь разные числовые типы или разные параметры вида. Кроме случая возведения в целую степень, если типы или параметры типа операндов не совпадают, эффект такой, как если бы каждый операнд, тип или параметр типа которого отличается от результата, до выполнения операции был преобразован к типу и параметру типа результата. Когда вещественное или комплексное значение возводится в целую степень, целый операнд не преобразуется.

Символьная встроенная операция, встроенная операция отношения и **логическая встроенная операция** определяются соответственно с помощью **знака символьной встроенной операции** (//), **знака встроенной операции отношения** (.EQ., .NE., GT., GE., LT., LE., ==, /=, >, >=, <, <=) и **знака логической встроенной операции** (.AND., .OR., .NOT., EQV. и .NEQV.). Для встроенной операции // параметры вида должны быть одинаковыми.

Числовая встроенная операция отношения - это встроенная операция отношения над операндами числового типа. **Символьная встроенная операция отношения** - это встроенная операция отношения над операндами символьного типа с одинаковыми значениями параметра вида.

7.1.3. Определяемые операции

Определяемая операция - это определяемая унарная операция или определяемая бинарная операция. **Определяемая унарная операция** - это

операция вида *знак-определяемой-унарной-операции* x_2, которая определяется с помощью функции и блока обобщенного интерфейса (12.3.1), или операция вида *знак-встроенной-операции* x_2, где тип x_2 - не тот, что требуется для унарной встроенной операции (7.1.2), и операция определяется с помощью функции и блока обобщенного интерфейса.

Определяемая бинарная операция - это либо операция вида x_1 *знак-определяемой-бинарной-операции* x_2, определенная с помощью функции и блока обобщенного интерфейса, либо операция вида x_1 *знак-встроенной-операции* x_2, где типы и размерности x_1 или x_2, либо их обоих, не совпадают с тем, что требуется для встроенной бинарной операции (7.1.2), и эта операция определяется с помощью функции и блока обобщенного интерфейса.

Заметим, что знак встроенной операции можно использовать как знак операции в определяемой операции. В этом случае родовые свойства знака операции расширены.

Операция расширения - это определяемая операция, в которой знак операции имеет вид *знака-определяемой-унарной-операции* или *знака-определяемой-бинарной-операции*. Такой знак операции называется **знаком операции расширения**. Знак операции, используемый в операции расширения, может быть таким, что обобщенный интерфейс для него может быть специфицирован с помощью нескольких функций.

7.1.4. Тип, параметры типа и конфигурация выражения

Тип и конфигурация выражения зависят от знаков операций и типа и конфигурации используемых в выражении первичных. Они определяются рекурсивно из синтаксической формы выражения. Тип для выражения - это либо один из встроенных типов (4.3), либо производный тип (4.4).

R725 *логическое-выражение* **есть** *выражение*

Ограничение: *логическое-выражение* должно быть логического типа.

R726 *символьное-выражение* **есть** *выражение*

Ограничение: *символьное-выражение* должно быть символьного типа.

R727 *выражение-станд-симв-типа* **есть** *выражение*

Ограничение: *выражение-станд-симв-типа* должно быть выражением стандартного символьного типа.

R728 *целое-выражение* **есть** *выражение*

Ограничение: *целое-выражение* должно быть целого типа.

R729 *числовое-выражение* **есть** *выражение*

Ограничение: *числовое-выражение* должно быть целого, вещественного или комплексного типа.

Выражение встроенного типа имеет параметр вида. Кроме того, выражение символьного типа имеет параметр длины. Параметры типа для выражения определяются видом выражения.

7.1.4.1. Тип, параметры типа и конфигурация первичного

Тип, параметры типа и конфигурация первичного определяются в соответствии с тем, является ли первичное константой, переменной, конструктором массива, конструктором структуры, ссылкой на функцию или выражением в скобках. Если первичное - константа, его тип, параметры типа и конфигурация определяются константой. Если это конструктор структуры, тогда это скаляр и его тип определяется по имени конструктора. Если это конструктор массива, его тип, параметры типа и конфигурация такие, как описано в 4.5. Если первичное - переменная или ссылка на функцию, его тип, параметры типа и конфигурация определяются из соответствующих атрибутов переменной (5.1.1, 5.1.2) или ссылки на функцию (12.4.2), соответственно. Заметим, что в случае ссылки на функцию, функция может быть обобщенной (12.3.2.1, 13.10), и тогда ее тип, параметры типа и конфигурация определяются типом, пара-

метрами типа и размерностями ее фактических аргументов. Если первичное - выражение в скобках, его тип, параметры типа и конфигурация такие же, как у выражения.

Если указатель появляется в качестве первичного во встроенной операции или в определяемой операции, в которой он соответствует формальному аргументу - не указателю, ссылка указывает на связанный с ним объект-адресат. Тип, параметры типа и конфигурация первичного те же, что у соответствующего адресата. Если же указатель не связан с адресатом, он может появиться в качестве первичного только как фактический аргумент в ссылке на процедуру, соответствующий формальный аргумент которой объявлен как указатель.

7.1.4.2. Тип, параметры типа и конфигурация результата операции

Тип результата встроенной операции $[x_1]$ *оп* x_2 указан в табл.7.1. Тип результата определяемой операции $[x_1]$ *оп* x_2 определяется функцией, описывающей операцию (7.3).

Конфигурацией результата встроенной операции $[x_1]$ *оп* x_2 является либо конфигурация x_2, если *оп* - унарная операция,или x_1 - скаляр, либо конфигурация x_1 - в противном случае.

Выражение встроенного типа имеет параметр вида. Символьное выражение имеет также параметр длины. Для выражения x_1 // x_2, где x_1 и x_2 - символьного типа, параметр длины - сумма длин операндов, а параметр вида - тот же, что у x_1, который должен совпадать с параметром вида x_2. Для выражения *оп* x_2, где *оп* - знак встроенной унарной операции, а x_2 имеет тип целый, вещественный или комплексный, параметр вида тот же, что у операнда. Для выражения x_1 *оп* x_2, где *оп* - знак числовой встроенной бинарной операции, один операнд целого типа, а другой - вещественного или комплексного типа, параметр вида такой же, как у вещественного или комплексного операнда. В случае, когда оба операнда имеют одинаковый тип и одинаковые параметры вида, либо один вещественный и один комплексный операнды с одинаковыми параметрами вида, параметр вида для выражения идентичен параметру вида каждого из операндов. В случае, когда оба операнда - целые, но с разными параметрами вида, параметр вида для выражения такой, как у операнда с большим диапазоном десятичного порядка, или зависит от процессора, если операнды имеют одинаковый диапазон десятичного порядка. Когда операнды вещественные или комплексные с разными параметрами вида, параметр вида для выражения такой, как у операнда, десятичная точность которого больше или зависит от процессора, если операнды имеют одинаковую десятичную точность. Для выражения x_1 *оп* x_2, где *оп* - встроенная бинарная логическая операция над двумя операндами с одинаковыми параметрами вида, параметр вида для выражения такой же, как у операндов. Если же оба операнда логического типа с разными параметрами вида, параметр вида для выражения зависит от процессора. Для выражения x_1 *оп* x_2, где *оп* - встроенная операция отношения, выражение имеет параметр вида,как для стандартного логического типа.

7.1.5. Правила согласования для встроенных операций

Два объекта **согласованы по конфигурации**, если либо оба они - массивы одинаковой конфигурации, либо если оба они - скаляры, либо один из них - массив, а другой - скаляр.

Для всех встроенных бинарных операций оба операнда должны быть согласованы по конфигурации. В случае, если один - скаляр, а другой - массив, скаляр воспринимается как если бы он был массивом той же конфигурации, что операнд-массив, и каждый элемент этого массива был бы равен значению скаляра.

7.1.6. Выражения-скаляры и выражения-массивы

Выражение может быть либо выражением-скаляром, либо выражением-массивом.

Пример скалярного выражения:

```
Q + 2.3 * R
```

где Q и R - скаляры.

Пример выражения-массива:

```
A(1:10) + B(2:11)
```

где A и B - массивы.

7.1.6.1. Константное выражение

Константное выражение - это выражение, в котором каждая операция - встроенная, а каждое первичное в таком выражении должно быть одной из следующих форм:

1) константа или подобъект константы, где все индексы, индексы секции, начальные и конечные точки подстроки являются константными выражениями;
2) конструктор массива, где каждый элемент, границы и шаг каждого из неявных циклов являются выражениями, в которых в качестве первичных используются только константные выражения и управляющие переменные неявных циклов;
3) конструктор структуры, где каждый компонент - константное выражение;
4) ссылка на поэлементную встроенную функцию, где каждый аргумент - константное выражение;
5) ссылка на встроенную функцию преобразования, где каждый аргумент - константное выражение;
6) ссылка на любую справочную функцию для массива (13.10.15), кроме ALLOCATED, битовую справочную функцию BIT_SIZE, символьную справочную функцию LEN, справочную функцию KIND для запроса о виде аргумента или числовую справочную функцию (13.10.8), где каждый аргумент - либо константное выражение, либо переменная, у которой параметры типа или границы, по поводу которых производится запрос, не являются неопределенными или определяемыми с помощью оператора ALLOCATE или оператора присваивания указателю;
7) константное выражение, заключенное в скобки.

Символьное константное выражение - это константное выражение символьного типа. **Целое константное выражение** - это константное выражение целого типа. **Логическое константное выражение** - это константное выражение логического типа. **Числовое константное выражение** - это константное выражение целого, вещественного или комплексного типа.

Выражение инициализации - это константное выражение, в котором возведение в степень разрешено только для целой степени, и каждое первичное в таком выражении должно быть одной из следующих форм:

1) константа или подобъект константы, где все индексы, индексы секций, начальные и конечные точки подстроки являются выражениями инициализации;
2) конструктор массива, где все элементы, границы и шаг каждого из неявных циклов являются выражениями, содержащими в качестве первичных либо выражения инициализации, либо управляющие переменные неявных циклов;
3) конструктор структуры, где каждый компонент - выражение инициализации;
4) ссылка на поэлементную встроенную функцию целого или символьного типа, где каждый аргумент является выражением инициализации целого или символьного типа;
5) ссылка на одну из функций преобразования, REPEAT, RESHAPE, SELECTED_INT_KIND, SELECTED_REAL_KIND, TRANSFER или TRIM, где каждый аргумент - выражение инициализации;

6) ссылка на любую справочную функцию для массива (13.10.15), кроме ALLOCATED, битовую справочную функцию BIT_SIZE, символьную справочную функцию LEN, справочную функцию KIND для запроса о виде аргумента или числовую справочную функцию (13.10.8), где каждый аргумент - либо выражение инициализации, либо переменная, у которой параметры типа или границы, по поводу которых производится запрос, не являются неопределенными или определяемыми при помощи оператора ALLOCATE или оператора присваивания указателю;

7) выражение инициализации, заключенное в скобки.

R730 *выражение-инициализации* **есть** *выражение*

Ограничение: *выражение-инициализации* должно быть выражением инициализации.

R731 *символьное-выражение-инициализации* **есть** *символьное-выражение*

Ограничение: *символьное-выражение-инициализации* должно быть выражением инициализации.

R732 *целое-выражение-инициализации* **есть** *целое-выражение*

Ограничение: *целое-выражение-инициализации* должно быть выражением инициализации.

R733 *логическое-выражение-инициализации* **есть** *логическое-выражение*

Ограничение: *логическое-выражение-инициализации* должно быть выражением инициализации.

Если выражение инициализации включает ссылку на справочную функцию для запроса о параметре типа или границе объекта, описанного в том же *разделе-спецификаций*, эти параметр типа или граница массива должны быть специфицированы раньше в этом же *разделе-спецификаций*. Эта предварительная спецификация может находиться слева от справочной функции в том же операторе.

Примеры константных выражений:

```
 3
-3+4
'AB'
'AB' // 'CD'
('AB' // 'CD') // 'EF'
SIZE (A)
DIGITS (X) + 4
```

где A - массив явной конфигурации, а X имеет стандартный вещественный тип.

Примеры константных выражений, не являющихся выражениями инициализации:

```
ABS (9.0)                          ! Не целый аргумент
3.0 ** 2.0                         ! Не целая степень
DOT_PRODUCT ((/2,3/),(/1,7/))      ! Недопустимая функция
```

7.1.6.2. Выражение спецификации

Ограниченное выражение - это выражение, в котором каждая операция - встроенная, а каждое первичное должно быть одной из следующих форм:

1) константа или подобъект константы;

2) переменная - формальный аргумент без атрибута OPTIONAL или INTENT(OUT) либо переменная, которая является подобъектом такого формального аргумента;

3) переменная или подобъект переменной из общего блока;

4) переменная, доступная через связь по использованию или по объемлющему контексту, либо подобъект такой переменной;

5) конструктор массива, в котором все элементы, границы и шаг каждого из неявных циклов - выражения, в которых в качестве первичного используются только ограниченные выражения и управляющие переменные неявных циклов;

6) конструктор структуры, в котором каждый компонент - ограниченное выражение;

7) ссылка на поэлементную встроенную функцию целого или символьного типа, где каждый аргумент является ограниченным выражением целого или символьного типа;

8) ссылка на одну из функций преобразования, REPEAT, RESHAPE, SELECTED_INT_KIND, SELECTED_REAL_KIND, TRANSFER или TRIM, где каждый аргумент - ограниченное выражение целого или символьного типа;

9) ссылка на любую справочную функцию для массива (13.10.15), кроме ALLOCATED, битовую справочную функцию BIT_SIZE, символьную справочную функцию LEN, справочную функцию KIND для запроса о виде аргумента или числовую справочную функцию (13.10.8), где каждый аргумент - либо ограниченное выражение, либо переменная, у которой параметры типа или границы, по поводу которых производится запрос, не являются неопределенными или определяемыми при помощи оператора ALLOCATE или оператора присваивания указателю или

10) ограниченное выражение, заключенное в скобки,

и в котором все индексы, индексы секции, начальные и конечные точки подстроки являются ограниченными выражениями.

Выражение спецификации (R509, R514, R515) - это ограниченное выражение, которое является скаляром целого типа.

R734 *выражение-спецификации* **есть** *скалярное-целое-выражение*

Ограничение: *скалярное-целое-выражение* должно быть ограниченным выражением.

Тип и параметры типа (если они есть) переменных в выражении спецификации должны быть определены либо с помощью предварительного описания в той же контекстной области, либо по действующим в ней в данный момент правилам неявной типизации, либо через связь по использованию или по объемлющему контексту. Если тип переменной в выражении спецификации определен по правилам неявной типизации, любое последующее ее вхождение в операторы объявления типа должно соответствовать неявно заданному типу и параметрам типа.

Если выражение спецификации включает ссылку на справочную функцию для запроса о параметрах типа или границах объекта, специфицированного в том же *разделе-спецификаций*, параметр типа или граница массива должны быть объявлены до появления выражения спецификации в этом же *разделе-спецификаций*. Если выражение спецификации включает ссылку на значение элемента массива, специфицированного в том же *разделе-спецификаций*, границы массива должны быть специфицированы в предварительном объявлении. Предварительное объявление может находиться в том же операторе слева от справочной функции.

Примеры выражений спецификации:

```
LBOUND (B,1) + 5          ! B - формальный массив
                          ! неопределенной конфигурации
M + LEN(C)                ! M и C - формальные аргументы
2 * PRECISION (A)         ! A - вещественная переменная,
                          ! доступная через оператор USE
```

7.1.7. Выполнение операций

Этот раздел относится и к встроенным, и к определяемым операциям.

Все переменные или ссылки на функции, используемые в качестве операндов в выражениях, должны быть определены к моменту выполнения ссылки. Если операнд - указатель, он должен быть связан с адресатом, который уже определен. Целый операнд должен иметь целое значение, а не значение метки оператора. Все символы в ссылке на символьный объект данных должны быть определены.

При ссылке на массив или секцию массива должны быть определены все выбираемые элементы. При ссылке на структуру должны быть определены все ее компоненты.

При выполнении программы запрещены любые численные операции, результат которых математически не определен, например, деление на нуль или возведение нулевого первичного в нулевую или отрицательную степень. Запрещено также возведение отрицательного вещественного первичного в вещественную степень.

Выполнение ссылки на функцию не должно зависеть от вычисления любого другого объекта в операторе и не должно влиять на него. Однако выполнение ссылки на функцию в логическом выражении оператора IF (8.1.2.4) или оператора WHERE (7.5.3.1) может определять переменные в том операторе, который выполняется, если значение выражения - "истина". Например, в операторах

```
IF (F(X)) A = X
WHERE (G(X)) B = X
```

F или G могут определять X. Если ссылка на функцию вызывает определение или переопределение фактического аргумента этой функции, то этот аргумент или любой другой связанный с ним объект не должны появляться где-либо в этом же операторе. Например, операторы

```
A(I) = F(I)
Y = G(X) + X
```

запрещены, если ссылка на F определяет или разопределяет (приводит к неопределенности значения) I или ссылка на G определяет или разопределяет X.

Тип выражения, в котором есть ссылка на функцию, не влияет на вычисление фактических аргументов этой функции.

Выполнение ссылки на элемент массива требует вычисления его индексов. Тип выражения, в котором находятся индексы, не влияет на вычисление индекса и не зависит от него. Выполнение ссылки на секцию массива требует вычисления индексов секции. Тип выражения, в котором находится секция массива, не зависит от вычисления индексов секции массива и не влияет на него. Выполнение ссылки на подстроку требует вычисления выражений подстроки. Тип выражения, в котором находится подстрока, не влияет на вычисление выражений подстроки и не зависит от них. Необязательно, чтобы процессор вычислял все индексные выражения и выражения подстроки массива нулевого размера или символьного объекта нулевой длины.

Появление конструктора массива требует вычисления границ и шага для каждого из неявных циклов, содержащихся в нем. Тип выражения, в котором находится конструктор массива, не зависит от вычисления выражений границ и шага и не влияет на него.

Когда встроенная бинарная операция применяется к скаляру и массиву или к двум массивам одинаковой конфигурации, операция выполняется поэлементно над соответствующими элементами массива-операнда. Например, выражение-массив

```
A + B
```

вырабатывает массив той же конфигурации, что A и B. Отдельные элементы результата имеют значение, равное, соответственно, первому элементу A, сложенному с первым элементом B, второму элементу A, сложенному со вторым элементом B и т.д. Процессор может выполнять поэлементные операции в любом порядке.

Когда встроенная унарная операция выполняется над одним массивом-операндом, операция производится поэлементно, в любом порядке. Результатом является массив той же конфигурации, что и операнд.

7.1.7.1. Вычисление операндов

Процессору не обязательно вычислять все операнды выражения или полностью каждый операнд, если значение выражения можно определить без этого. Чаще всего этот принцип применяется к выражениям логического типа, к массивам нулевого размера и к строкам нулевой длины, но он применим и ко всем выражениям. Например, при вычислении выражения

```
X .GT. Y .OR. L(Z)
```

где X, Y, Z - вещественные, L - функция логического типа, не обязательно вычислять ссылку L(Z), если X больше, чем Y.

Аналогично в выражении-массиве

```
X + W(Z)
```

где X - нулевого размера, а W - функция, ссылку на функцию W(Z) вычислять не обязательно. Если оператор содержит ссылку на функцию в той части выражения, которую можно не вычислять, все объекты, которые должны были стать определенными при выполнении этой ссылки, становятся неопределенными по завершении вычисления выражения, содержащего эту ссылку. В примерах, приведенных выше, вычисление выражений делает Z неопределенным, если L или W определяют свои аргументы.

7.1.7.2. Целостность скобок

Последующие разделы устанавливают определенные условия, при которых процессор может вычислять выражение не так, как этого требуют правила, приведенные в 7.1.1, 7.2 и 7.3. Однако любое выражение, заключенное в скобки, должно рассматриваться как единый объект. Например, при вычислении выражения A+(B-C), где A, B и C - числового типа, разность B и C должна быть вычислена до выполнения операции сложения; процессор не должен вычислять математически эквивалентное выражение (A+B)-C.

7.1.7.3. Выполнение числовых встроенных операций

Правила, приведенные в 7.2.1, описывают интерпретацию числовых встроенных операций. Установив однажды интерпретацию в соответствии с этими правилами, процессор может вычислять любое математически эквивалентное выражение при условии, что целостность скобок не нарушается.

Два выражения числового типа математически эквивалентны, если для всех возможных значений первичных их математические значения равны. Однако математически эквивалентные выражения числового типа могут вырабатывать разные численные результаты. Например, разница между значениями выражений (1./3.)*3 и 1.- это вычислительная разница, но не математическая.

Математическое определение целого деления дано в 7.2.1.1. Разница между значениями выражений 5/2 и 5./2. - это математическая, а не вычислительная разница.

Ниже приведены примеры выражений вместе с допустимыми альтернативными формами, которые могут быть использованы процессором при вычислении этих выражений. A, B и C - произвольные вещественные или комплексные операнды, I и J произвольные целые операнды, а X, Y и Z - произвольные числовые операнды.

Выражение	Допустимая альтернативная форма
X+Y	Y+X
X*Y	Y*X
-X+Y	Y-X
X+Z+Y	X+(Y+Z)
X-Y+Z	X-(Y-Z)
X*A/Z	X*(A/Z)
X*Y-X*Z	X*(Y-Z)
A/B/C	A/(B*C)
A/5.0	0.2*A

Далее следуют примеры выражений с запрещенными альтернативными формами, которые не должны использоваться процессором при вычислении этих выражений.

Выражение	Недопустимая альтернативная форма
I/2	0.5*I
X*I/J	X*(I/J)
(X+Y)+Z	(X+Y+Z)
I/J/A	I/(J*A)
(X+Y)+Z	X+(Y+Z)
(X*Y)-(X*Z)	X*(Y-Z)
X*(Y-Z)	X*Y-X*Z

Помимо скобок, необходимых для установления требуемой интерпретации, скобки могут быть включены для ограничения альтернативных форм, которые могут быть использованы процессором при фактическом вычислении выражения. Это полезно для контроля за величиной и точностью промежуточных значений, возникающих при вычислении выражения. Например, в выражении

```
A + (B - C)
```

выражение (B-C) должно быть вычислено, а потом сложено с A. Заметим, что включение скобок может изменить математическое значение выражения. Например, два выражения

```
A * I / J
A * (I / J)
```

могут иметь различные математические значения, если I и J целого типа.

Каждый операнд числовой встроенной операции имеет тип, который может зависеть от порядка вычисления, используемого процессором. Например, при вычислении выражения

```
Z + R + I
```

где Z, R и I представляют объекты соответственно комплексного, вещественного и целого типов, тип операнда, который складывается с I, может быть либо комплексным, либо вещественным, в зависимости от того, какая пара операндов (Z и R, R и I или Z и I) складывается первой.

7.1.7.4. Выполнение встроенной символьной операции

Правила, приведенные в 7.2.2, описывают интерпретацию символьной встроенной операции. Процессору надо вычислять встроенную символьную операцию ровно настолько, насколько необходимо для контекста, в котором появляется выражение. Например, в операторах

```
CHARACTER (LEN=2) C1, C2, C3, CF
C1 = C2 // CF (C3)
```

нет необходимости вычислять функцию CF, т.к. для определения значения C1 достаточно значения C2, поскольку C1 имеет длину 2.

7.1.7.5. Выполнение встроенных операций отношения

Правила, приведенные в 7.2.3, описывают интерпретацию встроенных операций отношения. Однажды установив интерпретацию в соответствии с этими правилами, процессор может вычислять любое другое выражение, которое эквивалентно данному, при условии, что не будет нарушена целостность скобок в каком-либо из выражений. Например, процессор может выбрать для вычисления выражения

```
I .GT. J
```

где I и J - целые переменные, выражение

```
J - I .LT. 0
```

Две встроенные операции отношения эквивалентны, если их логические значения равны для всех возможных значений их первичных.

7.1.7.6. Выполнение логических встроенных операций

Правила, приведенные в 7.2.4, описывают интерпретацию логических встроенных операций. Однажды установив интерпретацию выражения в соответствии с этими правилами, процессор может вычислять любое другое выражение, которое логически эквивалентно данному, при условии сохранения целостности скобок во всех выражениях. Например, для переменных логического типа L1, L2 и L3 процессор может вычислять выражение

```
L1 .AND. L2 .AND. L3
```

как

```
L1 .AND. (L2 .AND. L3)
```

Два выражения логического типа эквивалентны, если их значения равны для всех возможных значений их первичных.

7.1.7.7. Выполнение определяемых операций

Правила, приведенные в 7.3, описывают интерпретацию определяемых операций. Однажды установив интерпретацию выражения в соответствии с этими правилами, процессор может вычислять любое другое выражение, которое эквивалентно данному, при условии сохранения целостности скобок.

Два выражения производного типа эквивалентны, если их значения равны при всех возможных значениях их первичных.

7.2. Интерпретация встроенных операций

Встроенные операции определены в 7.1.2. Эти операции разделяются на следующие категории: числовые, символьные, отношения и логические. Интерпретации, определенные ниже, применяются как к скалярам, так и к массивам; для массивов применяется поэлементно скалярная интерпретация.

Тип, параметры типа и интерпретация выражения, которое состоит из встроенной унарной или бинарной операции, не зависят от контекста, в котором находится выражение. В частности, тип, параметры типа и интерпретация такого выражения не зависят от типа и параметров типа любого другого, большего выражения, в котором находится данное выражение. Например, если X вещественное, J - целое, а INT - встроенная функция преобразования из вещественного в целое, выражение INT(X+J) - целое выражение, а X+J - вещественное.

7.2.1. Числовые встроенные операции

Числовая операция используется для того, чтобы выразить численные манипуляции. При выполнении числовой операции вырабатывается числовое значение.

Допустимые типы данных для операндов числовых встроенных операций описаны в 7.1.2.

Числовые операции и их интерпретация в выражении приведены в табл. 7.2, где x_1 означает операнд слева от знака операции, а x_2 - справа.

Таблица 7.2

Интерпретация числовых встроенных операций

Знак операции	Название операции	Использование знака операции	Интерпретация
**	Возведение в степень	x_1 ** x_2	Возвести x_1 в степень x_2
/	Деление	x_1 / x_2	Разделить x_1 на x_2
*	Умножение	x_1 * x_2	Умножить x_1 на x_2
-	Вычитание	x_1 - x_2	Вычесть x_2 из x_1
-	Изменение знака	- x_2	Минус x_2
+	Сложение	x_1 + x_2	Сложить x_1 с x_2
+	Идентичность	+ x_2	То же, что x_2

Интерпретация деления зависит от типов данных для операндов (7.2.1.1).

Если x_1 и x_2 - целого типа и x_2 имеет отрицательное значение, интерпретация x_1**x_2 такая же, как у выражения 1/(x_1**ABS(x_2)), которое подчиняется правилам целочисленного деления (7.2.1.1). Например, 2**(-3) имеет значение 1/(2**3), которое равно нулю.

7.2.1.1. Целочисленное деление

Один операнд целого типа можно разделить на другой операнд целого типа. Хотя математическое частное двух целых не обязательно целое, таблица 7.1 показывает, что выражение, включающее в себя операцию деления с двумя целыми операндами, интерпретируется как выражение целого типа. Результатом такой операции является ближайшее к математическому частному целое, расположенное между нулем и математическим частным (включительно). Например, значение выражения (-8)/3 равно (-2).

7.2.1.2. Комплексное возведение в степень

При возведении комплексного значения в комплексную степень результатом операции x_1**x_2 является главное значение.

7.2.2. Символьная встроенная операция

Символьная встроенная операция // используется для конкатенации двух символьных операндов с одинаковыми параметрами типа. Вычисление символьной встроенной операции вырабатывает результат символьного типа.

Интерпретация символьной встроенной операции //, которая используется при формировании выражения, приведена в таблице 7.3, где x_1 означает операнд слева от знака операции, а x_2-справа.

Таблица 7.3

Интерпретация символьной встроенной операции //

Знак операции	Название операции	Использование знака операции	Интерпретация
//	Конкатенация	x_1 // x_2	Сцепляет x_1 с x_2

Результат символьной встроенной операции - это строка символов, значением которой является значение x_1, сцепленное справа со значением x_2. Длина этой строки равна сумме длин x_1 и x_2. Скобки, используемые для задания порядка вычисления, не влияют на значение символьного выражения. Например, значением выражения ('AB'//'CDE')//'F' является строка 'ABCDEF'. Точно так же значением выражения 'AB'('CDE'//'F') является строка 'ABCDEF'.

7.2.3. Встроенные операции отношения

Встроенная операция отношения используется для сравнения значений двух операндов с помощью знаков встроенных операций отношения `.LT.`, `.LE.`, `.GT.`, `.GE.`, `.EQ.`, `.NE.`, `<`, `<=`, `>`, `>=`, `==`, `/=`. Допустимые типы данных для операндов встроенной операции отношения описаны в 7.1.2. Заметим, что, как показано в таблице 7.1, встроенная операция отношения не должна использоваться для сравнения значений выражения числового типа с выражением символьного или логического типа. Кроме того, нельзя сравнивать два операнда логического типа, комплексный операнд можно сравнивать с другим числовым операндом только используя операции .EQ., .NE., == или /=, а два символьных операнда можно сравнивать только в случае одинаковых значений параметра вида у них.

Вычисление встроенной операции отношения вырабатывает результат логического типа.

Интерпретация встроенных операций отношения приведена в табл. 7.4, где x_1 означает операнд слева от знака операции, а x_2 - справа. Знаки операций `<`, `<=`, `>`, `>=`, `==`, `/=` имеют ту же семантику, что, соответственно, знаки операции `.LT.`, `.LE.`, `.GT.`, `.GE.`, `.EQ.` и `.NE.`

В издательстве "Финансы и статистика"
в 1998 г.
выходит из печати учебник

Пятибратов А.П., Гудыно Л.П. Кириченко А.А. **Вычислительные системы, сети и телекоммуникации, 30 л.**

Рассматриваются принципы построения, архитектура, функциональная и структурная организация, основные устройства и звенья вычислительных систем и телекоммуникационных вычислительных сетей. Указаны области их применения. Описываются их программное обеспечение, функционирование, эффективность и перспективы развития.

Для студентов вузов, обучающихся по специальности "Информационные системы в экономике" и смежных специальностей.

Книгу Вы можете приобрести в издательстве по адресу:
101000, Москва, ул. Покровка, 7
(метро "Китай-город", выход на ул. Маросейка)
Тел.: **(095) 925-35-02, 923-80-42**
Факс **(095) 925-09-57**

Таблица 7.4

Интерпретация встроенных операций отношения

Знак операции	Название операции	Использование знака операции	Интерпретация
.LT. <	Меньше чем Меньше чем	x_1 .LT. x_2 $x_1 < x_2$	x_1 меньше, чем x_2 x_1 меньше, чем x_2
.LE. <=	Меньше или равно Меньше или равно	x_1 .LE. x_2 $x_1 < x_2$	x_1 меньше или равно x_2 x_1 меньше или равно x_2
.GT. >	Больше, чем Больше, чем	x_1 .GT. x_2 $x_1 > x_2$	x_1 больше, чем x_2 x_1 больше, чем x_2
.GE. >=	Больше или равно Больше или равно	x_1 .GE. x_2 $x_1 >= x_2$	x_1 больше или равно x_2 x_1 больше или равно x_2
/EQ. ==	Равно Равно	x_1 .EQ. x_2 $x_1 == x_2$	x_1 равно x_2 x_1 равно x_2
.NE. /=	Не равно Не равно	x_1 .NE. x_2 $x_1 /= x_2$	x_1 не равно x_2 x_1 не равно x_2

Числовая встроенная операция отношения интерпретируется как имеющая логическое значение "истина", если значения операндов удовлетворяют отношению, определяемому знаком этой операции.

Числовая встроенная операция отношения интерпретируется как имеющая логическое значение "ложь", если значения операндов не удовлетворяют отношению, определяемому знаком этой операции.

В числовой операции отношения

x_1 оп-отношения x_2

если тип или параметры типа x_1 и x_2 отличаются, их значения до вычисления преобразуются к типу и параметрам типа выражения x_1+x_2.

Символьная встроенная операция отношения интерпретируется как имеющая логическое значение "истина", если значения операндов удовлетворяют отношению, определяемому знаком этой операции. Символьная встроенная операция отношения интерпретируется как имеющая логическое значение "ложь", если значения операндов не удовлетворяют отношению, определяемому знаком этой операции.

Для символьной встроенной операции отношения операнды сравниваются в каждый момент по одному символу, начиная с первого символа каждого символьного операнда. Если операнды имеют неодинаковую длину, более короткий операнд как бы дополняется справа пробелами до длины второго операнда. Если и x_1, и x_2 нулевой длины, то x_1 равно x_2; если каждый символ x_1 совпадает с символом в соответствующей позиции x_2, то x_1 равно x_2. Иначе в первой же позиции, где символьные операнды различаются, символьный операнд x_1 считается меньшим, чем x_2, если значение символа в этой позиции x_1 предшествует значению x_2 в упорядочивающей последовательности (4.3.2.1.1); x_1 больше, чем x_2, если значение символа x_1 в этой позиции следует за x_2 в упорядочивающей последовательности. Заметим, что упорядочение символов зависит частично от процессора; однако результат использования операций .EQ., .NE., == и /= не зависит от этого упорядочения.

Заметим, что для нестандартных символьных типов дополняющий символ зависит от процессора.

7.2.4. Логические встроенные операции

Логическая операция используется для того, чтобы выразить логическое вычисление. Выполнение логической операции вырабатывает результат логического типа. Допустимые типы данных для операндов логических встроенных операций описаны в 7.1.2. Логические операции и их интерпретация при формировании выражения приведены в табл. 7.5, где x_1 означает операнд слева от знака операции, а x_2 - справа от него.

Таблица 7.5

Интерпретация логических встроенных операций

Знак операции	Название операции	Использование знака операции	Интерпретация
.NOT.	Логическое отрицание	.NOT. x_2	"истина", если x_2 "ложь".
.AND.	Логическая конъюнкция	x_1 .AND. x_2	"истина", если и x_1, и x_2 - "истина"
.OR.	Логическая включающая дизъюнкция	x_1 .OR. x_2	"истина", если x_1 и/или x_2 - "истина"
.NEQV.	Логическая не-эквивалентность	x_1 .NEQV. x_2	"истина", если либо x_1, либо x_2 (но не оба) - "истина".
.EQV.	Логическая эквивалентность	x_1 .EQV. x_2	"истина", если и x_1, и x_2 - "истина", либо и x_1, и x_2 - "ложь"

Значения логических встроенных операций показаны в табл.7.6.

Таблица 7.6

Значение логических встроенных операций

x_1	x_2	.NOT. x_2	x_1 .AND. x_2	x_1 .OR. x_2	x_1 .EQV. x_2	x_1 .NEQV. x_2
истина	истина	ложь	истина	истина	истина	ложь
истина	ложь	истина	ложь	истина	ложь	истина
ложь	истина	ложь	ложь	истина	ложь	истина
ложь	ложь	истина	ложь	ложь	истина	ложь

7.3. Интерпретация определяемых операций

Интерпретация определяемой операции обеспечивается функцией, которая определяет эту операцию. Тип, параметры типа и интерпретация выражения, содержащего определяемую операцию, не зависят от типа и параметров типа какого-либо большего выражения, в котором находится данное. Операции <, <=, >, >=, == и /= всегда имеют ту же интерпретацию, что .LT., .LE., .GT., .GE., .EQ. и .NE., соответственно.

7.3.1. Унарная определяемая операция

Функция определяет унарную операцию *оп* x_2, если:

1) функция, описанная с помощью оператора FUNCTION (12.5.2.2) или ENTRY (12.5.2.5), имеет один формальный аргумент d_2;

2) блок интерфейса (12.3.2.1) предусматривает функцию с *обобщенным-спецификатором* OPERATOR (*оп*);

3) тип x_2 такой же, как у формального аргумента d_2;

4) параметры типа x_2 (если они есть) соответствуют параметрам типа d_2;
5) размерность x_2 и его конфигурация (если это массив) соответствуют d_2.

7.3.2. Бинарная определяемая операция

Функция определяет бинарную операцию x_1 *оп* x_2, если:

1) функция описана с помощью оператора FUNCTION (12.5.2.2) или ENTRY (12.5.2.5) с двумя формальными аргументами d_1 и d_2;
2) блок интерфейса (12.3.2.1) предусматривает функцию с *обобщенным-спецификатором* OPERATOR (*оп*);
3) типы x_1 и x_2 те же, что и у формальных аргументов d_1 и d_2, соответственно;
4) параметры типа x_1 и x_2 (если они есть) соответствуют параметрам d_1 и d_2, соответственно;
5) размерность x_1 и x_2 и их конфигурации, если хотя бы один из них - массив, соответствуют d_1 и d_2.

7.4. Старшинство операций

Существует старшинство (приоритет) встроенных операций и операций расширения, вытекающее из общей формы в 7.1.1. Старшинство определяет порядок, в котором комбинируются операнды, за исключением тех случаев, когда этот порядок изменяется при использовании скобок. Порядок старшинства операций представлен в табл.7.7.

Таблица 7.7

Виды операций и их относительное старшинство

Вид операции	Знаки операций	Старшинство
Расширения	*Знак-определяемой-унарной-операции*	Высшее
Числовая	**	
Числовая	* или /	
Числовая	Унарный + или -	
Числовая	Бинарный + или -	
Символьная	//	
Отношения	.EQ., .NE., .LT., .LE., .GT., .GE.	
	== , /= , < , <= , > , >=	
Логическая	.NOT.	
Логическая	.AND.	
Логическая	.OR.	
Логическая	.EQV. или .NEQV.	
Расширения	*Знак-определяемой-бинарной-операции*	Низшее

Определяемая операция имеет старшинство своего знака операции.

Например, в выражении

```
-A**2
```

операция возведения в степень (**) имеет более высокий приоритет, чем операция изменения знака (унарный -), следовательно, операнды операции возведения в степень комбинируются для образования выражения, которое используется как операнд для операции изменения знака.

Интерпретация выражения, приведенного выше, такая же, как для выражения

```
-(A**2)
```

Общая форма выражения (7.1.1) также устанавливает приоритет операций одного синтаксического класса, что позволяет определить порядок, в котором операнды комбинируются, если этот порядок не изменяется использованием скобок. Например, в интерпретации *выражения-2-уровня*, содержащего две или более бинарные операции + или - , каждый операнд (*слагаемое*) комбинируется слева направо. Следствием общей формы (7.1.1) является и аналогичная интерпретация слева направо для *множителя* в *слагаемом*, так же, как и для выражений других видов. Однако для интерпретации выражения вида: *множитель*, когда две или более операции возведения в степень (**) комбинируют операнды вида: *выражение-1-уровня*, каждое *выражение-1-уровня* формируется справа налево. Например, выражения

```
2.1 + 3.4 + 4.9
2.1 * 3.4 * 4.9
2.1 / 3.4 / 4.9
2 ** 3 ** 4
'AB' // 'CD' // 'EF'
```

интерпретируются так же, как выражения

```
(2.1 + 3.4) + 4.9
(2.1 * 3.4) * 4.9
(2.1 / 3.4) / 4.9
2 ** (3 ** 4)
('AB' // 'CD') // 'EF'
```

Как следует из общей формы (7.1.1), только первое *слагаемое* в *выражении-2-уровня* может быть предварено операцией идентичности (унарный +) или изменения знака (унарный -). Заметим также, что эти правила образования не допускают выражений, содержащих два стоящих рядом числовых знака операции, например, A**-B или A+-B. Однако допустимы такие выражения, как A**(-B) и A+(-B). Правила разрешают, чтобы за знаком бинарной операции или встроенной унарной операции следовал знак определяемой операции, например:

```
A * .INVERSE. B
- .INVERSE. (B)
```

В качестве другого примера в выражении

```
A .OR. B .AND. C
```

общая форма предполагает более высокий приоритет для операции .AND., чем для операции .OR.; следовательно, приведенное выше выражение интерпретируется аналогично выражению

```
A .OR. (B .AND. C)
```

Выражение может содержать более одного вида операции. Например, логическое выражение

```
L .OR. A + B .GE. C
```

где A, B и C - вещественного типа, а L - логического, содержит числовую операцию, операцию отношения и логическую операцию. Это выражение можно проинтерпретировать так же, как выражение

L .OR. ((A+B) .GE. C)

Например, если:

1) операция ** расширена до логического типа,
2) операция .STARSTAR. определена как дубликат функции ** для вещественных,
3) .MINUS. определена как дубликат унарной операции -,
4) L1 и L2 - логические, а X и Y - вещественные,

тогда по старшинству L1**L2 выше, чем X*Y; X*Y выше, чем X.STARSTAR.Y; а .MINUS.X выше, чем -X.

7.5. Присваивание

Выполнение присваивания приводит к определению или переопределению переменной. Выполнение оператора присваивания указателю либо связывает его с адресатом, либо делает статус связанности указателя - "несвязанный" или "неопределенный". Выполнение оператора WHERE или конструкции WHERE приводит к вычислению выражений и присваиванию значений в операторе присваивания массивам по маске в соответствии со значением логического выражения-массива.

7.5.1. Оператор присваивания

Выполнение оператора присваивания может определить или переопределить переменную.

7.5.1.1. Общая форма

R735 *оператор-присваивания* **есть** *переменная* = *выражение*

где *переменная* определяется правилом R601, а *выражение* - R723.

Ограничение: *переменная* в *операторе-присваивания* не должна быть массивом неопределенного размера.

Примеры оператора присваивания:

```
A = 3.5 + X * Y
I = INT(A)
```

Оператор присваивания может быть либо встроенным, либо определяемым.

7.5.1.2. Встроенный оператор присваивания

Встроенный оператор присваивания - это такой оператор присваивания, в котором *переменная* и *выражение* согласованы по конфигурации и

1) *переменная* и *выражение* имеют встроенные типы, как это определено в таблице 7.8 для присваивания, или
2) *переменная* и *выражение* имеют один и тот же производный тип, и для объектов этого типа не существует определяемого присваивания.

Встроенный оператор числового присваивания - это оператор встроенного присваивания, для которого *выражение* и *переменная* имеют числовые типы. **Встроенный оператор символьного присваивания** - это встроенный оператор присваивания, в котором *выражение* и *переменная* - символьного типа с одинаковыми значениями параметра вида. **Встроенный оператор логического присваивания** - это встроенный оператор присваивания, в котором *переменная* и *выражение* имеют тип логический. **Встроенный оператор присваивания для производного типа** - это встроенный оператор присваивания, в котором *переменная* и *выражение* принадлежат к одному производному типу, и нет доступного блока интерфейса со спецификатором ASSIGNMENT (=) для объектов этого типа. **Встроенный оператор присваивания массиву** - это встроенный оператор присваивания, в котором *переменная* - это массив. *Переменная* не может быть многозначной секцией массива (6.2.2.3.2).

Таблица 7.8

Согласование типов для встроенного оператора присваивания *переменная = выражение*

Тип переменной	Тип выражения
целый	целый, вещественный, комплексный
вещественный	целый, вещественный, комплексный
комплексный	целый, вещественный, комплексный
символьный	символьный с тем же параметром типа, что *переменная*
логический	логический
производный тип	тот же производный тип, что у *переменной*

7.5.1.3. Определяемый оператор присваивания

Определяемый оператор присваивания - это оператор присваивания, который не является встроенным оператором присваивания; он описывается с помощью подпрограммы и блока интерфейса (12.3.2.1) со специфицикацией ASSIGNMENT (=).

7.5.1.4. Правила согласования для встроенного присваивания

Для встроенного оператора присваивания *переменная* и *выражение* должны быть согласованы по конфигурации, и если *выражение* является массивом, *переменная* тоже должна быть массивом. Типы *переменной* и *выражения* должны согласовываться с правилами из табл. 7.8.

Если *переменная* - указатель, он должен быть связан с таким определимым адресатом, чтобы тип, параметры типа и конфигурация адресата и *выражения* согласовывались.

Для встроенного оператора числового присваивания *переменная* и *выражение* могут иметь разные числовые типы или разные параметры типа. В этом случае значение *выражения* преобразуется к типу или к параметрам типа *переменной* в соответствии с правилами, приведенными в табл. 7.9.

Таблица 7.9

Числовое преобразование и оператор присваивания

переменная = выражение

Тип *переменной*	Присваиваемое значение
целый	INT (*выражение*, KIND=KIND (*переменная*))
вещественный	REAL (*выражение*, KIND=KIND (*переменная*))
комплексный	CMPLX (*выражение*, KIND=KIND (*переменная*))

Примечание: функции INT, REAL, CMPLX и KIND - обобщенные функции, определенные в 13.13.

Для логического встроенного оператора присваивания *переменная* и *выражение* могут иметь разные параметры вида, при этом значение *выражения* приводится к параметру вида *переменной*.

Для символьного оператора встроенного присваивания *переменная* и *выражение* должны иметь одинаковые значения параметра вида, но могут иметь разные параметры длины. Если это так, преобразование *выражения* к длине *переменной* производится следующим образом:

1) если длина *переменной* меньше, значение *выражения* усекается справа до длины *переменной*;
2) если длина *переменной* больше, значение *выражения* расширяется вправо пробелами до длины *переменной*.

Заметим, что для нестандартных символьных типов конкретный дополняющий символ зависит от процессора.

7.5.1.5. Интерпретация встроенного присваивания

Выполнение встроенного присваивания вызывает вычисление *выражения* и всех выражений внутри *переменной* (7.1.7), возможное преобразование *выражения* к типу и параметрам типа *переменной* (табл.7.9) и определение *переменной* результирующим значением. Исполнение присваивания должно иметь такой эффект, как если бы вычисление всех операций в *выражении* и *переменной* имело место прежде, чем какая-либо часть *переменной* будет определена с помощью этого присваивания. Вычисление выражений, входящих в *переменную*, не должно ни влиять на вычисление *выражения*, ни зависеть от него. *Переменной* не присваивается значение, если *переменная* - символьного типа с нулевой длиной или массив нулевого размера.

Если *переменная* - указатель, значение *выражения* присваивается адресату *переменной*.

И *переменная*, и *выражение* могут содержать ссылки на любую часть *переменной*. Например, в символьном операторе встроенного присваивания

```
STRING(2:5) = STRING(1:4)
```

присваивание первого символа STRING второму не влияет на вычисление STRING(1:4). Например, если значение STRING до присваивания было 'ABCDEF', то после присваивания значение будет 'AABCDF'.

Если *выражение* во встроенном присваивании - скаляр, а *переменная* - массив, то *выражение* рассматривается как массив той же конфигурации, что и *переменная*, в котором каждый элемент равен скалярному значению *выражения*.

Когда *переменная* во встроенном присваивании - массив, присваивание производится поэлементно для соответствующих элементов *переменной* и *выражения*. Например, если A и B - массивы одной конфигурации, встроенное присваивание массива

```
A=B
```

выполняет присваивание значений соответствующих элементов B элементам A; т.е. первый элемент B присваивается первому элементу A, второй элемент B присваивается второму элементу A и т.д.

Процессор может производить поэлементное присваивание в любом выбранном порядке.

Например, следующий программный сегмент приводит к получению значений элементов массива X в обратном порядке:

```
REAL X(10)
...
X(1:10) = X(10:1:-1)
```

Встроенное присваивание производного типа выполняется так, как если бы каждый компонент *выражения* был присвоен соответствующему компоненту *переменной* с использованием операторов присваивания указателю (7.5.2) для компонентов-указателей или операторов встроенного присваивания для остальных компонентов. Процессор может выполнять покомпонентное присваивание в любом порядке и любым способом, дающим тот же эффект.

Например, для встроенного присваивания производного типа, когда C и D относятся к одному производному типу, у которого P - компонент-указатель, а компоненты-не-указатели S, T, U и V соответственно целого, логического, символьного и другого производного типов, встроенное присваивание

```
C=D
```

присваивает указатель D%P объекту C%P, а D%S - объекту C%S, используя встроенный оператор числового присваивания, D%T - объекту C%T, используя встроенный оператор логического присваивания, D%U - объекту C%U, используя встроенный оператор символьного присваивания и D%V - объекту C%V, используя встроенный оператор присваивания производного типа.

Когда *переменная* - подобъект, присваивание не влияет на статус определенности или на значение других частей объекта. Например, если *переменная* - секция массива, присваивание не влияет на статус определенности или на значение элементов массива-основы, не входящих в эту секцию массива.

7.5.1.6. Интерпретация определяемых операторов присваивания

Интерпретация определяемых операторов присваивания обеспечивается подпрограммой, которая определяет эту операцию.

Подпрограмма определяет присваивание $x_1 = x_2$ если имеет место следующее:

1) она описана с помощью оператора SUBROUTINE (12.5.2.3) или ENTRY (12.5.2.5) с двумя формальными аргументами d_1 и d_2;
2) блок интерфейса (12.3.2.1) обеспечивает подпрограмму со спецификатором ASSIGNMENT(=);
3) типы x_1 и x_2 такие же, как у формальных аргументов d_1 и d_2, соответственно;
4) параметры типа x_1 и x_2 (если они есть) совпадают с параметрами типа d_1 и d_2, соответственно;
5) размерности x_1 и x_2 и их конфигурации, если хотя бы один из них - массив, соответствуют d_1 и d_2.

Заметим, что x_1 и x_2 не должны обязательно быть оба числового типа, оба - логического типа, либо оба - символьного типа с одинаковым значением параметра вида.

7.5.2. Оператор присваивания указателю

R736 *оператор-присваивания-указателю* **есть** *объект-указатель* => *адресат*

R737 *адресат* **есть** *переменная*
или *выражение*

Ограничение: *объект-указатель* должен иметь атрибут POINTER.

Ограничение: *переменная* должна иметь атрибут TARGET или быть подобъектом объекта, имеющего атрибут TARGET, либо она должна иметь атрибут POINTER.

Ограничение: *адресат* должен иметь тот же тип, те же параметры типа и размерность, что и указатель.

Ограничение: *адресат* не должен быть секцией массива с векторным индексом.

Ограничение: *выражение* должно вырабатывать ссылку на результат.

Если *адресат* не является указателем, оператор присваивания указателю связывает *объект-указатель* с адресатом. Если *адресат* сам является указателем, который связан, *объект-указатель* связывается с тем же объектом, что и *адресат*. Если *адресат* является указателем, не связанным в данный момент, *объект-указатель* тоже становится не связанным. Если *адресат* является указателем со статусом связанности - не определен, *объект-указатель* тоже получает статус связанности - не определен.

Любая предыдущая связь между *объектом-указателем* и адресатом разрывается.

Можно осуществить присваивание компоненту структуры, являющемуся указателем, путем выполнения встроенного оператора присваивания производного типа (7.5.1.5) или определяемого оператора присваивания (7.5.1.6).

Помимо присваивания указателю, указатель можно связать с адресатом путем размещения указателя.

На указатель нельзя ссылаться и нельзя его определять, пока он не связан с адресатом, на который можно ссылаться или который можно определять.

Примеры операторов присваивания указателю:

```
NEW_NODE % LEFT => CURRENT_NODE
SIMPLE_NAME => STRUCTURE % SUBSTRUCT % COMPONENT
ROW => MAT2D (N,:)
WINDOW => MAT2D (I-1:I+1, J-1:J+1)
POINTER_OBJECT => POINTER_FUNCTION (ARG_1, ARG_2)
EVERY_OTHER => VECTOR (1:N:2)
```

7.5.3. Присваивание массивам по маске - WHERE

Присваивание массивам по маске используется для вычисления выражений по маске и присваивания значений в операторах присваивания массиву в соответствии со значением логического выражения-массива.

7.5.3.1. Общая форма присваивания массивам по маске

Присваивание массиву по маске - это либо оператор WHERE, либо конструкция WHERE.

R738 *оператор-WHERE* **есть** WHERE (*выражение-маска*) ♦
♦ *оператор-присваивания*

R739 *конструкция-WHERE* **есть** *оператор-конструкции-WHERE*
[*оператор-присваивания*]...
[*оператор-ELSEWHERE*
[*оператор-присваивания*]...]
оператор-ENDWHERE

R740 *оператор-конструкции-WHERE* **есть** WHERE (*выражение-маска*)

R741 *выражение-маска* **есть** *логическое-выражение*

R742 *оператор-ELSEWHERE* **есть** ELSEWHERE

R743 *оператор-END-WHERE* **есть** END WHERE

Ограничение: *выражение-маска* и определяемая переменная в каждом *операторе-присваивания* должны быть массивами одной конфигурации.

Ограничение: *оператор-присваивания* не должен быть определяемым присваиванием.

Примеры присваивания массиву по маске:

```
WHERE (TEMP > 100.0) TEMP = TEMP - REDUCE_TEMP

WHERE (PRESSURE <= 1.0)
    PRESSURE = PRESSURE + INC_PRESSURE
    TEMP = TEMP - 5.0
ELSEWHERE
    RAINING = .TRUE.
END WHERE
```

7.5.3.2. Интерпретация присваивания массиву по маске

Когда выполняется *оператор-присваивания* в *операторе-WHERE*, вычисляется *выражение* в операторе присваивания для всех элементов, где *выражение-маска* имеет значение "истина", и результат присваивается соответствующим элементам *переменной* по правилам встроенного присваивания (7.5.1). Когда выполняется *конструкция-WHERE*, *выражение-маска* вычисляется и результат сохраняется процессором. Выполняются последовательно все *операторы-присваивания* в *блоке-WHERE*, как если бы это были операторы WHERE (*выражение-маска*) *оператор-присваивания*, и затем последовательно все *операторы-присваивания* в блоке ELSEWHERE, как если бы это были операторы WHERE (.NOT.*выражение-маска*) *оператор-присваивания*.

Если в *выражении* оператора присваивания есть ссылка на не поэлементную функцию, функция вычисляется без какого-либо управления через *выражение-маску*, т.е. полностью вычисляются все ее выражения-аргументы и сама функция. Если результатом является массив и ссылка не находится в списке аргументов непоэлементной функции, для вычисления каждого *выражения* выбираются элементы, соответствующие значениям "истина" в *выражении-маске* (значениям "ложь" в *выражении-маске* после оператора ELSEWHERE).

Если в *выражении* оператора присваивания есть поэлементная встроенная операция или ссылка на функцию, операция выполняется или функция вычисляется только для элементов, соответствующих значениям "истина" в *выражении-маске* (значениям "ложь" - после ELSEWHERE).

В присваивании массиву по маске только оператор WHERE или оператор конструкции WHERE могут быть адресатами операторов ветвления. Значение *выражения-маски*, вычисляемое в начале присваивания массиву по маске, выполняет роль маски при исполнении присваивания массиву по маске. Последующие изменения объектов в *выражении-маске* не влияют на значение этой маски. Выполнение ссылки на функцию в *выражении-маске* оператора WHERE может влиять на объекты в операторе присваивания. Выполнение оператора END WHERE никакого действия не производит.

Примеры ссылок на функцию в присваиваниях массиву по маске:

```
WHERE (A > 0)
    A = LOG (A)                    ! LOG вызывается только для
                                   ! положительных элементов
    A = A / SUM (LOG (A))          ! LOG вызывается для всех
                                   ! элементов
END WHERE
```

Глава 8. УПРАВЛЕНИЕ ВЫПОЛНЕНИЕМ

Последовательность выполнения может управляться конструкциями, содержащими блоки, и некоторыми выполняемыми операторами, которые используются для изменения последовательности выполнения.

8.1. Выполняемые конструкции

Для управления последовательностью выполнения могут быть использованы следующие выполняемые конструкции, содержащие блоки:

1) конструкция IF;
2) конструкция CASE;
3) конструкция DO.

Кроме того, имеется неблочная форма конструкции DO.

Блок - это последовательность выполняемых операторов, которая рассматривается как единое целое.

R801 *блок* **есть** [*конструкция-раздела-выполняемых-операторов*]...

Выполняемые конструкции могут быть использованы для управления тем, какие блоки программы выполняются или сколько раз выполняется отдельный блок. Блоки всегда ограничены операторами, характерными для той конструкции, в которой они расположены. Однако в некоторых формах конструкции DO последовательность выполняемых конструкций без конечного ограничивающего оператора должна, тем не менее, подчиняться всем остальным правилам управления блоками (8.1.1). Заметим, что блок не должен обязательно содержать какие-либо выполняемые конструкции. Выполнение такого блока никакого действия не производит.

Любая из этих трех конструкций может иметь имя, тогда имя должно быть первой лексемой первого оператора конструкции и последней лексемой конструкции. В фиксированной форме исходного текста имя конструкции должно располагаться после шестой колонки.

Оператор внутри конструкции, который не содержит имени конструкции, **принадлежит** самой внутренней из вложенных конструкций; если же такой оператор содержит имя конструкции, он принадлежит конструкции с этим именем.

Пример конструкции, содержащей блоки:

```
IF (A>0.0) THEN
    B=SQRT(A)                    ! Эти два оператора
    C=LOG(A)                     ! образуют блок
END IF
```

8.1.1. Правила управления блоками

8.1.1.1. Выполняемые конструкции в блоках

Если блок содержит выполняемую конструкцию, она должна полностью находиться в этом блоке.

8.1.1.2. Передача управления в блоках

Запрещена передача управления извне внутрь блока. Допускается передача управления внутри блока и из блока наружу. Например, если оператор внутри блока помечен меткой, оператор GOTO, использующий эту метку, может появиться в этом же блоке. В блоке могут находиться ссылки на функции и подпрограммы (12.4.2, 12.4.3, 12.4.4, 12.4.5).

8.1.1.3. Выполнение блока

Выполнение блока начинается с выполнения первой выполняемой конструкции в блоке. Если в блоке нет передач управления наружу, выполнение блока завершается, когда выполнится последняя выполняемая конструкция в последовательности. Действие на конечной границе зависит от конкретной конструкции и от блока в этой конструкции. Обычно это передача управления.

8.1.2. Конструкция IF

Конструкция IF обеспечивает выбор для выполнения не более чем одного из составляющих ее блоков. **Оператор IF** управляет выполнением одного оператора (8.1.2.4).

8.1.2.1. Форма конструкции IF

R802 *конструкция-IF* **есть** *оператор-IF-THEN*
блок
[*оператор-ELSE-IF*
блок]...
[*оператор-ELSE*
блок]
оператор-END-IF

R803 *оператор-IF-THEN* **есть** [*имя-конструкции-IF* :] ♦
♦ IF (*скалярное-логическое-выражение*) THEN

R804 *оператор-ELSE-IF* **есть** ELSE IF ♦
♦ (*скалярное-логическое-выражение*) THEN [*имя-конструкции-IF*]

R805 *оператор-ELSE* **есть** ELSE [*имя-конструкции-IF*]

R806 *оператор-END-IF* **есть** END IF[*имя-конструкции-IF*]

Ограничение: если *оператор-IF-THEN* конструкции IF помечен *именем-конструкции-IF*, соответствующий *оператор-END-IF* должен специфицировать это же *имя-конструкции-IF*. Если же *оператор-IF-THEN* конструкции IF не имеет *имени-конструкции-IF*, в соответствующем *операторе-END-IF* не должно содержаться *имя-конструкции-IF*. Если *оператор-ELSE-IF* или *оператор-ELSE* содержат *имя-конструкции-IF*, соответствующий *оператор-IF-THEN* должен содержать это же *имя-конструкции-IF*.

8.1.2.2. Выполнение конструкции IF

Из блоков, содержащихся в конструкции IF, выполняется не более одного. Если в конструкции есть оператор ELSE, один из блоков, содержащихся внутри конструкции, обязательно будет выполняться. Скалярные логические выражения вычисляются в порядке их появления в конструкции, пока не будет найдено выражение со значением "истина" или пока не появится оператор ELSE или END IF. Если найдено значение "истина" или оператор ELSE, то выполняется блок, следующий непосредственно за ним, и этим завершается выполнение всей конструкции. Выражения во всех оставшихся операторах ELSE IF не вычисляются.

Если ни одно из выражений не имело значения "истина" и нет оператора ELSE, выполнение конструкции завершается без выполнения какого-либо из содержащихся внутри нее блоков.

На операторы ELSE IF и ELSE нельзя передавать управление. Переход на оператор END IF допустим как изнутри конструкции IF, так и извне. Выполнение оператора END IF никакого действия не производит.

8.1.2.3. Примеры конструкции IF

```
IF (CAR.EQ.'RESET') THEN
    I=0; J=0; K=0
END IF

PROOF_DONE: IF (PROP) THEN
    WRITE (3, '("QED")')
    STOP
ELSE
    PROP = NEXTPROP
END IF PROOF_DONE

IF (A.GT.0) THEN
    B=C/A
    IF (B.GT.0) THEN
        D=1.0
    END IF
ELSE IF (C.GT.0) THEN
    B=A/C
    D=-1.0
ELSE
    B=ABS (MAX(A,C))
    D=0
END IF
```

8.1.2.4. Оператор IF

Оператор IF управляет выполнением отдельного оператора действия (R216).

R807 *оператор-IF* **есть** IF (*скалярное-логическое-выражение*) *оператор-действия*

Ограничение: *оператор-действия* не должен быть оператором IF, END PROGRAM, END FUNCTION или END SUBROUTINE.

Выполнение оператора IF вызывает вычисление скалярного логического выражения и, если его значение "истина", выполняется оператор действия, а если значение "ложь", оператор действия не выполняется и выполнение продолжается так, как если бы был выполнен оператор CONTINUE (8.3).

Разрешается, чтобы выполнение ссылки на функцию в скалярном логическом выражении влияло на объекты в операторе действия.

Пример оператора IF:

```
IF (A > 0.0) A = LOG(A)
```

8.1.3. Конструкция CASE

Конструкция CASE выбирает для выполнения не более одного из образующих ее блоков.

8.1.3.1. Форма конструкции CASE

R808 *конструкция-CASE* **есть** *оператор-SELECT-CASE*
[*оператор-CASE*
блок]...
оператор-END-SELECT

R809 *оператор-SELECT-CASE* **есть** [*имя-конструкции-CASE:*] ♦
♦ SELECT CASE (*выражение-выбора*)

R810 *оператор-CASE* **есть** CASE *селектор-CASE* [*имя-конструкции-CASE*]

R811 *оператор-END-SELECT* **есть** END SELECT [*имя-конструкции-CASE*]

Ограничение: если *оператор-SELECT-CASE* конструкции CASE помечен *именем-конструкции-CASE*, в соответствующем *операторе-END-SELECT* должно быть задано это же *имя-конструкции-CASE*. Если же *оператор-SELECT-CASE* конструкции CASE не содержит *имени-конструкции-CASE*, соответствующий *оператор-END-SELECT* не должен содержать *имени-конструкции-CASE*. Если *оператор-CASE* помечен *именем-конструкции-CASE*, соответствующий *оператор-SELECT-CASE* должен содержать это же *имя-конструкции-CASE*.

R812 *выражение-выбора* **есть** *скалярное-целое-выражение*
или *скалярное-символьное-выражение*
или *скалярное-логическое-выражение*

R813 *селектор-CASE* **есть** (*список-диапазонов-значений-CASE*)
или DEFAULT

Ограничение: не более чем один *селектор-CASE* формы DEFAULT может находиться в одной *конструкции-CASE*.

R814 *диапазон-значений-CASE* **есть** *значение-CASE*
или *значение-CASE* :
или : *значение-CASE*
или *значение-CASE* : *значение-CASE*

R815 *значение-CASE* **есть** *скалярное-целое-выражение-инициализации*
или *скалярное-символьное-выражение-инициализации*
или *скалярное-логическое-выражение-инициализации*

Ограничение: для одной конструкции CASE каждое *значение-CASE* должно быть того же типа, что *выражение-выбора*. Для символьного типа допускаются различия в длине, но параметры вида должны быть одинаковыми.

Ограничение: если *выражение-выбора* логического типа, нельзя использовать *диапазон-значений-CASE* с двоеточием.

Ограничение: для одной *конструкции-CASE* недопустимо перекрытие *диапазонов-значений-CASE*, т.е. не должно быть такого возможного значения *выражения-выбора*, которое соответствовало бы более чем одному *диапазону-значений-CASE*.

8.1.3.2. Выполнение конструкции CASE

Выполнение оператора SELECT CASE вызывает вычисление выражения выбора. Результирующее значение называется **индексом-CASE**. В случае списка диапазонов значений CASE, если индекс-CASE соответствует какому-либо из диапазонов значений CASE в списке, возникает соответствие, определяемое следующим образом (*c* - значение индекса-CASE):

1) если диапазон значений-CASE содержит одно значение *v* без двоеточия, соответствие для данных логического типа имеет место, если значение выражения (*c*.EQV.*v*) есть "истина", а для данных целого или символьного типов значение выражения (*c*.EQ.*v*) -"истина";
2) если диапазон значений-CASE задан в виде *l*:*u*, соответствие имеет место, когда значение выражения *l*.LE.*c*.AND.*c*.LE.*u* -"истина";
3) если диапазон значений-CASE задан в виде: *l*:, соответствие имеет место, когда значение выражения *l*.LE.*c* - "истина";
4) если диапазон-значений-CASE задан в виде :*u*, соответствие имеет место, когда значение выражения *c*.LE.*u* - "истина";
5) если нет ни одного соответствия селектору и присутствует селектор DEFAULT, он и соответствует индексу-CASE;
6) если же нет ни соответствия селекторам, ни DEFAULT, соответствия нет вообще.

Выполняется блок, следующий за оператором CASE, который содержит соответствующий селектор (если таковой есть). Этим завершается выполнение всей конструкции.

Выполняется один и только один из блоков конструкции CASE.

На оператор CASE нельзя передавать управление. Переход на оператор END SELECT разрешен только внутри конструкции CASE.

8.1.3.3. Примеры конструкции CASE

Целая функция определения знака:

```
INTEGER FUNCTION SIGNUM (N)
SELECT CASE (N)
CASE (:-1)
    SIGNUM = -1
CASE (0)
    SIGNUM = 0
CASE (1:)
    SIGNUM = 1
END SELECT
END
```

Фрагмент программы проверки сбалансированности скобок:

```
CHARACTER (80) :: LINE
 ...
LEVEL = 0
DO I=1,80
    CHECK_PARENS: SELECT CASE (LINE(I:I))
    CASE ('(')
        LEVEL = LEVEL + 1
    CASE (')')
        LEVEL = LEVEL - 1
        IF (LEVEL.LT.0) THEN
            PRINT *, 'UNEXPECTED RIGHT PARENTHESIS'
            EXIT
        END IF

    CASE DEFAULT
        ! Все остальные символы игнорируются
    END SELECT CHECK_PARENS
END DO

IF (LEVEL.GT.0) THEN
    PRINT *, 'MISSING RIGHT PARENTHESIS'
END IF
```

Следующие три фрагмента эквивалентны:

```
IF (SILLY .EQ. 1) THEN
    CALL THIS
ELSE
    CALL THAT
```

```
END IF

SELECT CASE (SILLY .EQ. 1)
CASE (.TRUE.)
    CALL THIS
CASE (.FALSE.)
    CALL THAT;
END SELECT

SELECT CASE (SILLY)
CASE DEFAUT
    CALL THAT
CASE (1)
    CALL THIS
END SELECT
```

Фрагмент программы, показывающий несколько выборов одного блока:

```
SELECT CASE (N)
CASE (1, 3:5, 8)          ! Выбираются 1, 3, 4, 5, 8
    CALL SUB
CASE DEFAULT
    CALL OTHER
END SELECT
```

8.1.4. Конструкция DO

Конструкция DO используется для задания повторяющегося выполнения последовательности выполняемых конструкций. Такая повторяющаяся последовательность называется **циклом**. Для модификации выполнения цикла можно использовать операторы CYCLE и EXIT.

Число итераций цикла может быть определено в начале выполнения конструкции DO или может оставаться неопределенным ("бесконечный цикл" или DO WHILE). В любом случае для немедленного завершения цикла в любом месте конструкции DO может быть выполнен оператор EXIT (8.1.4.4.4). Конкретную итерацию цикла можно сократить путем выполнения оператора CYCLE (8.1.4.4.3).

8.1.4.1. Формы конструкции DO

Конструкция DO может быть записана в блочной форме или в неблочной форме.

R816 *конструкция-DO* **есть** *блочная-конструкция-DO*
или *неблочная-конструкция-DO*

8.1.4.1.1. Форма блочной конструкции DO

R817 *блочная-конструкция-DO* **есть** *оператор-DO*
блок-DO
конец-цикла

R818 *оператор-DO* **есть** *оператор-DO-с-меткой*
или *оператор-DO-без-метки*

R819 *оператор-DO-с-меткой* **есть** [*имя-конструкции-DO* :] DO *метка* ♦
♦ [*управление-циклом*]

R820 *оператор-DO-без-метки* **есть** [*имя-конструкции-DO* :] DO [*управление-циклом*]

R821 *управление-циклом* ♦
♦ **есть** [,] *DO-переменная* = *скалярное-числовое-выражение*, ♦

♦ *скалярное-числовое-выражение* ♦
♦ [, *скалярное-числовое-выражение*]

или [,] WHILE(*скалярное-логическое-выражение*)

R822 *DO-переменная* **есть** *скалярная-переменная*

Ограничение: *DO-переменная* должна быть скалярной переменной целого, стандартного вещественного или вещественного двойной точности типа.

Ограничение: каждое *скалярное-числовое-выражение* в *управлении-циклом* должно иметь тип целый, стандартный вещественный или вещественный двойной точности.

R823 *блок-DO* **есть** *блок*

R824 *конец-цикла* **есть** *оператор-END-DO*
или *оператор-CONTINUE*

R825 *оператор-END-DO* **есть** END DO [*имя-конструкции-DO*]

Ограничение: если в *операторе-DO* есть *имя-конструкции-DO*, *концом-цикла* такой конструкции должен быть *оператор-END-DO* с тем же *именем-конструкции-DO*. Если же в *операторе-DO* нет *имени-конструкции-DO*, его не должно быть и в соответствующем *конце-цикла*.

Ограничение: если оператор-DO - это *оператор-DO-без-метки* , соответствующим *концом-цикла* должен быть *оператор-END-DO* .

Ограничение: если *оператор-DO* - это *оператор-DO-с-меткой* , *конец-цикла* должен быть помечен этой меткой.

8.1.4.1.2. Форма неблочной конструкции DO

R826 *неблочная-конструкция-DO* **есть** *конструкция-DO-оканчивающаяся-оператором-действия*
или *внешняя-конструкция-DO-с-общим-окончанием*

R827 *конструкция-DO-оканчивающаяся-оператором-действия* **есть** *оператор-DO-с-меткой*
тело-цикла
оператор-действия-оканчивающий-цикл

R828 *тело-цикла* **есть** [*конструкция-раздела-выполняемых-операторов*]...

R829 *оператор-действия-оканчивающий-цикл* **есть** *оператор-действия*

Ограничение: *оператор-действия-оканчивающий-цикл* не должен быть оператором CONTINUE, GOTO, RETURN, STOP, EXIT, CYCLE, END FUNCTION, END SUBROUTINE, END PROGRAM, арифметический IF или GOTO по предписанию.

Ограничение: *оператор-действия-оканчивающий-цикл* должен быть идентифицирован меткой, и на нее должен ссылаться соответствующий *оператор-DO-с-меткой* (с той же меткой).

R830 *внешняя-конструкция-DO-с-общим-окончанием* **есть** *оператор-DO-с-меткой*
тело-цикла
конструкция-DO-с-общим-окончанием

R831 *конструкция-DO-с-общим-окончанием* **есть** *внешняя-конструкция-DO-с-общим-окончанием*
или *внутренняя-конструкция-DO-с-общим-окончанием*

R832 *внутренняя-конструкция-DO-с-общим-окончанием* **есть** *оператор-DO-с-меткой*
тело-цикла
общий-оператор-окончания-цикла

R833 *общий-оператор-окончания-цикла* **есть** *оператор-действия*

Ограничение: *общий-оператор-окончания-цикла* не должен быть оператором GO TO, RETURN, STOP, EXIT, CYCLE, END FUNCTION, END SUBROUTINE, END PROGRAM, арифметический IF или GO TO по предписанию.

Ограничение: общий-оператор-окончания-цикла должен быть помечен меткой, и все операторы-DO-с-меткой конструкции-DO-с-общим-окончанием должны ссылаться на эту метку.

Оператор-действия-оканчивающий-цикл, *оператор-общего-окончания-цикла* или *конструкция-DO-с-общим-окончнвнием*, следующие за телом цикла неблочной конструкции DO, называются **окончанием DO** для этой конструкции.

Все конструкции DO в контекстной области, операторы DO которых ссылаются на одну метку, являются неблочными конструкциями DO, и считается, что они **имеют общее окончание** - оператор, помеченный этой меткой.

8.1.4.2. Область конструкции DO

Областью блочной конструкции DO является *блок-DO*, который должен удовлетворять правилам для блоков (8.1.1). В частности, передача управления внутрь такого блока извне запрещена. Переход на окончание блочной конструкции DO допустим только из области этой конструкции DO.

Область неблочной конструкции DO состоит из *тела-цикла* и следующего за ним окончания DO. Конец такого диапазона не ограничивается каким-то конкретным оператором, как в других выполняемых конструкциях (например, END IF); тем не менее он должен удовлетворять правилам для блоков (8.1.1). Передача управления внутрь области или на окончание DO извне запрещена, в частности, переход на *общий-оператор-окончания-цикла* разрешен только в области, соответствующей *внутренней-конструкции-DO-с-общим-окончанием*.

8.1.4.3. Активные и неактивные конструкции DO

Конструкция DO может быть **активной** или **неактивной**. Сначала неактивная, она становится активной только когда выполняется ее оператор DO.

Активная конструкция DO становится неактивной, только когда выполнится одно из условий ее завершения (8.1.4.4.4). Когда конструкция DO становится неактивной, ее *DO-переменная*, если она есть, сохраняет свое последнее значение.

8.1.4.4. Выполнение конструкции DO

Конструкция DO специфицирует цикл, т.е. последовательность выполняемых конструкций, которые выполняются повторно. Имеются три фазы выполнения цикла: инициализация цикла, выполнение цикла и окончание цикла.

8.1.4.4.1. Инициализация цикла

Когда выполняется оператор DO, конструкция DO становится активной. Если *управление-циклом* задано в виде:

DO-переменная = скалярное-числовое-выражение$_1$, *скалярное-числовое-выражение*$_2$ [,*скалярное-числовое-выражение*$_3$]

последовательно выполняются следующие шаги.

1. Устанавливаются начальный параметр $м_1$, конечный параметр $м_2$ и параметр приращения $м_3$ путем вычисления значений *скалярного-числового-выражения*$_1$, *скалярного-числового-выражения*$_2$ и *скалярного-числового-выражения*$_3$, соответственно, включая, если нужно, преобразование к типу и параметру вида *DO-переменной* по правилам числового преобразования (табл. 7.9). Если *скалярное-числовое-выражение*$_3$ отсутствует, $м_3$ имеет стандартный целый тип и значение 1. Значение $м_3$ не должно быть равным нулю.
2. DO-переменная становится определенной, и ее значение равно значению начального параметра $м_1$.
3. Устанавливается **счетчик числа итераций**, и он равен значению выражения

$$\mathrm{MAX}\ (\mathrm{INT}((м_2 - м_1 + м_3)/м_3),\ 0)$$

Заметим, что число итераций равно нулю, когда

$$м_1 > м_2 \text{ и } м_3 > 0 \text{ или}$$
$$м_1 < м_2 \text{ и } м_3 < 0$$

Если *управление-циклом* опущено, счетчик числа итераций не вычисляется. При этом эффект такой же, как если бы был установлен большой положительный счетчик числа итераций, который невозможно уменьшить до нуля. Если управление циклом имеет форму [,] WHILE (*скалярное-логическое-выражение*), эффект такой же, как если бы *управление-*

циклом было опущено и в качестве первого оператора *блока-DO* был бы вставлен оператор:

IF (.NOT.(*скалярное-логическое-выражение*)) EXIT

После завершения выполнения оператора DO начинается выполнение цикла.

8.1.4.4.2. Выполнение цикла

Выполнение цикла конструкции DO содержит следующие шаги, выполняемые последовательно, повторно и до завершения.

1. Проверяется счетчик числа итераций, если он задан. Если он равен нулю, цикл завершается и конструкция DO становится неактивной. Если *управление-циклом* имеет форму [,] WHILE (*скалярное-логическое-выражение*), вычисляется *скалярное-логическое-выражение* и, если его значение - "ложь", выполнение цикла заканчивается, а конструкция DO становится неактивной. Если в результате все конструкции DO, имеющие *общий-оператор-окончания-цикла*, становятся неактивными, выполнение всех этих конструкций заканчивается. Однако, если среди них остались активные, выполнение продолжается с шага (3) той из активных конструкций DO, оператор DO которой выполнялся последним.
2. Если счетчик числа итераций отличен от нуля, выполняется тело цикла.
3. Счетчик числа итераций, если он есть, уменьшается на единицу. Значение DO-переменной, если она есть, увеличивается на значение параметра приращения $м_3$.

Кроме увеличения значения DO-переменной на шаге (3), значение этой переменной нельзя ни переопределять, ни делать неопределенным, пока конструкция DO активна.

8.1.4.4.3. Оператор CYCLE

Шаг (2) приведенного выше выполнения цикла может быть сокращен путем выполнения оператора CYCLE внутри *тела-цикла*.

R834 *оператор-CYCLE* **есть** CYCLE [*имя-конструкции-DO*]

Ограничение: если *оператор-CYCLE* ссылается на *имя-конструкции-DO*, он должен находиться внутри этой *конструкции-DO*, иначе он должен находиться внутри хотя бы одной какой-нибудь *конструкции-DO*.

Оператор CYCLE принадлежит конкретной конструкции DO, это либо та конструкция, на имя которой он ссылается, либо самая внутренняя из конструкций, в которых он находится.

Выполнение оператора CYCLE вызывает немедленное продвижение на шаг (3) текущего цикла выполнения той конструкции DO, которой он принадлежит. Если эта конструкция DO является неблочной, *оператор-действия-оканчивающий-цикл* или *общий-оператор-окончания-цикла* не выполняется.

В блочной конструкции DO передача управления на *окончание-DO* равнозначна выполнению оператора CYCLE, принадлежащего этой конструкции. В неблочной конструкции передача управления на *оператор-действия-оканчивающий-цикл* или *общий-оператор-окончания-цикла* вызывает их выполнение. Если нет дальнейшей передачи управления, далее выполняется шаг (3) текущего цикла выполнения конструкции DO.

8.1.4.4.4. Завершение цикла

Оператор EXIT - один из способов завершения цикла.

R835 *оператор-EXIT* **есть** EXIT [*имя-конструкции-DO*]

Ограничение: если *оператор-EXIT* ссылается на *имя-конструкции-DO*, он должен находиться внутри этой конструкции, иначе он должен входить в тело хотя бы одной из конструкций DO.

Оператор EXIT принадлежит конкретной конструкции DO. Это либо та конструкция, на имя которой он ссылается, либо самая внутренняя из конструкций, в которых он

находится. Выполнение цикла завершается, когда выполняется одно из следующих условий:

1) при проверке на шаге (1) описанного выше цикла выполнения счетчик числа итераций равен нулю или значение *скалярного-логического-выражения* - "ложь";
2) выполняется оператор EXIT, принадлежащий данной конструкции DO;
3) выполняется оператор EXIT или CYCLE, расположенный внутри данной конструкции DO, но принадлежащий внешней конструкции DO;
4) при передаче управления из оператора, расположенного внутри области конструкции DO, на оператор, который не является окончанием цикла и не находится внутри области той же конструкции DO;
5) внутри области конструкций DO выполняется оператор RETURN;
6) выполняется оператор STOP в каком-либо месте программы либо выполнение оканчивается по какой-то другой причине.

Когда конструкция DO становится неактивной, переменная цикла, если она есть, сохраняет свое последнее значение.

8.1.4.5. Примеры конструкции DO

Ниже приводятся примеры правильных блочных конструкций DO.

Пример 1:

```
DO I = 1, M
    DO J = 1, N
        C (I,J) = SUM (A (I, J, :) * B (:, I, J))
    END DO
END DO
```

В приведенном выше фрагменте программы вычисляется тензорное произведение двух массивов.

Пример 2:

```
READ (IUN, '(1X, G14.7)', IOSTAT = IOS) X
DO WHILE (IOS .NE. 0)
    IF (X .EQ. 0.) THEN
        CALL SUBA (X)
        CALL SUBB (X)
        ...
        CALL SUBZ (X)
    ENDIF
    READ (IUN, '(1X, G14.7)', IOSTAT = IOS) X
END DO
```

Приведенный выше фрагмент программы содержит конструкцию DO, которая использует форму WHILE *управления-циклом*. Цикл будет продолжать выполняться до тех пор, пока не встретится конец файла или не случится ошибка ввода/вывода, тогда в этой точке оператор DO завершит цикл. При считывании отрицательного значения X управление передается на следующий оператор READ, при этом пропускается большая часть области цикла.

Пример 3:

```
DO        ! Цикл "DO WHILE + 1/2"
    READ (IUN, '(1X, G14.7)', IOSTAT =IOS) X
    IF (IOS .NE. 0) EXIT
    IF (X < 0.) CYCLE
    CALL SUBA (X)
    CALL SUBB (X)
    ...
    CALL SUBZ (X)
END DO
```

В примере 3 цикл аналогичен примеру 2. Однако оператор READ перемещен внутрь цикла, так что теперь нужен всего один оператор READ. Кроме того, для того, чтобы избежать дополнительного уровня вложенности IF, используется оператор CYCLE.

Пример 4:

```
  SUM = 0.0
  READ (IUN) N
  OUTER: DO L = 1, N                          ! DO с именем конструкции
    READ (IUN) IQUAL, M, ARRAY (1:M)
    IF(IQUAL<IQUAL_MIN) CYCLE OUTER           ! Пропуск внутреннего цикла
    INNER: DO 40 I = 1, M                     ! DO с меткой и с именем
        CALL CALCULATE(ARRAY (I), RESULT)
        IF (RESULT < 0.0) CYCLE
        SUM = SUM + RESULT
        IF (SUM > SUM_MAX) EXIT OUTER
40  END DO INNER
  END DO OUTER
```

Внешний цикл имеет число итераций MAX(N,0) и будет выполняться либо указанное число раз, либо до тех пор, пока SUM не превысит SUM_MAX, в этом случае оператор EXIT OUTER завершит оба цикла. Если значение признака IQUAL слишком мало, *внутренний цикл пропускается при выполнении первого оператора* CYCLE. Если CALCULATE выдает отрицательный RESULT, второй оператор CYCLE обходит его суммирование. Заметим, что оба цикла имеют имена конструкций, а внутренний цикл еще и содержит метку. Имя конструкции необходимо в операторе EXIT, чтобы завершить оба цикла, а в операторах CYCLE не обязательно, поскольку каждый из них принадлежит своему самому внутреннему циклу.

Пример 5:

```
   N=0
   DO 50 I=1,10
        J=I
        DO K=1,5
             L=K
             N=N+1          ! Этот оператор выполняется 50 раз
        END DO              ! Цикл без метки внутри цикла с меткой
50 CONTINUE
```

После выполнения этого фрагмента I=11, J=10, K=6, L=5, N=50.

Пример 6:

```
      N=0
      DO I=1,10
          J=I
          DO 60 K=5, 1          ! Этот внутренний цикл не выполняется
              L=K
              N=N+1
60        CONTINUE              ! Цикл с меткой внутри цикла без метки
      END DO
```

После выполнения этого программного фрагмента I=11, J=10, K=5, N=0, L - не определяется этими операторами.

Ниже приводятся примеры правильных неблочных конструкций DO.

Пример 7:

```
   DO 70
      READ (IUN, '(1X, G14.7)', IOSTAT = IOS) X
      IF (IOS .NE. 0) EXIT
      F (X < 0.) GOTO 70
      CALL SUBA (X)
      CALL SUBB (X)
      ...
      CALL SUBY (X)
      CYCLE
70    CALL SUBNEG (X)            ! SUBNEG вызывается только при X < 0
```

Эта конструкция DO неблочная, т.к. кончается оператором, отличным от END DO или CONTINUE. Цикл продолжает выполняться до тех пор, пока не встретится конец файла или ошибка ввода/вывода.

Пример 8:

```
   SUM = 0.0
   READ (IUN) N
   DO 80, L = 1, N
      READ (IUN) IQUAL, M, ARRAY (1:M)
      IF (IQUAL < IQUAL_MIN) M = 0              ! Внутренний цикл пропускается
      DO 80 I = 1, M
           CALL CALCULATE (ARRAY (I), RESULT)
           IF (RESULT < 0.) CYCLE
           SUM = SUM + RESULT
           IF (SUM > SUM_MAX) GOTO 81
80    CONTINUE                                  ! Этот CONTINUE - общий для обоих циклов
81 CONTINUE
```

Этот пример аналогичен примеру 4 с той разницей, что два цикла не являются блочными конструкциями DO, т.к. они имеют общий оператор CONTINUE с меткой 80. Конечной конструкцией внешнего цикла является внутренняя конструкция DO целиком. Внутренний цикл пропускается, когда M становится равной 0. Если значение SUM станет слишком большим, оба цикла заканчиваются передачей управления на оператор CONTINUE с меткой 81. Для пропуска отрицательных значений RESULT используется оператор CYCLE во внутреннем цикле.

Пример 9:

```
      N=0
      DO 100 I=1, 10
        J = I
        DO 100 K = 1, 5
            L = K
100         N = N + 1              ! Этот оператор выполняется 50 раз
```

После выполнения этого фрагмента I=11, J=10, K=6, L=5, N=50.

Пример 10:

```
      N = 0
      DO 200 I = 1, 10
```

```
      J = I
      DO 200 K = 5,1        ! Этот внутренний цикл не выполняется никогда
         L = K
200      N = N + 1
```

Этот пример очень похож на предыдущий с той разницей, что внутренний цикл никогда не выполняется. После выполнения этого фрагмента I=11, J=10, K=5, N=0, L не определяется этими операторами.

8.2. Операторы перехода (операторы ветвления)

Операторы перехода (операторы ветвления) используются для изменения нормальной последовательности выполнения. Переход вызывает передачу управления из одного оператора контекстной области на помеченный оператор-адресат в той же контекстной области. **Адресат-оператора-ветвления** может быть одним из следующих операторов: оператор действия, операторы IF THEN, END IF, SELECT CASE, END SELECT, DO, END DO, *оператор-действия-оканчивающий-цикл*, *общий-оператор-окончания-цикла* или *оператор-конструкции-WHERE*.

На оператор END SELECT можно передавать управление только из той же конструкции CASE.

На оператор END IF можно передавать управление из той же конструкции IF и извне.

На *оператор-END-DO* и *оператор-действия-оканчивающий-цикл* можно передавать управление только изнутри той же конструкции DO. На *общий-оператор-окончания-цикла* управление можно передавать только из *внутренней-конструкции-DO-с-общим-окончанием*.

8.2.1. Метки операторов

Метка оператора обеспечивает возможность ссылки на отдельный оператор. Только на операторы-адресаты, операторы FORMAT и окончания DO можно ссылаться, используя метки операторов (3.2.5).

8.2.2. Оператор GO TO

R836 *оператор-GOTO* **есть** GO TO *метка*

Ограничение: *метка* должна быть меткой оператора-адресата, находящегося в той же контекстной области, что и оператор GO TO.

Выполнение оператора GO TO вызывает передачу управления таким образом, что далее выполняется оператор-адресат, помеченный этой меткой.

8.2.3. Вычисляемый оператор GO TO

R837 *вычисляемый-оператор-GOTO* **есть** GO TO (*список-меток*) ♦
♦ [,] *скалярное-целое-выражение*

Ограничение: каждая метка в *списке-меток* должна быть меткой оператора-адресата, находящегося в той же контекстной области, что *вычисляемый-оператор-GOTO*.

Одна и та же метка может появиться несколько раз в списке меток.

Выполнение вычисляемого оператора GO TO вызывает вычисление скалярного целого выражения. Пусть это значение равно *I* и оно такое, что 1 <= *I* <= *N*, где *N* - число меток в *списке-меток*. Тогда передача управления происходит таким образом, что следующим выполняется оператор, помеченный *I*-й меткой из списка меток. Если значение *I* меньше 1 или больше *N*, последовательность выполнения продолжается, как если бы был выполнен оператор CONTINUE.

8.2.4. Оператор ASSIGN и оператор GO TO по предписанию

R838 *оператор-ASSIGN* **есть** ASSIGN *метка* TO *скалярная-целая-переменная*

Ограничение: *метка* должна быть меткой оператора-адресата для оператора перехода или оператора FORMAT, который находится в той же контекстной области, что оператор ASSIGN.

Ограничение: *скалярная-целая-переменная* должна быть именованной переменной стандартного целого типа.

R839 *оператор-GOTO-по-предписанию* **есть** GO TO *скалярная-целая-переменная* [[,] (*список-меток*)]

Ограничение: каждая *метка* из *списка-меток* должна быть меткой оператора-адресата в той же контекстной области, что оператор ASSIGN.

Ограничение: *скалярная-целая-переменная* должна быть именованной переменной стандартного целого типа.

Выполнение оператора ASSIGN приводит к тому, что метка оператора присваивается целой переменной. Пока целая переменная имеет значение метки, на нее можно ссылаться только в контексте оператора GO TO по предписанию или как на спецификатор формата в операторе ввода/вывода. Целая переменная, определенная и имеющая значение метки оператора, может быть переопределена и может получить значение метки оператора или целое значение.

Целая переменная должна быть определена и иметь значение метки оператора FORMAT, когда выполняется оператор ввода/вывода, содержащий ее в качестве идентификатора формата (9.4.1.1).

Во время выполнения оператора GO TO по предписанию целая переменная должна быть определена и иметь значение метки оператора-адресата, который находится в той же контекстной области. Заметим, что переменная может получить значение метки оператора только при помощи оператора ASSIGN в той же контекстной области, что и GO TO по предписанию.

Выполнение оператора GO TO по предписанию вызывает передачу управления таким образом, что далее выполняется оператор-адресат, помеченный меткой оператора, которая в данный момент присвоена целой переменной.

Если есть список в скобках, метка оператора, присвоенная целой переменной, должна присутствовать в этом списке. Метка может появиться в списке меток оператора GO TO по предписанию более чем один раз.

8.2.5. Арифметический оператор IF

R840 *арифметический-оператор-IF* **есть** IF (*скалярное-числовое-выражение*)*метка,метка,метка*

Ограничение: каждая *метка* должна быть меткой оператора-адресата, который находится в той же контекстной области, что и *арифметический-оператор-IF*

Ограничение: *скалярное-числовое-выражение* не может быть комплексным.

Одна и та же метка может появиться в арифметическом операторе IF более одного раза.

Выполнение арифметического оператора IF вызывает вычисление числового выражения с последующей передачей управления. Выполняется оператор-адресат, помеченный первой, второй или третьей меткой, в зависимости от того, значение числового выражения меньше нуля, равно нулю или больше нуля, соответственно.

8.3. Оператор CONTINUE

Выполнение оператора CONTINUE никакого действия не производит.

R841 *оператор-CONTINUE* **есть** CONTINUE

8.4. Оператор STOP

R842 *оператор-STOP* **есть** STOP [*стоп-код*]

R843 *стоп-код* **есть** *скалярная-символьная-константа*
или *цифра* [*цифра* [*цифра* [*цифра*[*цифра*]]]]

Ограничение: *скалярная-целая-константа* должна иметь стандартный целый тип.

Выполнение оператора STOP вызывает окончание выполнения выполняемой программы. Во время завершения *стоп-код*, если он есть, доступен процессорно-зависимым способом. Цифры 0, стоящие в *стоп-коде* впереди, не являются значащими.

8.5. Оператор PAUSE

R844 *оператор-PAUSE* **есть** PAUSE [*стоп-код*]

Выполнение оператора PAUSE вызывает перерыв в выполнении выполняемой программы. Выполнение должно быть возобновимым. Во время приостановки выполнения стоп-код доступен процессорно-зависимым способом. Начальные нули стоп-кода не являются значащими. Возобновление выполнения не управляется программой. Если выполнение возобновлено, последовательность выполнения продолжается так, как если бы был выполнен оператор CONTINUE.

Глава 9. ОПЕРАТОРЫ ВВОДА/ВЫВОДА

Операторы ввода предоставляют средства для передачи данных из внешней среды во внутреннюю память либо из внутреннего файла во внутреннюю память. Этот процесс называется **чтением**. **Операторы вывода** предоставляют средства передачи данных из внутренней памяти во внешнюю среду или во внутренний файл. Этот процесс называется **записью**. Некоторые операторы ввода/вывода специфицируют, что должно быть выполнено редактирование данных.

Помимо операторов передачи данных, есть вспомогательные операторы ввода/вывода для действий с внешней средой, для запроса или для описания характеристик подсоединения ко внешней среде.

Операторы ввода/вывода: OPEN, CLOSE, READ, WRITE, PRINT, BACKSPACE, ENDFILE, REWIND и INQUIRE.

Оператор READ - это **оператор ввода данных**. Операторы WRITE и PRINT - это **операторы вывода данных**. Операторы OPEN и CLOSE - **операторы подсоединения файла**. Оператор INQUIRE - это **оператор запроса о файле**. Операторы BACKSPACE, ENDFILE и REWIND - **операторы установления позиции файла**.

9.1. Записи

Запись - это последовательность значений или последовательность символов. Например строчка на терминале обычно рассматривается как запись. Однако запись не обязательно должна соответствовать какому-то физическому объекту. Есть три вида записей:

1) форматные;
2) неформатные;
3) конец файла.

9.1.1. Форматная запись

Форматная запись состоит из последовательности символов, представимых в процессоре; однако процессор может запретить помещать в форматную запись некоторые управляющие символы (3.1). Длина форматной записи измеряется в символах и зависит, в первую очередь, от числа символов, занесенных в запись в процессе записи. Однако она может зависеть и от процессора, и от внешней среды. Длина может быть нулевой. Форматные записи можно читать и записывать только с помощью операторов форматного ввода/вывода.

Форматные записи можно подготовить каким-либо способом помимо Фортрана, например, с помощью ручного устройства ввода.

9.1.2. Неформатная запись

Неформатная запись состоит из последовательности значений в процессорно-зависимой форме и может содержать данные любого типа, а может и не содержать данных совсем. Длина неформатной записи измеряется в процессорно-зависимых единицах и зависит от списка вывода (9.4.2), используемого во время записи, от процессора и внешней среды. Длина может быть нулевой. Неформатные записи можно читать и записывать только с помощью операторов неформатного ввода/вывода.

9.1.3. Запись "конец файла"

Запись "конец файла" записывается явно с помощью оператора ENDFILE. Файл должен быть подключен для последовательного доступа. Неявно запись "конец файла" заносится в том случае, когда последним обращавшимся к файлу оператором передачи

данных был оператор вывода и после него для этого файла не выполнялся ни один из операторов изменения позиции файла и кроме того:

1) оператор REWIND или BACKSPACE ссылается на устройство, к которому подсоединен этот файл, или
2) устройство (файл) закрывается либо явно, через оператор CLOSE, либо неявно при завершении программы не по причине ошибки или при выполнении другого оператора OPEN для этого же устройства.

Запись "конец файла" может быть в файле только последней. Она не имеет длины.

9.2. Файлы

Файл - это последовательность записей. Есть два вида файлов:

1) внешние и
2) внутренние.

9.2.1. Внешние файлы

Внешний файл - это файл, существующий в среде, внешней по отношению к выполняемой программе.

В любой данный момент времени для файла имеются: **множество разрешенных способов доступа**, зависящее от процессора, **множество разрешенных форм**, зависящее от процессора, **множество разрешенных действий**, зависящее от процессора, и **множество разрешенных длин записей**, зависящее от процессора.

Файл может иметь имя и тогда он называется **именованным файлом**. Имя именованного файла - это строка символов. Множество допустимых имен зависит от процессора.

Внешний файл, подсоединенный к устройству, характеризуется **позицией** (9.2.1.3).

9.2.1.1. Существование файла

В любой данный момент времени есть определенное процессором множество файлов, которые, как говорят, **существуют** для выполняемой программы. Файл может быть известен процессору, но не существовать для выполняемой программы в какое-то конкретное время, например, по причине защиты файла. Файл может существовать и не содержать записей, например, вновь созданный файл, в который еще ничего не записано.

Создать файл - значит вызвать существование ранее не существовавшего файла. **Удалить файл** - значит завершить его существование.

Все операторы ввода/вывода могут ссылаться на файлы, которые существуют. Операторы INQUIRE, OPEN, CLOSE, WRITE, PRINT, REWIND и ENDFILE могут также ссылаться на несуществующие файлы. Выполнение оператора WRITE или PRINT, ссылающегося на предварительно подсоединенный, но не существующий файл, создает файл.

9.2.1.2. Доступ к файлу

Есть два способа доступа к записям внешнего файла: последовательный и прямой. Для некоторых файлов может быть разрешено более одного способа доступа, другие ограничиваются каким-то одним. Например, процессор может разрешить для файлов на магнитной ленте только последовательный доступ. Таким образом, множество разрешенных способов доступа зависит от файла и от процессора.

Способ доступа к файлу определяется при подсоединении файла к устройству (9.3.2) или, в случае предварительного подсоединения, при создании файла (9.3.3).

9.2.1.2.1. Последовательный доступ

Внешний файл, подсоединяемый для **последовательного доступа**, обладает следующими свойствами.

1) Если во множество разрешенных способов доступа для файла не входит еще и прямой доступ, тогда порядок записей в нем - это порядок, в котором они записываются в файл. Если же разрешен и прямой доступ, порядок записей совпадает с тем порядком, который задан для прямого доступа. В этом случае первая запись, доступная при последовательном доступе, это запись с номером 1 при прямом доступе. Вторая запись, доступная при последовательном доступе,-это запись, номер которой равен 2 при прямом доступе и т.д. Запись, которая не записывалась с момента создания файла, не должна читаться.
2) Записи в файле либо все форматные, либо все неформатные. Исключение: последней записью файла может быть запись "конец файла". Если предыдущая ссылка на файл не была оператором вывода или оператором установки позиции файла, последней записью (если она есть) должна быть запись "конец файла".
3) Записи файла нельзя читать или записывать с помощью операторов ввода/вывода прямого доступа.

9.2.1.2.2. Прямой доступ

Файл, подсоединяемый для **прямого доступа**, обладает следующими свойствами.

1) Каждая запись в файле однозначно идентифицируется положительным целым, называемым **номером записи**, который задается при занесении записи в файл. Однажды установленный, номер записи не может никогда измениться. Заметим, что запись нельзя удалить, ее можно только перезаписать. Порядок записей соответствует порядку их номеров.
2) Записи в файле либо все форматные, либо все неформатные. Если среди разрешенных для файла способов доступа есть и последовательный, запись "конец файла", если она есть, не рассматривается как часть файла, пока он подсоединен для прямого доступа. Если же для файла последовательный доступ не входит во множество разрешенных способов доступа, файл не должен содержать запись "конец файла".
3) Чтение и запись записей выполняется только операторами ввода/вывода прямого доступа.
4) Все записи в файле имеют одинаковую длину.
5) Необязательно читать или записывать записи по порядку их номеров; любую запись можно занести в файл, пока он подсоединен к устройству. Например, можно записать запись с номером 3, даже если записи 1 и 2 не были записаны. Любую запись можно прочитать из файла, пока он подсоединен к устройству, при условии, что эта запись была записана с момента создания файла.
6) Записи файла нельзя читать или записывать при помощи форматирования, управляемого списком (10.8), форматирования по NAMELIST (10.9) или оператора ввода/вывода без продвижения.

9.2.1.3. Позиция файла

Выполнение некоторых операторов ввода/вывода влияет на позицию файла. При некоторых обстоятельствах позиция файла становится неопределенной.

Начальная точка файла - это позиция непосредственно перед первой записью. **Конечная точка файла** - это позиция непосредственно после последней

записи. Если в файле нет записей, позиция начальной и конечной точки совпадает. Если позиция файла установлена внутри записи, эта запись является **текущей записью**, в противном случае текущей записи нет.

Предположим, что N - число записей в файле. Если $1 < I <= N$ и позиция файла установлена внутри I-й записи или между (I-1)-й записью и I-й записью, (I-1)-я запись является **предшествующей записью**. Если $N >= 1$ и позиция файла установлена в конечную точку, предшествующей записью является N-я, последняя запись. Если $N = 0$ или если позиция файла установлена в начальной точке или внутри первой записи, предшествующей записи нет.

Если $1 <= I < N$ и позиция файла установлена внутри I-й записи или между I-й и (I+1)-й записями, (I+1)-я запись является **следующей записью**. Если $N >= 1$ и позиция файла установлена в начальную точку, следующей является первая запись. Если $N = 0$ или если позиция файла установлена в конечной точке или внутри последней, N-й записи, следующей записи нет.

9.2.1.3.1. Ввод/вывод с продвижением и без продвижения

Оператор ввода/вывода с продвижением всегда устанавливает позицию файла после последней считанной или записанной записи (если не было ошибок).

Оператор ввода/вывода без продвижения может устанавливать позицию файла на символьную позицию внутри текущей записи. Используя такой оператор ввода/вывода, можно прочитать или записать одну запись с помощью нескольких операторов ввода/вывода, причем каждый из них будет обращаться к части записи. Можно также читать записи переменной длины с регистрацией их длин.

9.2.1.3.2. Позиция файла перед передачей данных

Установка позиции файла перед передачей данных зависит от способа доступа к нему: последовательного и прямого.

Для последовательного доступа при вводе, если есть текущая запись, позиция файла не меняется, иначе позиция файла устанавливается на начало следующей записи, и эта запись становится текущей. Ввод запрещен, если нет следующей записи, либо если есть текущая запись и последний обращавшийся к этому файлу оператор выполнял вывод.

Если в файле есть запись "конец файла", позиция файла перед началом передачи данных не должна быть установлена после нее. Однако для изменения позиции файла в этом случае можно использовать операторы REWIND или BACKSPACE.

Для последовательного доступа при выводе, если есть текущая запись, позиция файла не меняется и текущая запись становится последней записью файла. Иначе создается новая запись как следующая запись файла; эта новая запись становится текущей и последней записью в файле, и позиция файла устанавливается на начало этой записи.

При прямом доступе позиция файла устанавливается в начало записи, определяемой спецификатором записи. Эта запись становится текущей записью.

9.2.1.3.3. Позиция файла после передачи данных

Если возникла ситуация ошибки (9.4.3), позиция файла не определена. Если ситуации ошибки нет, но в результате чтения записи "конец файла" возникла ситуация "конец файла" (9.4.3), позиция файла устанавливается после записи "конец файла".

При вводе без продвижения, если не было ни ситуации ошибки, ни ситуации "конец файла", но есть ситуация "конец записи" (9.4.3), позиция файла устанавливается после только что считанной записи. Если же в операторе ввода без продвижения не было ситуаций ошибки, "конец файла" или "конец записи", позиция файла не изменяется. В операторе вывода без продвижения, если не было ситуации ошибки, позиция файла не

изменяется. Во всех остальных случаях позиция файла устанавливается после последней считанной или записанной записи, так что она становится предшествующей записью.

9.2.2. Внутренние файлы

Внутренние файлы являются средством для преобразования и передачи данных из внутренней памяти во внутреннюю память.

9.2.2.1. *Свойства внутренних файлов*

Внутренние файлы обладают следующими свойствами.

1) Файл - это символьная переменная, но не секция массива с векторным индексом.
2) Запись внутреннего файла - это скалярная символьная переменная.
3) Если файл является скалярной символьной переменной, он состоит из одной записи, длина которой совпадает с длиной скалярной символьной переменной. Если файл является символьным массивом, он рассматривается как последовательность элементов символьного массива. Каждый элемент массива (если он есть) является записью файла. Упорядочение записей в файле совпадает с упорядочением элементов в массиве (6.2.2.2) или в секции массива (6.2.2.3). Длина каждой из записей файла совпадает с длиной элемента массива.
4) Запись внутреннего файла становится определенной при ее записи. Если число занесенных в запись символов меньше, чем длина записи, остальная ее часть заполняется пробелами. Число символов, которые следует записать, не должно быть больше, чем длина записи.
5) Прочитать запись можно, только если она определена.
6) Запись внутреннего файла может быть определена (или лишена определенности) способом, отличным от оператора вывода. Например, символьная величина или элемент символьного массива могут быть определены с помощью символьного оператора присваивания.
7) Перед передачей данных позиция внутреннего файла всегда установлена на начало первой записи. Эта запись становится текущей записью.
8) При вводе пробелы трактуются так же, как для внешнего файла, открытого со спецификатором BLANK, имеющим значение NULL и, если надо, запись дополняется пробелами (9.4.4.4.2).
9) При выводе, управляемом списком, символьные константы выводятся без ограничителей (10.8.2).

9.2.2.2. Ограничения на внутренние файлы

Внутренние файлы имеют следующие ограничения:

1) читать и записывать записи можно только с помощью форматных *операторов ввода/вывода последовательного доступа, которые не* специфицируют форматирования по NAMELIST;
2) внутренние файлы нельзя задавать в операторах подсоединения и установки позиции файлов, в операторе запроса о файле.

9.3. Подсоединение файла

Устройство, специфицируемое *устройством-вв*,-это средство обращения к файлу.

R901 *устройство-вв* **есть** *устройство-внешнего-файла*
или *
или *устройство-внутреннего-файла*

R902 *устройство-внешнего-файла* **есть** *скалярное-целое-выражение*

R903 *устройство-внутреннего-файла* **есть** *переменная-станд- символьного-типа*

Ограничение: *переменная-станд-символьного-типа* не должна быть секцией массива с векторным индексом.

Устройство - это либо внешнее устройство, либо внутреннее устройство. **Внешнее устройство** используется для ссылки на внешний файл и задается либо через *устройство-внешнего-файла*, либо с помощью звездочки. **Внутреннее устройство** используется для ссылки на внутренний файл и задается через *устройство-внутреннего-файла*.

Если символьная переменная, идентифицирующая устройство внутреннего файла, является указателем, он должен быть связан. Если эта переменная - динамически размещаемый массив или его подобъект, массив должен быть размещенным в данный момент.

Скалярное целое выражение, идентифицирующее устройство внешнего файла, должно быть нулевым или положительным.

Устройство-вв в операторах запроса, подсоединения и установки позиции файла должно быть *устройством-внешнего-файла*.

Внешнее устройство, идентифицированное с помощью значения *скалярного-целого-выражения*, является одним и тем же внешним устройством для всех программных единиц выполняемой программы. В примере:

```
SUBROUTINE A
    READ (6) X
    .
    .
    .
SUBROUTINE B
    N=6
    REWIND N
```

значение 6, используемое в обеих программных единицах, идентифицирует одно и то же внешнее устройство.

Звездочка идентифицирует конкретное внешнее устройство, зависящее от процессора и предварительно подсоединенное для форматного последовательного доступа (9.4.4.2).

9.3.1. Существование устройства

В любой данный момент времени есть процессорно-определенное множество внешних устройств, которые, как говорят, существуют для выполняемой программы.

Все операторы ввода/вывода могут ссылаться на существующие устройства. Операторы INQUIRE и CLOSE могут также ссылаться на устройства, которые не существуют.

9.3.2. Подсоединение файла к устройству

Внешнее устройство обладает свойством быть **подсоединенным** или не подсоединенным. Если устройство подсоединено, оно ссылается на внешний файл. Внешнее устройство может быть подсоединено предварительно или путем выполнения оператора OPEN. Свойство подсоединенности является симметричным: если устройство подсоединено к файлу, то файл тоже подсоединен к устройству.

Все операторы ввода/вывода, кроме OPEN, CLOSE и INQUIRE, должны ссылаться на устройство, подсоединенное к файлу, и таким образом использовать этот файл или воздействовать на него.

Файл может быть подсоединен, но не существовать. Примером может служить предварительно подсоединенный файл, в который еще ничего не записывалось (9.2.1.1).

Устройство не должно быть подсоединено одновременно более чем к одному файлу и, наоборот, файл не может быть одновременно подсоединен более чем к одному устройству. Однако есть средства изменить статус внешнего устройства для подсоединения его к другому файлу.

После того, как внешнее устройство было отсоединено от файла при выполнении оператора CLOSE, его можно снова подсоединить в той же выполняемой программе к тому же или другому файлу. После того, как внешний файл был отсоединен при выполнении оператора CLOSE, его можно снова подсоединить в той же выполняемой программе к тому же или другому устройству. Заметим, однако, что ссылаться на отсоединенный файл можно только путем включения его имени в операторы OPEN или INQUIRE. Таким образом, может не оказаться средств для повторного подсоединения однажды отсоединенного неименованного файла.

Внутреннее устройство всегда подсоединено к внутреннему файлу, обозначенному переменной стандартного символьного типа, идентифицирующей устройство.

9.3.3. Предварительное подсоединение

Предварительное подсоединение означает, что устройство подсоединено к файлу в начале выполнения программы и на него, таким образом, можно ссылаться в операторах ввода/вывода без предварительного выполнения оператора OPEN.

9.3.4. Оператор OPEN

Оператор OPEN инициирует и модифицирует связь между внешним файлом и специфицированным устройством. Оператор OPEN можно использовать для подсоединения существующего файла к устройству, для создания предварительно подсоединенного файла, для создания файла и подсоединения его к устройству или для изменения некоторых спецификаторов подсоединения файла к устройству.

Внешнее устройство может быть подсоединено с помощью оператора OPEN в любой программной единице выполняемой программы, и после этого на него можно ссылаться из любой программной единицы выполняемой программы.

Если устройство подсоединено к существующему файлу, для такого устройства разрешено выполнение оператора OPEN. Если в этом операторе отсутствует спецификатор FILE= , подсоединенный к устройству файл - это тот файл, с которым устройство уже соединено.

Если подсоединяемый к устройству файл не существует, но это тот файл, который был подсоединен к нему предварительно, характеристики, задаваемые оператором OPEN, становятся частью подсоединения.

Если же файл, который надо подсоединить к устройству, не тот файл, с которым оно сейчас соединено, действие оператора OPEN таково, как если бы непосредственно перед ним был выполнен оператор CLOSE для этого устройства без спецификатора STATUS= .

Если файл, подсоединяемый к устройству, уже был к нему подсоединен, значения, отличные от действующих, можно задавать только для спецификаторов BLANK= , DELIM= , PAD= , ERR= или IOSTAT= . Выполнение такого оператора OPEN присваивает новые значения спецификаторам BLANK= , DELIM= или PAD= , но не вносит каких-либо изменений в отсутствующие спецификаторы и в позицию файла. Спецификаторы ERR= и IOSTAT= из выполненных ранее операторов OPEN не влияют на выполняемый в данный момент оператор OPEN.

Если файл уже подсоединен к устройству, выполнение оператора OPEN для этого файла и другого устройства не разрешается.

R904 *оператор-OPEN* **есть** OPEN *(список-специф-подсоединения)*

R905 *специф-подсоединения* **есть** [UNIT=] *устройство-внешнего-файла*
или IOSTAT= *скалярная-целая-переменная*

или ERR= *метка*
или FILE= *выражение-имени-файла*
или STATUS= *скалярное-выражение-станд-символьного-типа*
или ACCESS= *скалярное-выражение-станд-символьного-типа*
или FORM= *скалярное-выражение-станд-символьного-типа*
или RECL= *скалярное-целое-выражение*
или BLANK= *скалярное-выражение-станд-символьного-типа*
или POSITION= *скалярное-выражение-станд-символьного-типа*
или ACTION= *скалярное-выражение-станд-символьного-типа*
или DELIM= *скалярное-выражение-станд-символьного-типа*
или PAD= *скалярное-выражение-станд-символьного-типа*

R906 *выражение-имени-файла* **есть** *скалярное-символьное-выражение*

Ограничение: если необязательные символы UNIT= в спецификаторе устройства опущены, он должен быть первым элементом в *списке-специф-подсоединения*.

Ограничение: каждый из спецификаторов не должен появляться в одном *операторе-OPEN* более одного раза; *устройство-внешнего-файла* должно быть задано обязательно.

Ограничение: метка, используемая в спецификаторе ERR=, должна быть меткой адресата оператора ветвления, находящегося в той же контекстной области, что оператор OPEN.

Если спецификатор STATUS= имеет значение OLD, NEW или REPLACE, спецификатор FILE= должен присутствовать. Если же спецификатор STATUS= имеет значение SCRATCH, спецификатор FILE= должен отсутствовать.

Спецификатор, требующий *скалярное-выражение-станд-символьного-типа*, может иметь ограниченный список символьных значений. Эти значения перечислены для каждого спецификатора. Все конечные пробелы игнорируются. Если у процессора есть возможность представления букв как верхнего, так и нижнего регистров, значение задается безотносительно к регистру. Некоторые спецификаторы имеют значения по умолчанию, если спецификатор опущен.

Спецификаторы IOSTAT= и ERR= описаны, соответственно, в 9.4.1.4 и 9.4.1.5.

Пример оператора OPEN:

```
OPEN (10,FILE='employee.names', ACTION='READ', PAD='YES')
```

9.3.4.1. Спецификатор FILE= в операторе OPEN

Значение спецификатора FILE= - это имя файла, который должен быть подсоединен к заданному устройству. Все конечные пробелы игнорируются. *Выражение-имени-файла* должно быть именем, разрешенным процессором. Если спецификатор опущен и устройство не подсоединено к файлу, должен быть задан спецификатор STATUS= со значением SCRATCH; в этом случае производится подсоединение к файлу, определенному процессором. Если процессор имеет возможность представления букв верхнего и нижнего регистров, интерпретация регистра зависит от процессора.

9.3.4.2. Спецификатор STATUS= в операторе OPEN

Скалярное-выражение-станд-символьного-типа должно после вычисления иметь значение OLD, NEW, SCRATCH, REPLACE или UNKNOWN. Если задано OLD, файл должен существовать. Если задано NEW, файл не должен существовать.

Успешное выполнение оператора OPEN с заданным NEW создает файл и изменяет его статус на OLD. Если задано REPLACE и файл еще не существует, создается новый файл с тем же именем, и его статус меняется на OLD. Если же задано REPLACE и файл уже существует, файл удаляется и создается новый файл с тем же именем, а статус его меняется на OLD. Если задано SCRATCH, файл подсоединяется к данному устройству для использования выполняемой программой, но удаляется при выполнении оператора CLOSE, ссылающегося на то же устройство, или при завершении выполняемой программы.

Заметим, что нельзя задавать SCRATCH с именованным файлом. Если задано UNKNOWN, статус зависит от процессора. Если этот спецификатор опущен, предполагается значение UNKNOWN.

9.3.4.3. Спецификатор ACCESS= в операторе OPEN

Скалярное-выражение-станд-символьного-типа должно после вычисления иметь значение SEQUENTIAL или DIRECT. Спецификатор ACCESS задает при подсоединении файла способ доступа: последовательный или прямой. Если этот спецификатор опущен, предполагается значение SEQUENTIAL. Для существующего файла заданный способ доступа должен быть включен в множество разрешенных способов доступа. Для нового файла процессор создает файл с множеством разрешенных способов доступа, включающим заданный способ.

9.3.4.4. Спецификатор FORM= в операторе OPEN

Скалярное-выражение-станд-символьного-типа должно после вычисления иметь значение FORMATTED или UNFORMATTED. Спецификатор FORM= определяет, подсоединяется ли файл для форматного или для неформатного ввода/вывода. Когда этот спецификатор опущен, предполагается значение UNFORMATTED, если файл подсоединяется для прямого доступа, и значение FORMATTED, если файл подсоединяется для последовательного доступа. В случае существующего файла заданный формат должен входить в множество разрешенных форматов для файла. Для нового файла процессор создает файл с множеством разрешенных форматов, включающих заданный.

9.3.4.5. Спецификатор RECL= в операторе OPEN

Значение спецификатора RECL= должно быть положительным. Оно задает длину каждой записи в файле, подсоединяемом для прямого доступа, или максимальную длину записи в файле, подсоединяемом для последовательного доступа. Этот спецификатор должен присутствовать, если файл подсоединяется для прямого доступа. Если спецификатор опущен, когда файл подсоединется для последовательного доступа, значение по умолчанию зависит от процессора. Если файл подсоединяется для форматного ввода/вывода, длина - это число символов для всех записей, содержащих символы только стандартного символьного типа. Если файл подсоединяется для неформатного ввода/вывода, длина измеряется в процессорно-зависимых единицах. Для существующего файла значение спецификатора RECL= должно быть включено в множество разрешенных для файла длин записей. Для нового файла процессор создает файл с множеством разрешенных длин записей, включающим заданное значение.

9.3.4.6. Спецификатор BLANK= в операторе OPEN

Скалярное-выражение-станд-символьного-типа должно после вычисления иметь значение NULL или ZERO. Спецификатор BLANK= разрешен только для файлов, подсоединяемых для форматного ввода/вывода. Если задано NULL, все символы пробела в полях числового ввода на заданном устройстве игнорируются (исключение: поле, состоящее из одних пробелов, имеет нулевое значение). Если задано ZERO, все пробелы, кроме начальных, воспринимаются как нули. Если этот спецификатор опущен, предполагается значение NULL.

9.3.4.7. Спецификатор POSITION= в операторе OPEN

Скалярное-выражение-станд-символьного-типа должно после вычисления иметь значение REWIND, APPEND или ASIS. Подсоединение должно быть для последовательного доступа. Позиция файла, который не существовал предварительно (новый файл, заданный явно или по умолчанию), устанавливается на его начальную точку. REWIND устанавливает позицию существующего файла на его начальную точку. APPEND устанавливает позицию существующего файла таким образом, что следующей становится запись "конец файла", если она есть. Если существующий файл не имеет записи "конец

файла", APPEND устанавливает позицию файла на его конечную точку. ASIS оставляет позицию файла неизменной, если файл существует и уже подсоединен. Если же файл существует, но не подсоединен, ASIS оставляет позицию файла не заданной. Если этот спецификатор опущен, подразумевается значение ASIS.

9.3.4.8. Спецификатор ACTION= в операторе OPEN

Скалярное-выражение-станд-символьного-типа должно после вычисления иметь значение READ, WRITE или READWRITE. READ означает, что на данное подсоединение не могут ссылаться операторы WRITE и ENDFILE. WRITE означает, что на данное подсоединение не может ссылаться оператор READ. READWRITE позволяет любому оператору ввода/вывода ссылаться на данное подсоединение. Если спецификатор опущен, подразумеваемое значение зависит от процессора. Если READWRITE включено в множество разрешенных для файла действий, в это же множество для данного файла должны быть включены READ и WRITE. Для существующего файла заданное действие должно входить в множество разрешенных для файла действий. В случае нового файла процессор создает файл с множеством разрешенных действий, включающим заданное.

9.3.4.9. Спецификатор DELIM= в операторе OPEN

Скалярное-станд-символьное-выражение должно после вычисления иметь значение APOSTROPHE, QUOTE или NONE. Если задано APOSTROPHE, для символьных констант, записываемых с помощью формата, управляемого списком или по NAMELIST, в качестве ограничителей должны использоваться апострофы; все внутренние апострофы должны быть сдвоены. Если задано QUOTE, для символьных констант, записываемых с помощью формата, управляемого списком или по NAMELIST, в качестве ограничителей должны использоваться кавычки; внутренние кавычки должны быть сдвоены. Если заданы APOSTROPHE или QUOTE, перед открывающим ограничителем констант нестандартных символьных типов должны быть использованы символ подчеркивания и *параметр-вида*. Если значением спецификатора является NONE, символьные константы не ограничиваются апострофами или кавычками, а внутренние кавычки или апострофы не сдваиваются. Если спецификатор опущен, значением по умолчанию является NONE. Этот спецификатор разрешен только для файлов, подключаемых для форматного ввода/вывода. Во время ввода форматных записей этот спецификатор игнорируется.

9.3.4.10.Спецификатор PAD= в операторе OPEN

Скалярное-выражение-станд-символьного-типа должно после вычисления иметь значение YES или NO. Если задано YES, когда список ввода и спецификация формата требуют из записи больше данных, чем она содержит, запись форматного ввода дополняется пробелами (9.4.4.4.2). Если задано NO, список ввода и спецификация формата не должны требовать больше символов, чем содержит запись. Если спецификатор опущен, по умолчанию подразумевается YES. Этот спецификатор разрешен только для файлов, подсоединяемых для форматного ввода/вывода. Во время вывода форматных записей он игнорируется. Заметим, что для нестандартных символьных типов символ заполнения пустых позиций зависит от процессора.

9.3.5. Оператор CLOSE

Оператор CLOSE используется для прекращения связи конкретного устройства с внешним файлом.

Выполнение оператора CLOSE, ссылающегося на устройство, может производиться в любой программной единице выполняемой программы, причем необязательно это должна быть та программная единица, в которой выполняется оператор OPEN, ссылающийся на то же устройство.

Выполнение оператора CLOSE, ссылающегося на устройство, которое не существует или к которому не подсоединен ни один файл, разрешено и не оказывает никакого воздействия на файлы.

После того, как устройство было отсоединено посредством выполнения оператора CLOSE, его можно снова подсоединить в той же выполняемой программе к тому же файлу или к другому. После того, как файл был отсоединен при выполнении оператора CLOSE, его можно снова подсоединить в той же выполняемой программе к тому же устройству или к другому при условии, что файл еще существует.

По завершении выполнения программы по причинам, не связанным с ситуацией ошибки, все подсоединенные устройства закрываются со статусом DELETE, если статус файла перед закрытием был SCRATCH, и со статусом KEEP в остальных случаях.

Отметим, что закрытие по окончании программы производится так, как если бы для каждого из подсоединенных устройств был выполнен оператор CLOSE без спецификатора STATUS= .

R907 *оператор-CLOSE* **есть** CLOSE (*список-специф-закрытия*)

R908 *специф-закрытия* **есть** [UNIT=] *устройство-внешнего-файла*
или IOSTAT= *скалярная-целая-переменная*
или ERR = *метка*
или STATUS=*скалярное-выражение-станд-символьного-типа*

Ограничение: если в спецификаторе устройства опущены необязательные символы UNIT= , спецификатор устройстства должен быть первым элементом в *списке-специф-закрытия*.

Ограничение: ни один из спецификаторов не должен входить в один *оператор-CLOSE* более одного раза; спецификатор устройства должен присутствовать всегда.

Ограничение: *метка*, используемая в спецификаторе ERR= , должна быть меткой адресата оператора ветвления, находящегося в той же контекстной области.

Спецификатор, который требует *скалярное-выражение-станд-символьного-типа*, может иметь ограниченный список символьных значений. Конечные пробелы игнорируются. Если процессор имеет возможность представления букв верхнего и нижнего регистров, значение задается без учета регистра.

Спецификаторы IOSTAT= и ERR= описаны, соответственно, в разделах 9.4.1.4 и 9.4.1.5.

Пример оператора CLOSE:

```
CLOSE ( 10, STATUS = 'KEEP' )
```

9.3.5.1. Спецификатор STATUS= в операторе CLOSE

Скалярное-выражение-станд-символьного-типа должно после вычисления иметь значение KEEP или DELETE. Спецификатор STATUS= определяет диспозицию файла, подсоединенного к заданному устройству. Нельзя задавать KEEP для файла, статус которого перед выполнением оператора CLOSE был SCRATCH. Если задано KEEP для существующего файла, он продолжает существовать после выполнения оператора CLOSE. Если задано KEEP для несуществующего файла, он не будет существовать после выполнения оператора CLOSE. Если задано DELETE, файл не будет существовать после выполнения оператора CLOSE.

Когда этот спецификатор опущен, предполагается значение DELETE, если статус файла до выполнения оператора CLOSE был SCRATCH, и значение KEEP в остальных случаях.

9.4. Операторы передачи данных

Оператор READ - это оператор ввода данных. **Оператор WRITE** и **оператор PRINT** - операторы вывода данных.

R909 *оператор-READ* **есть** READ (*список-специф-управления-вв*) ♦
♦ [*список-элементов-ввода*]
или READ *формат* [,*список-элементов-ввода*]

R910 *оператор-WRITE* **есть** WRITE (*список-специф-управления-вв*) ♦
♦ [*список-элементов-вывода*]

R911 *оператор-PRINT* **есть** PRINT *формат* [,*список-элементов-вывода*]

Примеры операторов передачи данных:

```
READ (6, *) SIZE
READ 10,A,B
WRITE (6,10) A,S,J
PRINT 10,A,S,J
10    FORMAT (2E16.3, I5)
```

9.4.1. Список управляющей информации

Список-специф-управления-вв - это **список управляющей информации**, в который входят следующие составляющие:

1) ссылка на источник или место назначения передаваемых данных;
2) необязательная спецификация редактирующих действий;
3) необязательная спецификация для идентификации записи;
4) необязательная спецификация обработки исключительных ситуций;
5) необязательная выдача статуса;
6) необязательная спецификация продвижения записи;
7) необязательная выдача числа считанных символов.

Список управляющей информации управляет передачей данных.

R912 *специф-управления-вв* ♦
♦ **есть** [UNIT=] *устройство-вв*
или [FMT=] *формат*
или [NML=] *имя-группы-NAMELIST*
или REC= *скалярное-целое-выражение*
или IOSTAT= *скалярная-целая-переменная*
или ERR= *метка*
или END= *метка*
или ADVANCE=*скалярное-выражение-станд-символьного-типа*
или SIZE= *скалярное-целое-выражение*
или EOR= *метка*

Ограничение: *список-специф-управления-вв* должен содержать ровно одно *устройство-вв* и может содержать не более чем по одному все остальные спецификаторы.

Ограничение: спецификаторы END=, EOR= и SIZE= не должны входить в *оператор-WRITE*.

Ограничение: *метка*, используемая в спецификаторах ERR=, EOR= и END=, должна быть меткой оператора-адресата для операторов ветвления в той же контекстной области, где находится оператор передачи данных.

Ограничение: если в операторе передачи данных присутствует *список-элементов-ввода* или *список-элементов-вывода*, в нем не должно быть *имени-группы-NAMELIST*.

Ограничение: *список-специф-управления-вв* не должен содержать одновременно и *формат*, и *имя-группы-NAMELIST*.

Ограничение: если в спецификаторе устройства опущены необязательные символы UNIT=, спецификатор устройства должен быть первым элементом в списке управляющей информации.

Ограничение: если в спецификаторе формата опущены необязательные символы FMT=, спецификатор формата должен быть вторым элементом в списке

управляющей информации, а первым элементом в нем должен быть спецификатор устройства без необязательных символов UNIT= .

Ограничение: если в спецификаторе NAMELIST опущены необязательные символы NML= , этот спецификатор должен быть вторым элементом в списке управляющей информации, а первым должен быть спецификатор устройства *без необязательных символов* UNIT= .

Ограничение: если спецификатор устройства специфицирует внутренний файл, в *списке-специф-управления-вв* не должно быть спецификатора REC= или *имени-группы-NAMELIST*.

Ограничение: если есть спецификатор REC=, спецификатор END= и *имя-группы-NAMELIST* не должны появляться, а формат (если он есть) не должен быть звездочкой, задающей ввод/вывод, управляемый списком.

Ограничение: *спецификатор* ADVANCE= *может присутствовать только в форматных* последовательных операторах ввода/вывода с явной спецификацией формата (10.1), список управляющей информации которой не содержит спецификатора внутреннего файла.

Ограничение: если присутствует спецификатор EOR= , должен присутствовать и спецификатор ADVANCE= .

Спецификатор SIZE= может находиться только в операторе ввода, содержащем спецификатор ADVANCE= со значением 'NO'.

Спецификатор EOR= может находиться только в операторе ввода, содержащем спецификатор ADVANCE= со значением 'NO'.

Если оператор передачи данных содержит формат или *имя-группы-NAMELIST*, он является **форматным оператором ввода/вывода**, иначе он является **неформатным оператором ввода/вывода**.

В операторе передачи данных переменная, заданная в спецификаторе (если он есть) IOSTAT= или SIZE= , не должна быть связана ни с каким-либо объектом в списке ввода/вывода (9.4.2) или в *списке-объектов-группы-NAMELIST*, ни с *переменными-цикла* в *неявных-циклах-вв* списка ввода/вывода.

Если переменная, заданная в спецификаторе IOSTAT= или SIZE= оператора передачи данных, является ссылкой на элемент массива, на значения ее индексов не должны оказывать влияние передача данных, обработка *неявного-цикла-вв*, определение или вычисление остальных спецификаторов в *списке-специф-управления-вв*.

Для спецификатора ADVANCE= *скалярная-переменная-станд-символьного-типа* имеет ограниченный список символьных значений. Все конечные пробелы игнорируются. Если процессор может представлять буквы и верхнего, и нижнего регистров, значение задается независимо от регистра.

Пример оператора READ:

```
READ (IOSTAT=IOS, UNIT=6, FMT='(10F8.2)') A, B
```

9.4.1.1. Спецификатор формата

R913 *формат* **есть** *выражение-станд-символьного-типа*
или *метка*
или *
или *скалярная-целая-переменная*

Ограничение: *метка* должна быть меткой оператора FORMAT, расположенного в той же контекстной области, что оператор, содержащий спецификацию формата.

Скалярной-целой-переменной должно быть присвоено (8.2.4) значение метки оператора FORMAT, расположенного в той же контекстной области, что оператор FORMAT.

Выражение-станд-символьного-типа должно после вычисления иметь значение правильной спецификации формата (10.1.1 и 10.1.2.). Заметим, что *выражением-станд-символьного-типа* является и символьная константа.

Если *выражение-станд-символьного-типа* является массивом, это интерпретируется так, как если бы все элементы массива были заданы в стандартном порядке и сцеплены между собой.

Если *формат* задан звездочкой, оператор является **оператором ввода/вывода, управляемого списком.**

Пример формата, заданного символьным выражением:

```
READ (6, FMT="(" // CHAR_FMT // ")" ) X, Y, Z
```

где CHAR_FMT - переменная стандартного символьного типа.

9.4.1.2. Спецификатор NAMELIST

Спецификатор NML= задает *имя-группы-NAMELIST* (5.4). Это имя идентифицирует конкретный набор объектов данных, для которых должна быть выполнена передача.

Если есть *имя-группы-NAMELIST*, оператор является **оператором ввода/вывода по NAMELIST.**

9.4.1.3. Номер записи

Спецификатор REC= задает номер записи, которая должна быть считана или записана. Он может присутствовать только в операторе ввода/вывода, задающем устройство, подсоединенное для прямого доступа. Если список управляющей информации содержит спецификатор REC= , оператор является **оператором ввода/вывода прямого доступа**. В противном случае это **оператор ввода/вывода последовательного доступа**.

9.4.1.4. Статус ввода/вывода

Выполнение оператора ввода/вывода, содержащего спецификатор IOSTAT= , вызывает присваивание специфицированной в нем переменной одного из следующих значений:

1) нулевое значение, если не случилось ни ситуации "конец файла", ни ситуации "конец записи" и нет ситуации ошибки;
2) положительное целое значение, зависящее от процессора, если случилась ситуация ошибки;
3) отрицательное целое значение, зависящее от процессора, если нет ситуации ошибки, но есть ситуация "конец файла";
4) отрицательное целое значение, зависящее от процессора и отличное от значения "конец файла", если есть ситуация конца-записи, но нет ситуации "конец файла".

Заметим, что ситуация "конец файла" может возникнуть только во время исполнения оператора последовательного ввода, а ситуация "конец записи" - только во время выполнения оператора ввода без продвижения.

Рассмотрим пример:

```
READ (FMT="(E8.3)", UNIT=3, IOSTAT=IOSS) X
 !
IF (IOSS<0) THEN
      !   Выполнение обработки ситуации "конец файла" для файла,
      !   подсоединенного к устройству 3.
      CALL END_PROCESSING
      !
ELSE IF(IOSS>0) THEN
```

```
      !
      ! Выполнение обработки ошибки
CALL ERROR_PROCESSING
      !
END IF
```

9.4.1.5. Ветвь ошибки

Если оператор ввода/вывода содержит спецификатор ERR= и во время его выполнения процессор обнаруживает ситуацию ошибки (9.4.3), происходит следующее:

1) выполнение оператора ввода/вывода прекращается;
2) позиция заданного оператором ввода/вывода файла становится неопределенной;
3) если в операторе ввода/вывода есть и спецификатор IOSTAT= , переменная, специфицированная в нем, определяется и ей присваивается положительное целое значение, зависящее от процессора;
4) если оператор - это оператор READ со спецификатором SIZE= , переменная определяется и получает целое значение (9.4.1.9);
5) выполнение продолжается с оператора, заданного спецификатором ERR= .

9.4.1.6. Ветвь "конец файла"

Если оператор ввода содержит спецификатор END= и во время выполнения оператора возникла ситуация "конец файла" (9.4.3) при отсутствии ситуации ошибки (9.4.3), происходит следующее:

1) выполнение оператора READ завершается,
2) если заданный в операторе ввода файл является внешним файлом, его позиция устанавливается после записи "конец файла";
3) если оператор содержит еще и спецификатор IOSTAT=, специфицированная в нем переменная определяется и ей присваивается отрицательное целое значение, зависящее от процессора;
4) исполнение продолжается с оператора, заданного спецификатором END= .

В операторе WRITE список управляющей информации не должен включать спецификатор END= .

9.4.1.7. Ветвь "конец записи"

Если оператор ввода содержит спецификатор EOR= и во время исполнения оператора возникла ситуация "конец записи" (9.4.3), но при этом не было ситуации ошибки, происходит следующее:

1) если спецификатор PAD= имеет значение 'YES', запись заполняется пробелами так, чтобы было соответствие между элементом списка ввода (9.4.4.4.2) и описателем редактирования данных, который требует больше символов, чем содержится в записи,
2) выполнение оператора ввода прекращается,
3) позиция файла, указанного в операторе, устанавливается после текущей записи;
4) если оператор ввода содержит и спецификатор IOSTAT= , переменная определяется и ей присваивается отрицательное целое значение, зависящее от процессора,
5) если оператор ввода/вывода содержит спецификатор SIZE= , переменная определяется и получает целое значение (9.4.1.9),
6) исполнение продолжается с оператора, заданного спецификатором EOR= .

Список управляющей информации оператора WRITE не может содержать спецификатор EOR= .

9.4.1.8. Спецификатор продвижения

Скалярное-выражение-станд-символьного-типа должно после вычисления иметь значение YES или NO. Спецификатор ADVANCE= определяет, выполняется ли для этого оператора ввод/вывод с продвижением. Если специфицировано NO, выполняется ввод/вывод без продвижения. Если специфицировано YES, имеет место форматный последовательный ввод/вывод с продвижением. Если этот спецификатор опущен, подразумевается значение YES.

9.4.1.9. Число символов

Когда заканчивается оператор ввода без продвижения, переменная, указанная в спецификаторе SIZE= , определяется и получает значение, равное числу символов, переданных описателями редактирования данных во время выполнения текущего оператора ввода. Пробелы, вставляемые для заполнения, при этом не учитываются.

9.4.2. Список ввода/вывода для передачи данных

Список ввода/вывода специфицирует объекты, значения которых передаются при помощи операторов ввода/вывода для передачи данных.

R914 *элемент-ввода* — **есть** *переменная*
или *неявный-цикл-вв*

R915 *элемент-вывода* — **есть** *выражение*
или *неявный-цикл-вв*

R916 *неявный-цикл-вв* — **есть** (*список-объектов-нцкл-вв, управление-нцкл-вв*)

R917 *объект-нцкл-вв* — **есть** *элемент-ввода*
или *элемент-вывода*

R918 *управление-нцкл-вв* — **есть** *переменная-цикла=*
♦ *скалярное-числовое-выражение,* ♦
♦ *скалярное-числовое-выражение* ♦
♦ [,*скалярное-числовое-выражение*]

Ограничение: переменная, которая является *элементом-ввода*, не должна быть массивом неопределенного размера.

Ограничение: *переменная-цикла* должна быть скаляром целого, стандартного вещественного или вещественного двойной точности типа.

Ограничение: все *скалярные-числовые-выражения* в *управлении-нцкл-вв* должны быть целого, стандартного вещественного или вещественного двойной точности типа.

Ограничение: в *списке-элементов-ввода объект-нцкл-вв* должен быть *элементом-ввода*. В *списке-элементов-вывода объект-нцкл-вв* должен быть *элементом-вывода*.

Элемент-ввода не может быть ни связан, ни использован в качестве *переменной-цикла* для того *неявного-цикла-вв*, в котором он содержится.

Если элемент ввода - указатель, он должен быть в данный момент связан с определимым адресатом, и данные передаются из файла этому адресату. Если элемент вывода - указатель, он должен быть в данный момент связан с адресатом и значение данного передается от адресата в файл.

Если элемент ввода или элемент вывода - динамически размещаемый массив, он должен быть в данный момент размещен.

Переменная-цикла неявного-цикла-вв, вложенного в другой *неявный-цикл-вв*, не может быть использована в качестве (или связана с) *переменной-цикла* объемлющего *неявного-цикла-вв*.

Если элементом списка ввода/вывода является массив, это воспринимается так, как если бы все элементы массива, если они есть, были перечислены в обычном для массива порядке (6.2.2.2). Однако элемент этого массива не может влиять на значение какого-либо выражения в *элементе-ввода* и ни один элемент не может появиться дважды в *элементе-ввода*. Например:

```
INTEGER A(100), J(100)
     . . .
READ *, A(A)                                  ! Запрещено
READ *, A(LBOUND(A,1):UBOUND(A,1))            ! Разрешено
READ *, A(J)                                  ! Разрешено при условии,
                                              !  что в J нет элементов
                                              !  с одинаковым значением
READ *, A(A(1) : A(10))                       ! Запрещено
```

Объект производного типа не должен быть элементом списка ввода/вывода, если какой-либо из конечных компонентов, образующих такой объект, не доступен в контекстной области, содержащей оператор ввода/вывода. В качестве примера можно назвать структуру, взятую из модуля, в котором ее спецификация доступности - PUBLIC, а спецификация доступности ее компонентов - PRIVATE.

Если производный тип содержит конечный компонент-указатель, объект такого типа не может быть ни элементом ввода, ни результатом вычисления списка вывода.

Если объект производного типа входит в качестве элемента списка ввода/вывода в форматный оператор ввода/вывода, это воспринимается, как если бы все его компоненты были заданы в порядке их вхождения в описание производного типа.

Элемент списка ввода/вывода производного типа в неформатном операторе ввода/вывода воспринимается как одно значение в форме, зависящей от процессора. Заметим, что в этом случае вхождение объекта производного типа в качестве элемента списка ввода/вывода не эквивалентно списку его компонентов.

Неявный цикл инициируется и выполняется так же, как конструкция DO (8.1.4.4).

Если оператор ввода/вывода задает внутренний файл, его список ввода/вывода не может содержать элементов нестандартного символьного типа.

Отметим, что константа, выражение, включающее операции или ссылки на функции, выражение, заключенное в скобки, могут быть элементами списка вывода, но не могут быть элементами списка ввода.

Пример списка вывода с неявным циклом:

```
WRITE (LP, FMT='(10F8.2)') (LOG (A(I)), I=1, N+9, K), G
```

9.4.3. Ситуации ошибки, "конец записи" и "конец файла"

Множество ситуаций ошибки зависит от процессора.

Ситуация "конец записи" возникает, когда оператор ввода без продвижения пытается передать данные из позиции, находящейся после конца текущей записи.

Ситуация "конец файла" возникает в любом из следующих случаев:

1) во время чтения из файла, подсоединенного для последовательного доступа, встретилась запись "конец файла";
2) произведена попытка считать запись, расположенную после конца внутреннего файла.

Ситуация "конец файла" может возникнуть в начале выполнения оператора ввода. Она же может возникнуть во время выполнения оператора форматного ввода, когда в результате взаимодействия списка ввода и формата может потребоваться более чем одна запись.

Если во время выполнения оператора ввода/вывода возникла ситуация ошибки или "конец файла", выполнение оператора ввода/вывода прекращается и становятся неопределенными переменные всех его неявных циклов. Если это ситуация ошибки, позиция файла становится неопределенной.

Если ситуация ошибки или "конец файла" возникла при вводе, все элементы списка ввода становятся неопределенными.

Если ситуация "конец записи" возникла во время выполнения оператора ввода без продвижения, происходит следующее: если спецификатор PAD= имеет значение YES, запись заполняется пробелами (9.4.4.4.2) в соответствии с элементом списка ввода и описателем редактирования данных, который требует больше символов, чем их содержит запись; выполнение оператора ввода прекращается и все переменные неявных циклов становятся неопределенными, а позиция файла, специфицированного в операторе ввода, устанавливается после текущей записи.

Выполнение программы прекращается в следующих случаях:

если ситуация ошибки возникла во время выполнения оператора ввода/вывода, не содержащего ни спецификатора IOSTAT= , ни спецификатора ERR= ;

если ситуация "конец файла" возникла при исполнении оператора ввода, не содержащего спецификаторов END= и IOSTAT= ;

если ситуация "конец записи" возникла при выполнении оператора READ без продвижения, не содержащего спецификаторов IOSTAT= и EOR= .

9.4.4. Выполнение операторов ввода/вывода для передачи данных

Выполнение оператора передачи данных должно быть таким, как если бы были выполнены следующие действия и именно в таком порядке:

1) определение направления передачи данных;
2) идентификация устройства;
3) установление формата, если он задан;
4) установка позиции файла перед передачей данных (9.2.1.3.2);
5) обмен данными между файлом и объектами списка ввода/вывода (если он есть) или списка NAMELIST;
6) выявление ситуации ошибки, "конец файла" или "конец записи";
7) установка позиции файла после передачи данных (9.2.1.3.3);
8) определение значений переменных в спецификаторах IOSTAT= и SIZE= .

9.4.4.1. Направление передачи данных

Выполнение оператора READ вызывает передачу значений из файла объектам, заданным в списке ввода, если он есть, или заданным в самом файле для ввода по NAMELIST. Выполнение операторов WRITE и PRINT вызывает передачу в файл значений объектов, заданных списком вывода и спецификацией формата, если они есть, или *именем-группы-NAMELIST* для вывода по NAMELIST. Выполнение операторов WRITE и PRINT для несуществующего файла создает файл (если не возникнет ситуация ошибки).

9.4.4.2. Идентификация устройства

Оператор ввода/вывода данных, в котором есть список управления вводом/выводом, содержит спецификатор устройства файла, который идентифицирует внешний или внутренний файл. Оператор READ, в котором отсутствует список управления вводом/выводом, специфицирует конкретное устройство, определенное процессором, то же самое, которое определяется звездочкой в операторе READ, содержащем список управления вводом/выводом. Оператор PRINT специфицирует другое устройство, определенное процессором, его же специфицирует звездочка в операторе WRITE. Таким образом, каждый оператор ввода/вывода идентифицирует внешнее устройство или внутренний файл.

Устройство, идентифицированное оператором передачи данных, должно быть подсоединено к файлу в момент начала исполнения оператора. Заметим, что это может быть предварительное подсоединение.

9.4.4.3. Установление формата

Если список управляющей информации содержит в качестве формата звездочку, устанавливается форматирование, управляемое списком. Если есть *имя-группы-NAMELIST*, устанавливается форматирование по NAMELIST. Если не специфицированы ни формат, ни *имя-группы-NAMELIST*, устанавливается неформатная передача данных. Во всех остальных случаях устанавливается спецификация формата, идентифицированная спецификатором формата. Если форматом является массив, эффект такой, как если бы все элементы массива были сцеплены в обычном для массива порядке.

Если при выводе указан внутренний файл, не разрешается, чтобы спецификация формата находилась в файле или была с ним связана.

9.4.4.4. Передача данных

Данные передаются между записями и объектами, специфицированными в списке ввода/вывода или в списке NAMELIST. Для всех операторов ввода/вывода для передачи данных (кроме операторов ввода по NAMELIST) элементы списка обрабатываются в порядке их вхождения в список ввода/вывода. Элемент списка, который должен обрабатываться следующим, называется **следующим действующим элементом**. Массивы нулевого размера, списки неявных циклов с нулевым счетчиком числа итераций при определении следующего действующего элемента игнорируются. Символьный элемент нулевой длины трактуется как действующий. Элементы списка для форматных операторов ввода по NAMELIST обрабатываются в порядке расположения объектов в записях ввода. Элементы списка для операторов вывода по NAMELIST обрабатываются в порядке задания объектов данных (переменных) в *списке-объектов-группы-NAMELIST*.

Все значения, необходимые для определения заданного элемента списка ввода/вывода, определяются в начале обработки этого элемента.

Все значения передаются в объекты, заданные элементом списка ввода/вывода (или в обратном направлении), до обработки любого последующего элемента списка во всех операторах передачи данных. В примере:

```
READ (N),N,X(N)
```

старое значение N идентифицирует устройство, а новое значение N является индексом X.

Для ввода по NAMELIST все значения, следующие за частью *имя-* объекта NAMELIST в записи ввода (10.9), передаются в соответствующий объект, заданный элементом *списка-объектов-группы-NAMELIST*, до обработки любого последующего элемента в записи ввода. Если во время выполнения оператора ввода по NAMELIST объект указан в записи ввода более чем один раз, последнее его вхождение будет определять значение (значения), используемое (используемые) для этого объекта.

Элемент списка ввода или связанный с ним объект не должны содержать какой-либо части установленной спецификации формата.

Если элементом списка ввода/вывода является указатель, производится передача данных между файлом и связанным с указателем адресатом.

Если специфицирован внутренний файл, элемент списка ввода/вывода не должен находиться в файле или быть с ним связан. Заметим, что файл является объектом данных.

В начале обработки элементов, образующих диапазон *неявного-цикла-вв*, определяется переменная цикла и устанавливается счетчик итераций.

При выводе каждый объект, значение которого передается, должен быть определен.

9.4.4.4.1. Неформатная передача данных

Во время неформатной передачи данных данные передаются без редактирования из текущей записи объектам, специфицированным списком ввода/вывода, или в обратном направлении. Считывается или записывается ровно одна запись.

Неформатными операторами ввода/вывода могут передаваться объекты встроенных и производных типов.

При вводе позиция файла должна быть установлена так, чтобы читалась неформатная запись или запись "конец файла". Число значений, требуемых списком ввода, должно быть меньше или равно числу значений в записи. Каждое значение в записи должно быть того же типа, что соответствующий объект из списка ввода, кроме случаев, когда один комплексный объект может соответствовать двум вещественным объектам или два вещественных значения могут соответствовать одному комплексному объекту. Заметим, что если объект в списке ввода имеет тип символьный, он должен иметь ту же длину и тот же параметр вида, что у символьного значения. Заметим также, что если два вещественных значения соответствуют одному комплексному значению или один комплексный объект соответствует двум вещественным объектам, все три объекта должны иметь одинаковое значение параметра вида.

При выводе в файл, подсоединенный для прямого доступа, список вывода не может задавать больше значений, чем помещается в запись. Если при выводе в файл, подсоединенный для прямого доступа, значения, заданные списком вывода, не заполняют запись полностью, остаток ее будет неопределенным.

Если файл подсоединен для последовательного доступа, создается запись, длина которой достаточна для помещения всех значений из списка вывода. Эта длина должна входить во множество разрешенных для файла длин записи и не должна превышать значения, заданного спецификатором RECL= (если он есть) оператора OPEN, установившего подсоединение.

Если файл подсоединен для форматного ввода/вывода, неформатная передача данных запрещена.

Специфицированное устройство должно быть внешним устройством.

9.4.4.4.2. Форматная передача данных

Во время форматной передачи данных обмен между файлом и объектами, заданными списком ввода/вывода или *именем-группы-NAMELIST*, выполняется с редактированием. Инициируется управление форматом и выполняется редактирование (см. главу 10). Читается или записывается текущая запись и, возможно, дополнительные записи.

С помощью форматных операторов передачи данных значения могут быть переданы в объекты встроенных или производных типов (и в обратном направлении). В случае производного типа передаются значения встроенных типов конечным компонентам (из компонентов) производного типа, из которых, в конце концов, и образуются эти структурированные объекты.

При вводе позиция файла должна быть установлена так, чтобы читалась либо форматная запись, либо запись "конец файла".

Если файл подсоединен для неформатного ввода/вывода, форматная передача данных запрещена.

Во время ввода с продвижением из файла, у которого спецификатор PAD= имеет значение NO, список ввода и спецификация формата не должны требовать символов больше, чем содержится в записи.

Во время ввода с продвижением из файла, у которого спецификатор PAD= имеет значение YES, или во время ввода из внутреннего файла процессор добавляет символы пробела, если список ввода и спецификация формата требуют больше символов, чем содержит запись.

Во время ввода без продвижения из файла, у которого спецификатор PAD= имеет значение NO, если список ввода и спецификация формата требуют больше символов, чем содержит запись, возникает ситуация "конец записи" (9.4.3).

Если во время ввода без продвижения из файла, у которого спецификатор PAD= имеет значение YES, элемент списка ввода и соответствующий описатель редактирования

данных требуют больше символов, чем содержит запись, возникает ситуация "конец записи" и процессором добавляются пробелы.

Если файл подсоединен для прямого доступа, номер записи увеличивается на единицу, когда читается или записывается каждая следующая запись.

Если при выводе указан файл, подсоединенный для прямого доступа, или внутренний файл и символы, заданные списком вывода и форматом, не заполняют запись, она дополняется символами пробела.

При выводе список вывода и спецификация формата не должны задавать символов больше, чем специфицировано спецификатором RECL= в операторе OPEN или длиной записи во внутреннем файле.

9.4.4.5. Форматирование, управляемое списком

Если установлено форматирование, управляемое списком, редактирование выполняется так, как это описано в 10.8.

9.4.4.6. Форматирование по NAMELIST

Если установлено форматирование по NAMELIST, редактирование выполняется так, как описано в 10.9.

9.4.5. Печать форматных записей

Передача информации в форматных записях на конкретные устройства, определенные процессором, называется **печатью**. При печати форматной записи первый ее символ не печатается. Остальные символы записи, если они есть, печатаются в одну строку, начиная с левого поля.

Первый символ такой записи должен относиться к стандартному символьному типу и должен определять число пропущенных при печати строк следующим образом:

Символ	Пропуски строк перед печатью
Пробел	Одна строка
0	Две строки
1	До первой строки следующей страницы
+	Нет продвижения

Если в записи нет символов, пропускается одна строка, состоящая только из пробелов.

Оператор PRINT не подразумевает обязательной печати, а оператор WRITE не означает, что печать не будет выполняться.

9.4.6. Завершение операторов передачи данных

Завершение операторов ввода/вывода для передачи данных происходит при создании любой из следующих ситуаций:

1) при обработке формата обнаруживается описатель редактирования данных, а в *списке-элементов-ввода* или *списке-элементов-вывода* уже не осталось элементов;
2) при неформатной передаче данных или передаче данных, управляемой списком, исчерпан *список-элементов-ввода* или *список-элементов-вывода*;
3) при выводе по NAMELIST исчерпан *список-объектов-группы-NAMELIST* или ввод по NAMELIST достиг "конца записи" после обработки подпоследовательностей имя-значение для каждого элемента в *списке-объектов-группы-NAMELIST*;
4) возникла ситуация ошибки;
5) возникла ситуация "конец файла";

6) в записи, читаемой при вводе под управлением списка или по NAMELIST, встретился в качестве разделителя значений (10.8, 10.9) символ "/";

7) во время исполнения оператора ввода/вывода без продвижения возникла ситуация "конец записи".

9.5. Операторы установки позиции файла

R919 *оператор-BACKSPACE* **есть** BACKSPACE *устройство-внешнего-файла*
или BACKSPACE (*список-специф-позиции*)

R920 *оператор-ENDFILE* **есть** ENDFILE *устройство-внешнего-файла*
или ENDFILE (*список-специф-позиции*)

R921 *оператор-REWIND* **есть** REWIND *устройство-внешнего-файла*
или REWIND (*список-специф-позиции*)

В операторах BACKSPACE, ENDFILE и REWIND нельзя ссылаться на файл, не подсоединенный для последовательного доступа. На файл, подсоединенный со значением READ спецификатора ACTION=, нельзя ссылаться в операторе ENDFILE.

R922 *специф-позиции* **есть** [UNIT=] *устройство-внешнего-файла*
или IOSTAT= *скалярная-целая-переменная*
или ERR= *метка*

Ограничение: *метка* в спецификаторе ERR= должна быть меткой адресата оператора ветвления, находящегося в той же контекстной области, что оператор установки позиции файла.

Ограничение: если в спецификаторе устройства опущены необязательные символы UNIT=, спецификатор устройства должен быть первым элементом *списка-специф-позиции*.

Ограничение: *список-специф-позиции* должен содержать ровно одно *устройство-внешнего-файла* и может содержать не более чем по одному остальные спецификаторы.

Спецификаторы IOSTAT= и ERR= описаны соответственно в 9.4.1.4 и 9.4.1.5.

9.5.1. Оператор BACKSPACE

Выполнение оператора BACKSPACE приводит к тому, что позиция файла, подсоединенного к заданному устройству, устанавливается перед текущей записью, если она есть, либо перед предшествующей, если нет текущей. Если же нет ни текущей записи, ни предшествующей, позиция файла не меняется. Заметим, что если предшествующей является запись "конец файла", позиция файла устанавливается перед ней. Если оператор BACKSPACE вызывает неявный вывод записи "конец файла", позиция файла устанавливается перед записью, которая ей предшествует.

Сдвиг назад для подсоединенного, но не существующего файла запрещен. Запрещен сдвиг назад через записи, записанные с помощью форматирования, управляемого списком или по NAMELIST.

Пример оператора BACKSPACE:

```
BACKSPACE (10, ERR=20)
```

9.5.2. Оператор ENDFILE

Выполнение оператора ENDFILE приводит к тому, что запись "конец файла" становится следующей записью в файле. Затем позиция файла устанавливается после записи "конец файла". Если файл может быть подсоединен и для прямого доступа, записанными считаются только те записи, которые расположены до записи "конец файла". Таким образом, только эти записи можно будет считать при последующем подсоединении файла для прямого доступа.

После выполнения оператора ENDFILE должен быть выполнен оператор BACKSPACE или REWIND для переустановки позиции файла прежде, чем будут выполняться какие-либо операторы передачи данных или оператор ENDFILE.

Выполнение оператора ENDFILE для файла, который подсоединен, но не существует, создает файл и заносит в него запись "конец файла".

Пример оператора ENDFILE:

```
ENDFILE K
```

9.5.3. Оператор REWIND

Выполнение оператора REWIND приводит к установке позиции заданного файла в его начальную точку. Заметим, что если она уже установлена там, выполнение оператора REWIND на позицию файла не влияет.

Выполнение оператора REWIND для файла, который подсоединен, но не существует, разрешено, но никакого действия не производит.

Пример оператора REWIND:

```
REWIND 10
```

9.6. Запрос о файле

Оператор INQUIRE может быть использован для запроса о характеристиках либо конкретного именованного файла, либо подсоединения к конкретному устройству. Имеется три формы оператора INQUIRE: **запрос о файле**, который использует спецификатор FILE= , **запрос об устройстве**, который использует спецификатор UNIT= , и **запрос о списке вывода**, который использует только спецификатор IOLENGTH= . Все присваивания значений спецификаторам производятся в соответствии с правилами операторов присваивания.

Оператор INQUIRE можно выполнять до, во время или после того, как файл был подсоединен к устройству. Все значения, присваиваемые оператором INQUIRE, являются текущими в момент выполнения оператора.

R923 *оператор-INQUIRE* **есть** INQUIRE (*список-специф-запроса*)
или INQUIRE (IOLENGTH= ♦
♦ *скалярная-переменная-станд-целого-типа*)*список-элементов-вывода*

Примеры оператора INQUIRE:

```
INQUIRE (IOLENGTH=IOL) A(1:N)
INQUIRE (UNIT=JOAN, OPENED=LOG_01, NAMED=LOG_02,                  &
                        FORM=CHAR_VAR, IOSTAT=IOS)
```

9.6.1. Спецификаторы запроса

Запрос о файле или об устройстве формируется (с учетом ограничений) из следующих спецификаторов оператора INQUIRE:

R924 *специф-запроса* ♦

♦ **есть** [UNIT=] *устройство-внешнего-файла*
или FILE = *выражение-имени-файла*
или IOSTAT = *скалярная-переменная-станд-целого-типа*
или ERR = *метка*
или EXIST = *скалярная-переменная-станд-логического-типа*
или OPENED = *скалярная-переменная-станд-логического-типа*
или NUMBER = *скалярная-переменная-станд-целого-типа*
или NAMED = *скалярная-переменная-станд-логического-типа*
или NAME = *скалярная-переменная-станд-символьного-типа*
или ACCESS = *скалярная-переменная-станд-символьного-типа*
или SEQUENTIAL= *скалярная-переменная-станд-символьного-типа*
или DIRECT = *скалярная-переменная-станд-символьного-типа*

или FORM = *скалярная-переменная-станд-символьного-типа*
или FORMATTED= *скалярная-переменная-станд-символьного-типа*
или UNFORMATTED=*скалярная-переменная-станд-символьного-типа*
или RECL = *скалярная-переменная-станд-целого-типа*
или NEXTREC = *скалярная-переменная-станд-целого-типа*
или BLANK = *скалярная-переменная-станд-символьного-типа*
или POSITION = *скалярная-переменная-станд-символьного-типа*
или ACTION = *скалярная-переменная-станд-символьного-типа*
или READ = *скалярная-переменная-станд-символьного-типа*
или WRITE = *скалярная-переменная-станд-символьного-типа*
или READWRITE= *скалярная-переменная-станд-символьного-типа*
или DELIM = *скалярная-переменная-станд-символьного-типа*
или PAD = *скалярная-переменная-станд-символьного-типа*

Ограничение: *список-специф-запроса* должен содержать либо один спецификатор FILE= , либо один спецификатор UNIT= (но не оба одновременно) и не более чем по одному остальные спецификаторы.

Ограничение: если в операторе INQUIRE, заданном в форме запроса об устройстве, опущены необязательные символы UNIT= в спецификаторе устройства, он должен быть первым элементом в *списке-специф-запроса*.

Если выдаваемое в спецификатор значение имеет символьный тип (кроме спецификатора NAME=), а процессор располагает возможностью представления букв как верхнего, так и нижнего регистров, значение выдается на верхнем регистре.

Если во время выполнения оператора INQUIRE возникла ситуация ошибки, все переменные в спецификаторах запроса становятся неопределенными, кроме спецификатора IOSTAT= (если он есть).

Спецификаторы IOSTAT= и ERR= описаны, соответственно, в 9.4.1.4 и 9.4.1.5.

9.6.1.1 Спецификатор FILE= в операторе INQUIRE

Значение *выражения-имени-файла* в спецификаторе FILE= задает имя файла, относительно которого производится запрос. Именованный файл не обязательно должен существовать или быть подсоединен к устройству. Значение *выражения-имени-файла* должно быть в виде, допустимом для процессора в качестве имени файла. Все конечные пробелы игнорируются. Если в процессоре есть возможность представления букв верхнего и нижнего регистров, интерпретация выбора зависит от процессора.

9.6.1.2. Спецификатор EXIST= в операторе INQUIRE

Выполнение оператора INQUIRE для файла приводит к тому, что *скалярной-переменной-стандартного-логического-типа* в спецификаторе EXIST= присваивается значение "истина", если файл с заданным именем существует, или значение "ложь" - в противном случае. Выполнение оператора INQUIRE для устройства приводит к тому, что присваивается значение "истина", если заданное устройство существует, в противном случае присваивается значение "ложь".

9.6.1.3. Спецификатор OPENED= в операторе INQUIRE

В результате выполнения оператора INQUIRE для файла *скалярной-переменной-стандартного-логического-типа* в спецификаторе OPENED= присваивается значение "истина", если заданный файл подсоединен к устройству, или значение "ложь" - в противном случае. Выполнение оператора INQUIRE для устройства приводит к присваиванию *скалярной-переменной-стандартного-логического-типа* значения "истина", если указанное устройство подсоединено к файлу, или значения "ложь" в противном случае.

9.6.1.4. Спецификатор NUMBER= в операторе INQUIRE

Скалярной-переменной-стандартного-целого-типа в спецификаторе NUMBER= присваивается значение идентификатора внешнего устройства, подсоединенного в данный момент к файлу. Если такого устройства нет, присваивается значение -1.

9.6.1.5. Спецификатор NAMED= в операторе INQUIRE

Скалярной-переменной-стандартного-логического-типа в спецификаторе NAMED= присваивается значение "истина", если файл имеет имя, или значение "ложь" - в противном случае.

9.6.1.6. Спецификатор NAME= в операторе INQUIRE

Скалярной-переменной-стандартного-символьного-типа в спецификаторе NAME= присваивается значение имени файла, если файл его имеет, в противном случае она становится неопределенной. Заметим, что если этот спецификатор входит в оператор INQUIRE для файла, его значение не обязательно то же самое, что имя, заданное спецификатором FILE= . Например, процессор может выдать имя файла, уточненное пользовательской идентификацией. Однако выдаваемое значение должно быть пригодно для использования в качестве значения *выражения-имени-файла* в спецификаторе FILE= оператора OPEN. Если в процессоре есть возможность представления букв верхнего и нижнего регистров, выбор символов, присваиваемых *скалярной-переменной-стандартного-символьного-типа*, зависит от процессора.

9.6.1.7. Спецификатор ACCESS= в операторе INQUIRE

Скалярная-переменная-стандартного-символьного-типа в спецификаторе ACCESS= получает значение SEQUENTIAL, если файл подсоединен для последовательного доступа, или DIRECT, если файл подсоединен для прямого доступа. Если подсоединения нет, она получает значение UNDEFINED.

9.6.1.8. Спецификатор SEQUENTIAL= в операторе INQUIRE

Скалярная-переменная-стандартного-символьного-типа в спецификаторе SEQUENTIAL= получает значение YES, если SEQUENTIAL включено в множество разрешенных способов доступа для файла, NO - если не включено, UNKNOWN - если процессор не может определить, включено ли SEQUENTIAL в множество разрешенных способов доступа к файлу.

9.6.1.9. Спецификатор DIRECT= в операторе INQUIRE

Скалярная-переменная-стандартного-символьного-типа в спецификаторе DIRECT= получает значение YES, если DIRECT включено в множество разрешенных способов доступа к файлу, NO - если не включено, и UNKNOWN, если процессор не может определить, включено ли DIRECT в множество разрешенных способов доступа к файлу.

9.6.1.10. Спецификатор FORM= в операторе INQUIRE

Скалярная-переменная-стандартного-символьного-типа в спецификаторе FORM= получает значение FORMATTED, если файл подсоединен для форматного ввода/вывода, UNFORMATTED - если для неформатного, и UNDEFINED, если подсоединения нет.

9.6.1.11. Спецификатор FORMATTED= в операторе INQUIRE

Скалярная-переменная-стандартного-символьного-типа в спецификаторе FORMATTED= получает значение YES, если FORMATTED включено в множество разрешенных форм для файла, NO - если не включено, и UNKNOWN, если процессор не может определить, включено ли FORMATTED в множество разрешенных форм для файла.

9.6.1.12. Спецификатор UNFORMATTED= в операторе INQUIRE

Скалярная-переменная-стандартного-символьного-типа в спецификаторе UNFORMATTED= получает значение YES, если UNFORMATTED включено в множество разрешенных форм для файла, NO - если не включено, и UNKNOWN, если процессор не может определить, включено ли UNFORMATTED в множество разрешенных форм для файла.

9.6.1.13. Спецификатор RECL= в операторе INQUIRE

Скалярной-целой-переменной в спецификаторе RECL= присваивается либо значение длины записи в файле, подсоединенном для прямого доступа, либо максимальное значение длины записи в файле, подсоединенном для последовательного доступа. Если файл подсоединен для форматного ввода/вывода, длина - это число символов. Если файл подсоединен для неформатного ввода/вывода, длина измеряется в единицах, зависящих от процессора. Если файл не существует, *скалярная-целая-переменная* становится неопределенной.

9.6.1.14. Спецификатор NEXTREC= в операторе INQUIRE

Скалярная-целая-переменная в спецификаторе NEXTREC= получает значение *n*+1, где *n* - номер последней считанной или записанной записи в файле, подсоединенном для прямого доступа. Если файл подсоединен и с момента подсоединения ни одна запись не читалась и не записывалась, *скалярной-целой-переменной* присваивается значение 1. Если файл не подсоединен для прямого доступа или позиция его не определена вследствие предшествующей ситуации ошибки, *скалярная-целая-переменная* становится неопределенной.

9.6.1.15. Спецификатор BLANK= в операторе INQUIRE

Скалярная-переменная-стандартного-символьного-типа в спецификаторе BLANK= получает значение NULL или ZERO в зависимости от того, какое управление пробелами действует в данный момент для файла, подсоединенного для форматного ввода/вывода. Если подсоединения нет или оно неформатное, *скалярная-переменная-стандартного-символьного-типа* получает значение UNDEFINED.

9.6.1.16. Спецификатор POSITION= в операторе INQUIRE

Скалярная-переменная-стандартного-символьного-типа в спецификаторе POSITION= получает значение REWIND, если файл подсоединялся оператором OPEN с установкой позиции на начальную точку, APPEND, если позиция файла при подсоединении устанавливалась на конечную точку, ASIS, если файл подсоединялся без изменения позиции. Если же подсоединения нет или файл подсоединен для прямого доступа, *скалярная-переменная-стандартного-символьного-типа* получает значение UNDEFINED. Если позиция файла с момента подсоединения менялась, *скалярной-переменной-стандартного-символьного-типа* присваивается зависящее от процессора значение, которое может быть REWIND, только если позиция файла установлена на свою начальную точку, и может быть APPEND, только если позиция файла установлена таким образом, чтобы запись "конец файла" была следующей записью, либо, если записи "конец файла" нет-в конечную точку файла.

9.6.1.17. Спецификатор ACTION= в операторе INQUIRE

Скалярная-переменная-стандартного-символьного-типа в спецификаторе ACTION= получает значение READ, если файл подсоединен только для ввода, WRITE - если только для вывода, READWRITE, если файл подсоединен и для ввода, и для вывода. Если подсоединения нет, *скалярная-переменная-стандартного-символьного-типа* получает значение UNDEFINED.

9.6.1.18. Спецификатор READ= в операторе INQUIRE

Скалярная-переменная-стандартного-символьного-типа в спецификаторе READ= получает значение YES, если READ включено в множество разрешенных действий для файла, NO - если не включено, и UNKNOWN, если процессор не может определить включено ли READ в множество разрешенных действий для файла.

9.6.1.19. Спецификатор WRITE= в операторе INQUIRE

Скалярная-переменная-стандартного-символьного-типа в спецификаторе WRITE= получает значение YES, если WRITE включено в множество разрешенных действий для файла, NO - если не включено, и UNKNOWN, если процессор не может определить, включено ли WRITE в множество разрешенных действий для файла.

9.6.1.20. Спецификатор READWRITE= в операторе INQUIRE

Скалярная-переменная-стандартного-символьного-типа в спецификаторе READWRITE= получает значение YES, если READWRITE включено в множество разрешенных действий для файла, NO - если не включено, и UNKNOWN - если процессор не может определить, включено ли READWRITE в множество разрешенных действий для файла.

9.6.1.21. Спецификатор DELIM= в операторе INQUIRE

Скалярная-переменная-стандартного-символьного-типа в спецификаторе DELIM= получает значение APOSTROPHE, если в качестве ограничителя символьных данных, записанных посредством форматирования, управляемого списком или по NAMELIST, должен быть использован апостроф. Если для ограничения таких данных используются кавычки, присваивается значение QUOTE. Если ни апостроф, ни кавычки не используются для ограничения символьных данных, присваивается значение NONE. Если подсоединения нет или оно не для форматного ввода/вывода, *скалярной-переменной-стандартного-символьного-типа* присваивается значение UNDEFINED.

9.6.1.22. Спецификатор PAD= в операторе INQUIRE

Скалярная-переменная-стандартного-символьного-типа в спецификаторе PAD= получает значение NO, если подсоединение файла к устройству включало спецификатор PAD= и его значение было NO. В противном случае *скалярная-переменная-стандартного-символьного-типа* получает значение YES.

9.6.2. Ограничения на спецификаторы запроса

Если переменная или любой связанный с ней объект могут стать определенными или неопределенными в результате их использования в качестве спецификатора в операторе INQUIRE, на них нельзя ссылаться в каком-либо другом спецификаторе в этом же операторе INQUIRE.

Список-спецификаторов-запроса в операторе INQUIRE для файла должен содержать ровно один спецификатор FILE= и не должен содержать спецификатора UNIT= . *Список-спецификаторов-запроса* в операторе INQUIRE для устройства должен содержать ровно один спецификатор UNIT= и не должен содержать спецификатора FILE= . Заданное устройство не обязательно должно существовать или быть подсоединенным к файлу. Если оно подсоединено, запрос производится о подсоединении и о подсоединенном файле.

9.6.3. Запрос о списке вывода

Запрос о списке вывода имеет форму оператора INQUIRE, который не содержит ни спецификатора UNIT= , ни спецификатора FILE= , содержит только спецификатор IOLENGHT= и список вывода.

Скалярной-переменной-стандартного-целого-типа в спецификаторе IOLENGTH= присваивается зависящее от процессора значение, которое получается при использовании списка вывода в неформатном операторе вывода. Значение должно быть

пригодным для использования в качестве спецификатора RECL= в операторе OPEN, подсоединяющем файл для неформатного прямого доступа, когда есть операторы ввода/вывода с тем же списком ввода/вывода.

9.7. Ограничения для ссылок на функции и для элементов списка

Не разрешается ссылаться на функцию в выражении, находящемся в любой части оператора ввода/вывода, если эта ссылка приведет к исполнению другого оператора ввода/вывода. Заметим, что ограничения на вычисление выражений (7.1.7) запрещают некоторые побочные эффекты.

9.8. Ограничения на операторы ввода/вывода

Если устройство или подсоединенный к нему файл не обладает всеми свойствами, необходимыми для выполнения конкретного оператора ввода/вывода, такой оператор не должен ссылаться на это устройство.

Глава 10. РЕДАКТИРОВАНИЕ ВВОДА/ВЫВОДА

Формат, используемый вместе с оператором ввода/вывода, предоставляет информацию, которая управляет преобразованием данных из внутреннего представления в представление символов в последовательности форматных записей и наоборот.

Спецификатор формата (9.4.1.1) в операторе ввода/вывода может ссылаться на оператор FORMAT или на символьное выражение, которое содержит спецификацию формата. Спецификация формата предоставляет явную редактирующую информацию. Спецификатор формата может быть задан также в виде звездочки (*), которая означает форматирование, управляемое списком (10.8). Вместо спецификатора формата может быть задано *имя-группы-NAMELIST*, что означает форматирование по NAMELIST (10.9).

10.1. Методы явной спецификации формата

Явная спецификация формата может быть задана:

1) в операторе FORMAT;
2) в символьном выражении.

10.1.1. Оператор FORMAT

R1001 *оператор-FORMAT* **есть** FORMAT *спецификация-формата*

R1002 *спецификация-формата* **есть** ([*список-элементов-формата*])

Ограничение: *оператор-FORMAT* должен быть помечен.

Ограничение: запятая, используемая для разделения *элементов-формата* в *списке-элементов-формата*, может быть опущена в следующих случаях:

1) между описателем редактирования P и следующим за ним описателем редактирования F, E, EN, ES, D или G (10.6.5);
2) перед описателем редактирования "/", когда отсутствует необязательная спецификация повторения (10.6.2);
3) после описателя редактирования "/";
4) до или после описателя редактирования ":" (10.6.3).

Символы пробела могут предшествовать начальной левой скобке спецификации формата. Дополнительные символы пробела могут появиться в любом месте спецификации формата, не влияя на ее интерпретацию (кроме описателя редактирования "символьная строка" (10.7.1, 10.7.2)).

Примеры операторов FORMAT:

```
5    FORMAT (1PE12.4, I10)
9    FORMAT (I12, /, ' Dates: ', 2 (2I3,I5))
```

10.1.2. Символьная спецификация формата

Результатом вычисления символьного выражения, используемого в качестве спецификации формата в операторе форматного ввода/вывода, должна быть символьная строка, начальная часть которой представляет собой правильную спецификацию формата. Заметим, что спецификация формата начинается с левой скобки и заканчивается правой скобкой.

Во время выполнения ввода/вывода все символьные позиции, вплоть до последней правой скобки, должны быть определены и не могут переопределяться или терять определенность во время выполнения оператора. Если имеются символьные позиции после правой скобки, заканчивающей спецификацию формата, они не обязательно должны быть определены и могут содержать любые символьные данные, не влияя при этом на интерпретацию спецификации формата.

Если спецификатор формата ссылается на символьный массив, это рассматривается как сцепление всех элементов массива в том порядке, который задан для элементов массива. Однако, если спецификатор формата ссылается на элемент символьного массива, спецификация формата должна полностью поместиться в нем.

10.2. Форма списка элементов формата

R1003 *элемент-формата* **есть** [*r*] *описатель-редактирования-данных*
или *управляющий-описатель-редактирования*
или *описатель-редактирования-симв-строка*
или [*r*] (*список-элементов-формата*)

R1004 *r* **есть** *целая-литеральная-константа*

Ограничение: *r* должно быть положительным.

Ограничение: *r* должно быть задано без параметра вида.

Целая литеральная константа *r* называется **спецификацией повторения**.

10.2.1. Описатели редактирования

Описатель редактирования - это либо **описатель редактирования данных**, либо **управляющий описатель редактирования**, либо **описатель редактирования символьная строка.**

R1005 *описатель-редактирования-данных* **есть** I*w*[.*m*]
или B*w*[.*m*]
или O*w*[.*m*]
или Z*w*[.*m*]
или F*w*.*d*
или E*w*.*d* [E*e*]
или EN*w*.*d* [E*e*]
или ES*w*.*d* [E*e*]
или G*w*.*d* [E*e*]
или L*w*
или A[*w*]
или D*w*.*d*

R1006 *w* **есть** *целая-литеральная-константа*

R1007 *m* **есть** *целая-литеральная-константа*

R1008 *d* **есть** *целая-литеральная-константа*

R1009 *e* **есть** *целая-литеральная-константа*

Ограничение: *w* и *e* должны быть положительными.

Ограничение: *w*, *m*, *d* и *e* должны быть специфицированы без параметров вида.

I, B, O, Z, F, E, EN, ES, D, G, L и A указывают способ редактирования.

R1010 *управляющий-описатель-редактирования* ♦
♦ **есть** *описатель-редактирования-позиции*
или [*r*] /
или :
или *описатель-редактирования-знака*
или *k*P
или *описатель-редактирования-интерпр-пробелов*

R1011 *k* **есть** *целая-литеральная-константа-со-знаком*

Ограничение: *k* должно быть специфицировано без параметра вида.

R1012 *описатель-редактирования-позиции* **есть** T*n*
или TL*n*
или TR*n*
или *n*X

R1013 *n* **есть** *целая-литеральная-константа*

Ограничение: *N* должно быть положительным.

Ограничение: *N* должно быть специфицировано без параметра вида.

R1014 *описатель-редактирования-знака* **есть** S
или SP
или SS

R1015 *описатель-редактирования-интерпр-пробелов* **есть** BN
или BZ

В описателе *kP*, *k* называется **масштабным множителем**.

T, TL, TR, X, /, :, S, SP, SS, P, BN и BZ указывают способ редактирования.

R1016 *описатель-редактирования-симв-строка* ♦
♦ **есть** *символьная-литеральная-константа*
или *cH представимый-символ* [*представимый-символ*]...

R1017 *c* **есть** *целая-литеральная-константа*

Ограничение: *c* должно быть положительным.

Ограничение: *c* должно быть специфицировано без параметра вида.

Ограничение: *представимый-символ* в форме *cH* должен относиться к стандартному символьному типу.

Ограничение: *символьная-литеральная-константа* должна быть без параметра вида.

Каждый *представимый-символ* в описателе редактирования "символьная строка" должен быть одним из символов, представимых в процессоре.

Описатель редактирования "символьная строка" предназначен для вывода константных данных, при вводе его использование является ошибкой.

Символы, ограничивающие символьную константу - апостроф или кавычки, внутри нее должны быть представлены парой последовательных символов без промежуточных пробелов. Каждая такая пара представляет одиночное вхождение ограничивающего символа.

В описателе редактирования H число символов после буквы H специфицируется значением *c*.

Если процессор имеет возможность представления букв верхнего и нижнего регистров, для описателей редактирования это не имеет значения, кроме символов, следующих за H в описателе редактирования H, и символов в символьной константе.

10.2.2. Поля

Поле - это часть записи, которая читается при вводе или записывается при выводе, когда управление форматом встречает один из описателей редактирования данных или описатель редактирования "символьная строка". **Ширина поля** - это размер поля в символах.

10.3. Взаимодействие между списком ввода/вывода и форматом

Начало форматной передачи данных, использующей спецификацию формата, инициирует **управление форматом** (9.4.4.4.2). Каждое действие управления форматом зависит от информации, предоставляемой совместно:

1) следующим описателем редактирования в спецификации формата и
2) следующим действующим элементом списка ввода/вывода, если он есть.

Если в списке ввода/вывода задан хотя бы один действующий элемент списка, в спецификации формата должен присутствовать как минимум один описатель редактирования данных. Заметим, что пустую спецификацию формата вида ( ) можно использовать только в случае, если список ввода/вывода пуст или все его элементы нулевого размера.

Кроме элемента формата, перед которым есть спецификация повторения *r*, спецификация формата интерпретируется слева направо.

Элемент формата, которому предшествует спецификация повторения, обрабатывается как список из *r* разделенных запятыми элементов, каждый из которых идентичен элементу формата, но без спецификации повторения. Заметим, что если спецификация повторения опущена, это трактуется так же, как спецификация повторения со значением 1.

Каждому описателю редактирования данных, интерпретируемому в спецификации формата, соответствует один действующий элемент, задаваемый списком ввода/вывода (9.4.2), если не считать того, что элемент списка ввода/вывода комплексного типа требует интерпретации двух описателей редактирования F, E, EN, ES, D или G. Для управляющего описателя редактирования или описателя редактирования "символьная строка" нет соответствующего элемента в списке ввода/вывода и управление форматом обменивается информацией непосредственно с записью.

Когда управление форматом обнаруживает в спецификации формата описатель редактирования данных, оно определяет, есть ли соответствующий действующий элемент в списке ввода/вывода. Если он есть, производится обмен информацией, соответствующим образом отредактированной, между элементом списка и записью и затем управление форматом продолжается. Если соответствующего элемента нет, управление форматом завершается.

Если управление форматом встречает в спецификации формата описатель редактирования "двоеточие", а очередной действующий элемент списка ввода/вывода не задан, управление форматом завершается.

Если управление форматом встречает крайнюю правую скобку для всей спецификации формата, а очередной действующий элемент списка ввода/вывода не задан, управление форматом завершается. Однако, если такой элемент имеется, позиция файла устанавливается на начало следующей записи и управление форматом возвращается на начало того элемента формата, который заканчивается предыдущей правой скобкой. Если такой правой скобки нет, управление форматом возвращается на первую левую скобку в спецификации формата. При таком возврате повторно используемая часть спецификации формата должна содержать хотя бы один описатель редактирования данных. Если управление форматом возвращается на скобку, перед которой стоит спецификация повторения, эта спецификация тоже используется повторно. Сам по себе возврат управления форматом не влияет на масштабный множитель (10.6.5.1), на описатели редактирования, управляющие знаком (10.6.4), на описатели редактирования, интерпретирующие пробелы (10.6.6).

Пример. Спецификация формата

```
10 FORMAT(1X, 2 (F10.3, I5) )
```

со списком вывода

```
WRITE (10,10) 10.1, 3, 4.7, 1, 12.4, 5, 5.2, 6
```

производит тот же вывод, что спецификация формата:

```
10 FORMAT (1X, F10.3, I5, F10.3, I5 / F10.3, I5, F10.3, I5)
```

10.4. Установка позиции файла при управлении форматом

После обработки любого из описателей редактирования данных или описателя редактирования "символьная строка" позиция файла устанавливается после последнего считанного или записанного символа в текущей записи.

Позиция файла после обработки описателей редактирования T, TL, TR, и X описана в 10.6.1. Позиция файла после описателя редактирования "/" описана в 10.6.2.

После возврата управления форматом (10.3) позиция файла устанавливается так же, как после обработки описателя редактирования "/" (10.6.2).

Во время операции чтения все необработанные символы пропускаются всякий раз, когда читается следующая запись.

10.5. Описатели редактирования данных

Описатели редактирования данных вызывают преобразование данных в их внутреннее представление или из внутреннего представления. Символы в записи должны иметь параметр вида по умолчанию, если они соответствуют значениям объектов числового, логического или стандартного символьного типа, и должны иметь параметр вида не по умолчанию, если соответствуют объектам данных нестандартного символьного типа. Символы, передаваемые в запись в результате работы описателя редактирования "символьная строка", должны иметь параметр вида, подразумеваемый по умолчанию. При вводе специфицированная переменная (если не возникла ситуация ошибки, "конец записи" или "конец файла") становится определенной. При выводе, если задано выражение, оно вычисляется.

10.5.1. Описатели редактирования чисел

Описатели редактирования I, B, O, Z, F, E, EN, ES, D и G используются для задания ввода/вывода целых, вещественных и комплексных данных. Используются следующие основные правила.

1. При вводе начальные пробелы не являются значащими. Интерпретация остальных пробелов определяется комбинацией спецификатора BLANK= (9.3.4.6) и управления пробелами BN или BZ, которые действуют в данный момент для устройства (10.6.6). Знаки плюс можно опускать. Поле из одних пробелов рассматривается как нуль.
2. При вводе с описателем редактирования F, E, EN, ES, D и G десятичная точка, находящаяся в поле ввода, отменяет действие той части описателя редактирования, которая определяет положение десятичной точки. Поле ввода может содержать больше цифр, чем использует процессор для аппроксимации значения данного.
3. При выводе с описателем редактирования I, F, E, EN, ES, D или G представлению положительного или нулевого внутреннего значения в поле может предшествовать плюс в зависимости от того, что задано описателями редактирования S, SP, SS или процессором. Представлению отрицательного внутреннего значения в поле должен предшествовать минус. Однако процессор не должен вырабатывать отрицательные нули со знаком в записи форматного вывода.
4. При выводе представление сдвигается к правому краю поля. Если число символов, полученное в результате редактирования, меньше, чем ширина поля, в поле вставляются начальные пробелы.
5. Если при выводе число полученных символов больше, чем ширина поля, или если экспонента превышает длину, заданную для нее описателями редактирования E*w.dEe*, EN*w.dEe*, ES*w.dEe* или G*w.dEe*, процессор должен заполнить все поле ширины *w* звездочками. Однако, если с учетом того, что все необязательные символы опущены, ширина поля не будет превышена, процессор не должен вырабатывать звездочки. Отметим, что во время действия описателя редактирования SP плюс не является необязательным.

10.5.1.1. Редактирование целых данных

Описатели редактирования I*w*, I*w.m*, B*w*, B*w.m*, O*w*, O*w.m*, Z*w* и Z*w.m* указывают, что редактируемое поле занимает *w* позиций. Заданный элемент списка ввода/вывода должен быть целого типа. Кроме того, для редактирования целых данных можно использовать описатель редактирования G (10.5.4.1.1).

При вводе *m* ни на что не влияет.

Символьная строка в поле ввода для описателя редактирования I должна иметь форму целой константы с необязательным знаком, который может влиять только на интерпретацию пробелов. Для описателей редактирования B, O и Z символьная строка

должна состоять из двоичных, восьмеричных и шестнадцатеричных цифр (R408, R409, R410) в соответствующем поле ввода. Если процессор способен представлять буквы и верхнего, и нижнего регистра, шестнадцатеричные цифры нижнего регистра от a до f в шестнадцатеричном поле ввода эквивалентны соответствующим шестнадцатеричным цифрам верхнего регистра.

Поле вывода для описателя редактирования I*w* состоит из некоторого количества (включая нулевое) начальных пробелов, за которыми следует знак "минус", если значение внутреннего данного отрицательно, или необязательный знак "плюс" в противном случае, далее следует абсолютная величина внутреннего значения в форме целой константы без знака и без начальных нулей. Заметим, что целая константа всегда имеет хотя бы одну цифру.

Поле ввода для описателей редактирования B*w*, O*w* и Z*w* состоит из некоторого (в том числе, нулевого) количества пробелов, за которыми следуют внутренние значения в той же самой форме, что и цифры двоичных, восьмеричных или шестнадцатеричных констант, соответственно, с теми же значениями и без начальных нулей. Заметим, что двоичная, восьмеричная или шестнадцатеричная константа всегда содержит, как минимум, одну цифру.

Поле вывода для описателя редактирования I*w.m*, B*w.m*, O*w.m* и Z*w.m* такое же, как для I*w*, B*w*, O*w* и Z*w*, соответственно, с той разницей, что целая константа без знака состоит, как минимум из *m* цифр и в случае необходимости имеет начальные нули. Значение *m* не должно превышать значение *w*. Если *m* - нуль и значение внутреннего данного - нуль, поле вывода состоит из одних пробелов, независимо от действующего в данный момент управления знаком.

10.5.1.2. Описатели редактирования вещественных и комплексных данных

Описатели редактирования F, E, EN, ES и D задают редактирование вещественных и комплексных данных. Элемент списка ввода/вывода, соответствующий описателю редактирования F, E, EN, ES или D должен быть вещественным или комплексным. Кроме того, для редактирования вещественных или комплексных данных можно использовать описатель редактирования G (10.5.4.1.2).

Если процессор способен представлять буквы и верхнего, и нижнего регистра, буквы нижнего регистра в экспоненте числового поля ввода эквивалентны соответствующим буквам верхнего регистра.

10.5.1.2.1. Описатель редактирования F

Описатель редактирования F*w.d* означает, что поле занимает *w* позиций, а его дробная часть состоит из *d* цифр.

Поле ввода состоит из необязательного знака, за которым следует строка из одной или более цифр, которая может содержать необязательную десятичную точку и пробелы, интерпретируемые как нули. Если поле ввода содержит десятичную точку, *d* игнорируется. Если десятичная точка опущена, правые *d* цифр строки с начальными нулями, если необходимо, интерпретируются как дробная часть представленного значения. Строка цифр может содержать больше цифр, чем использует процессор для аппроксимации значения константы. За основной формой может следовать экспонента в следующем виде:

1) целая константа с явным знаком;
2) E, за которым следует какое-то количество (включая нулевое) пробелов, затем целая константа с необязательным знаком;
3) D, за которым следует некоторое (в том числе нулевое) количество пробелов, затем целая константа с необязательным знаком.

Экспонента, содержащая D, обрабатывается аналогично экспоненте, содержащей E.

Заметим, что если поле ввода не содержит экспоненту, эффект такой же, как если бы за основной формой следовала экспонента со значением -*k*, где *k* устанавливает масштабный множитель (10.6.5.1).

Поле вывода состоит из пробелов (если это необходимо), за которыми следует минус, когда внутреннее значение отрицательно, или необязательный плюс в противном случае, затем строка цифр, содержащая десятичную точку и представляющая абсолютную величину внутреннего значения, преобразованную в соответствии с установленным масштабным множителем и округленную до *d* цифр дробной части. Начальные нули не разрешаются, кроме необязательного нуля, который ставится слева от десятичной точки, если абсолютная величина значения в поле вывода меньше единицы. Необязательный нуль становится обязательным, если без него в поле вывода совсем не будет цифр.

10.5.1.2.2. Описатели редактирования E и D

Описатели редактирования E*w.d*, D*w.d* и E*w.dEe* означают, что внешнее поле занимает *w* позиций, дробная его часть состоит из *d* цифр (если не действует масштабный множитель, больший единицы), а экспоненциальная часть состоит из *e* цифр. Значение *e* не влияет на ввод, а *d* - на вывод.

Форма и интерпретация поля ввода та же, что при F-редактировании (10.5.1.2.1).

Форма поля вывода при нулевом масштабном множителе следующая:

$[\pm][0].x_1x_2...x_d exp$

где ± - означает плюс или минус,

$x_1, x_2 ... x_d$ - *d* старших значащих цифр значения данного после округления,

exp - десятичная экспонента, имеющая одну из форм, приведенных ниже:

Описатель редактирования	Абсолютное значение экспоненты	Форма экспоненты
E*w.d*	$\lvert exp \rvert <= 99$	$E\pm z_1z_2$ или $\pm 0z_1z_2$
	$99 < \lvert exp \rvert <= 999$	$\pm z_1z_2z_3$
E*w.dEe*	$\lvert exp \rvert <= 10^e-1$	$E\pm z_1z_2 ... z_e$
D*w.d*	$\lvert exp \rvert <= 99$	$D\pm z_1z_2$ или $E\pm z_1z_2$ или $\pm 0z_1z_2$
	$99 < \lvert exp \rvert <= 999$	$\pm z_1z_2z_3$

где каждое *z* - цифра.

Знак в экспоненте вырабатывается. Если значение экспоненты нуль, вырабатывается знак плюс. Нельзя использовать формы E*w.d* и D*w.d*, если $\lvert exp \rvert > 999$.

Масштабный коэффициент *k* управляет десятичной нормализацией (10.2.1 и 10.6.5.1). Если $-d < k <= 0$, поле вывода содержит ровно *k* начальных нулей и *d* - $\lvert k \rvert$ значащих цифр после десятичной точки. Если $0 < k < d+2$, поле вывода содержит ровно *k* значащих цифр слева от десятичной точки и $d-k+1$ значащих цифр справа от десятичной точки. Другие значения *k* не допускаются.

10.5.1.2.3. Описатель редактирования EN

Описатель редактирования EN вырабатывает поле вывода в форме вещественного числа в инженерной нотации, так что десятичная экспонента всегда кратна трем, а абсолютное значение значащей части (4.3.1.2) больше или равно единице и меньше 1000,

кроме случая, когда выводимое значение равно нулю. Масштабный множитель не влияет на этот описатель редактирования при выводе.

Описатель редактирования имеет две формы: EN*w.d* и EN*w.dEe* означающие, что внешнее поле занимает *w* позиций, дробная часть его состоит из *d* цифр, а экспоненциальная часть - из *e* цифр.

Форма и интерпретация поля ввода те же, что при F-редактировании (10.5.1.2.1).

Форма поля вывода:

[±] $yyy.x_1x_2...x_d exp$

где ± - плюс или минус,

yyy - 1-3 десятичные цифры, представляющие старшие значащие цифры значения данного после округления (*yyy* - целое, такое, что 1 <= *yyy* < 1000 или, если выводимое значение - нуль, *yyy* = 0),

$x_1x_2...x_d$ - *d* следующих старших значащих цифр значения данного после округления,

exp - кратная трем десятичная экспонента, которая имеет одну из форм, приведенных ниже:

Описатель редактирования	Абсолютное значение экспоненты	Форма экспоненты
EN*w.d*	|*exp*| <= 99	$E\pm z_1z_2$ или $\pm 0z_1z_2$
	99 < |*exp*| <= 999	$\pm z_1z_2z_3$
EN*w.dEe*	|*exp*| <= 10^e-1	$E\pm z_1z_2 ... z_e$

где каждое *z* - десятичная цифра.

Знак в экспоненте вырабатывается. Если значение экспоненты нуль, вырабатывается знак "плюс". Нельзя использовать форму EN*w.d*, если |*exp*| > 999.

Примеры:

Внутреннее значение	Поле вывода с использованием SS, EN12.3
6.421	6.421E+00
-.5	-500.000E-03
.00217	2.170E-03
4721.3	4.721E+03

10.5.1.2.4. Описатель редактирования ES

Описатель редактирования ES вырабатывает поле вывода в форме вещественного числа в научной нотации, так что абсолютное значение значащего (4.3.1.2) больше или равно 1 и меньше 10, кроме случая, когда выводимое значение равно нулю. Масштабный множитель не влияет на этот описатель редактирования.

Описатель редактирования имеет две формы: ES*w.d* и ES*w.dEe*, означающие, что внешнее поле занимает *w* позиций, дробная часть его состоит из *d* цифр, а экспоненциальная часть - из *e* цифр.

Форма и интерпретация поля ввода те же, что при F-редактировании (10.5.1.2.1).

Форма поля вывода:

[±] $y.x_1x_2...x_d exp$

где ± - означает плюс или минус;

y - десятичная цифра, представляющая старшую значащую цифру значения данного после округления;

$x_1x_2...x_d$ - *d* следующих старших значащих цифр значения данного после округления;

exp - десятичная экспонента, которая имеет одну из форм, приведенных ниже:

Описатель редактирования	Абсолютное значение экспоненты	Форма экспоненты
ES*w.d*	|*exp*| ≤ 99	$E\pm z_1z_2$ или $\pm 0z_1z_2$
	99 < |*exp*| ≤ 999	$\pm z_1z_2z_3$
ES*w.dEe*	|*exp*| ≤ 10^e-1	$E\pm z_1z_2 \dots z_e$

где *z* - десятичная цифра.

Знак в экспоненте вырабатывается. Если значение экспоненты нуль, вырабатывается знак "плюс". Нельзя использовать форму ES*w.d*, если |*exp*| > 999.

Примеры:

Внутреннее значение	Поле вывода с использованием SS, ES12.3
6.421	6.421E+00
-.5	-5.000E-01
.00217	2.170E-03
4721.3	4.721E+03

10.5.1.2.5. Редактирование комплексных данных

Комплексное данное образуется парой отдельных вещественных данных. Редактирование скалярного данного комплексного типа задается двумя описателями редактирования, каждый из которых задает редактирование вещественных данных. Первый из описателей редактирования задает действительную часть, второй - мнимую. Эти два описателя редактирования могут быть разными. Между ними могут быть управляющие описатели редактирования и описатели редактирования "символьная строка".

10.5.2. Редактирование логических данных

Описатель редактирования L*w* означает, что поле занимает *w* позиций. Заданный элемент списка ввода/вывода должен быть логического типа. Кроме того, для редактирования логических данных можно использовать описатель редактирования G (10.5.4.2).

Поле ввода состоит из необязательных пробелов, за которыми может следовать необязательная десятичная точка, затем T для значения "истина" или F для значения "ложь", за которыми могут в поле находиться дополнительные символы. Отметим, что логические константы .TRUE. и .FALSE. являются допустимыми формами ввода. Если процессор способен представить буквы и верхнего, и нижнего регистров, в логическом поле ввода буква нижнего регистра эквивалентна соответствующей букве верхнего регистра.

Поле вывода состоит из *w*-1 пробелов, за которыми следует T или F, в зависимости от того, является ли значением внутреннего данного "истина" или "ложь", соответственно.

10.5.3. Редактирование символьных данных

Описатель редактирования A[*w*] используется с элементами списка ввода/вывода символьного типа. Кроме того, для редактирования символьных данных можно использовать описатель редактирования G (10.5.4.3). Все символы, передаваемые и преобразуемые под управлением описателей редактирования A или G, должны иметь то же значение параметра вида, что элемент списка ввода/вывода.

Если в описателе редактирования A задана ширина поля *w*, поле состоит из *w* символов. Если ширина поля *w* не задана, число символов в поле равно длине соответствующего элемента списка ввода/вывода независимо от значения его параметра вида.

Предположим, *len* - длина элемента списка ввода/вывода. Если заданная ширина поля *w* для ввода по A больше или равна *len*, правые *len* символов выбираются из поля ввода. Если *w* меньше *len*, во внутреннее представление занесутся сдвинутые влево *w* символов, а за ними *len-w* концевых пробелов.

При выводе, если заданная ширина поля *w* для A больше *len*, поле вывода образуется из *w-len* пробелов, за которыми следуют *len* символов из внутреннего представления. Если *w* меньше или равно *len*, поле вывода состоит из *w* левых символов внутреннего представления.

Заметим, что для нестандартных символьных типов символ для заполнения пустых позиций зависит от процессора.

10.5.4. Обобщенные описатели редактирования

Описатели редактирования G*w.d* и G*w.dEe* используются с элементами списка ввода/вывода любого встроенного типа. Данные описатели редактирования означают, что внешнее поле занимает *w* позиций, из которых мантисса содержит максимум *d* цифр и экспоненциальная часть содержит *e* цифр. Когда описатели редактирования используются для обозначения ввода/вывода целых, логических и символьных данных, *d* и *e* игнорируются.

10.5.4.1. Обобщенные описатели редактирования чисел

Если для обозначения ввода/вывода вещественных, целых или комплексных данных используются описатели редактирования G*w.d* и G*w.dEe*, то следуют общие правила редактирования чисел (10.5.1).

Заметим, что описатель редактирования G*w.dEe* следует любым дополнительным правилам для описателя редактирования E*w.dEe*.

10.5.4.1.1. Обобщенные описатели редактирования данных целого типа

Если для обозначения ввода/вывода целых данных используются описатели редактирования G*w.d* и G*w.dEe*, то действуют правила для описателя редактирования I*w* (10.5.1.1).

10.5.4.1.2. Обобщенные описатели редактирования данных вещественного и комплексного типов

Форма и интерпретация поля ввода такая же, как для описателя редактирования F (10.5.1.2.1).

Способ представления в поле вывода зависит от абсолютной величины редактируемого данного. Предположим, что *N* - абсолютная величина внутреннего представления данного. Если $0 < N < 0.1-0.5*10^{-d-1}$ или $N >= 10^{d}-0.5$, то описатель редактирования G*w.d* при выводе - то же самое, что описатель редактирования *k*PE*w.d* при выводе, а описатель G*w.dEe* при выводе - то же, что *k*PE*w.dEe* при выводе, где *k* - действующий масштабный множитель (10.6.5.1). Если $0.1 - 0.5*10^{-d-1} <= N < 10^{d}-0.5$ или $N = 0$, масштабный множитель игнорируется и значение *N* определяет редактирование следующим образом:

Абсолютная величина данного	Эквивалентное преобразование
$N = 0$	F(*w* - *n*).(*d*-1), *n* ('*b*')
$0.1 - 0.5*10^{-d-1} <= N < 1 - 0.5*10^{-d}$	F(*w* - *n*).*d*, *n* ('*b*')
$1 - 0.5*10^{-d} <= N < 10 - 0.5*10^{-d+1}$	F(*w* - *n*).(*d*-1), *n* ('*b*')
$10 - 0.5*10^{-d+1} <= N < 100 - 0.5*10^{-d+2}$	F(*w* - *n*).(*d*-2), *n* ('*b*')
.	.
.	.
.	.
$10^{d-2} - 0.5*10^{-2} <= N < 10^{d-1} - 0.5*10^{-1}$	F(*w* - *n*).1, *n* ('*b*')
$10^{d-1} - 0.5*10^{-1} <= N < 10^{d} - 0.5$	F(*w* - *n*).0, *n* ('*b*')

где *b* - пробел. *n*=4 для G*w*.*d* и *n*=*e*+2 для G*w*.*d*E*e*.

Заметим, что масштабный множитель игнорируется, если величина редактируемого данного не находится вне диапазона, позволяющего эффективно использовать описатель редактирования F.

10.5.4.2. Обобщенные описатели редактирования данных логического типа

Если для обозначения ввода/вывода логических данных используются описатели редактирования G*w*.*d* и G*w*.*d*E*e*, действуют правила для описателя редактирования L*w* (10.5.2).

10.5.4.3. Обобщенные описатели редактирования данных символьного типа

Если для обозначения ввода/вывода символьных данных используются описатели редактирования G*w*.*d* и G*w*.*d*E*e*, действуют правила для описателя редактирования A*w* (10.5.3).

10.6. Управляющие описатели редактирования

Управляющие описатели редактирования не вызывают ни передачи данных, ни преобразования данных во внутреннее представление (или наоборот), но могут повлиять на преобразование, выполняемое последующими описателями редактирования данных.

10.6.1. Описатели редактирования позиции

Описатели редактирования T, TL, TR и X задают позицию в записи, в которую или из которой должен передаваться следующий символ. Если какой-либо из символов, которые пропускаются описателями редактирования T, TL, TR или X, относится к нестандартному символьному типу, результат редактирования этой позиции зависит от процессора.

Позиция, заданная описателем редактирования T, может находиться в любом направлении от текущей позиции. Это позволяет при вводе отдельные части записи обрабатывать повторно и, может быть, с другим описателем редактирования.

Позиция, заданная описателем редактирования X, находится перед текущей. При вводе, если не заданы символы, которые надо передать, можно указать позицию после последнего символа. Отметим, что описатель редактирования *n*X действует так же, как описатель редактирования TR*n*.

При выводе описатели редактирования T, TL, TR и X сами по себе не вызывают передачи символов и, таким образом, не влияют на длину записи. Если символы передаются в позиции, находящиеся после позиции, заданной описателями

редактирования T, TL, TR и X (начиная с нее), позиции пропускаются и те из них, которые не были заполнены, заполняются пробелами. Результат такой же, как если бы вся запись первоначально заполнялась пробелами.

При выводе можно заменить символы в записи. Однако описатели редактирования T, TL, TR и X никогда не вызывают непосредственной замены уже помещенных в запись символов. Эти описатели устанавливают позицию так, что последующее редактирование вызовет замену.

10.6.1.1. Описатели редактирования T, TL и TR

Левая граница табуляции влияет на установку позиции с помощью описателей редактирования T и TL. Устанавливаемая непосредственно перед передачей данных, она определяется как символьная позиция в текущей записи, на которую ссылается оператор ввода/вывода. Если во время передачи данных позиция файла устанавливается на другую запись, левая граница табуляции определяется как символьная позиция 1 в этой записи.

Описатель редактирования T*n* указывает, что передача следующего символа в запись или из записи будет происходить в *n*-й позиции символов относительно левой границы табуляции.

Описатель редактирования TL*n* указывает, что передача следующего символа в запись или из записи будет происходить в позиции, находящейся на *n* символов назад от текущей. Однако, если *n* больше, чем разница между текущей позицией и левым пределом табуляции, описатель редактирования TL*n* означает, что передача следующего символа в запись или из записи будет производиться в левой границе табуляции.

Описатель редактирования TR*n* указывает, что передача следующего символа в запись или из записи будет происходить в символьной позиции, находящейся на *n* символов впереди текущей.

Заметим, что *n* должно быть задано и быть больше нуля.

10.6.1.2. Описатель редактирования X

Описатель редактирования *n*X указывает, что передача следующего символа в запись или из записи будет производиться в позицию или из позиции, расположенной на *n* символов впереди текущей. Заметим, что *n* должно быть задано и быть больше нуля.

10.6.2. Описатель редактирования "дробная черта"

Описатель редактирования "дробная черта" означает конец передачи в текущую запись или из нее.

При вводе из файла, подсоединенного для последовательного доступа, оставшаяся часть записи пропускается и позиция файла устанавливается на начало следующей записи. Эта запись становится текущей. При выводе в файл, подсоединенный для последовательного доступа, создается новая пустая запись, которая становится текущей записью. Эта новая запись затем становится последней и текущей записью в файле, а позиция файла устанавливается на начало созданной новой записи.

Если файл подсоединен для прямого доступа, номер записи увеличивается на единицу и позиция файла устанавливается на начало той записи, которая имеет полученный номер. Эта запись становится текущей.

Заметим, что при выводе можно записать пустую (не содержащую символов) запись. Если файл внутренний или подсоединенный для прямого доступа, запись дополняется символами пробела. Отметим, что при вводе может быть пропущена целиком запись. Спецификация повторения в описателе редактирования "дробная черта" является необязательной. Если она не задана, подразумевается значение 1.

10.6.3. Описатель редактирования "двоеточие

Описатель редактирования "двоеточие" заканчивает управление форматом, если в списке ввода/вывода (9.4.2) больше нет действующих элементов. Если они есть, описатель редактирования "двоеточие" никакого действия не производит.

10.6.4. Описатели редактирования S, SP и SS

Описатели редактирования S, SP и SS могут быть использованы для управления необязательными символами "плюс" в полях числового вывода. В начале выполнения каждого форматного оператора вывода в процессоре есть режим выработки плюса в полях числового вывода. Если встретился описатель редактирования SP, процессор должен вырабатывать плюс во всех последующих позициях, где обычно предусматривается необязательный плюс. Если встретился описатель редактирования SS, процессор не должен вырабатывать плюс во всех последующих позициях, где обычно предусмотрен необязательный плюс. Если встретился описатель редактирования S, восстанавливается процессорный режим выработки плюса.

Описатели редактирования S, SP и SS влияют только на описатели редактирования I, F, E, D, EN, ES и G во время выполнения оператора вывода. Они никак не влияют на выполнение операторов ввода.

10.6.5. Описатель редактирования P

Описатель редактирования *k*P устанавливает значение масштабного множителя равным *k*. Масштабный множитель может повлиять на редактирование числовых величин.

10.6.5.1. Масштабный множитель

В начале выполнения каждого оператора ввода/вывода значение масштабного множителя равно нулю. Это касается всех последовательно интерпретируемых описателей редактирования F, E, EN, ES, D и G до тех пор, пока не встретится другой масштабный множитель, который затем и устанавливается. Заметим, что возврат управления форматом (10.3) не влияет на установленный масштабный множитель.

Масштабный множитель *k* влияет на соответствующее редактирование следующим образом.

1) Для описателей редактирования F, E, EN, ES, D и G при вводе (при условии, что поле не содержит экспоненты) и для описателя редактирования F при выводе действие масштабного множителя заключается в том, что число во внешнем представлении равно внутреннему представлению, умноженному на 10^k.
2) Для описателей редактирования F, E, EN, ES, D и G при вводе, если в поле есть экспонента, масштабный множитель игнорируется.
3) Для описателей редактирования E и D при выводе значащая (4.3.1.2) часть величины, которая должна быть выработана, умножается на 10^k, а экспонента уменьшается на k.
4) При выводе с описателем редактирования G действие масштабного множителя подавляется, если абсолютная величина редактируемого данного не выходит за пределы диапазона, позволяющего использовать описатель редактирования F. Если же требуется описатель редактирования E, масштабный множитель действует так же, как при выводе с описателем редактирования E.
5) Масштабный множитель не влияет на описатели редактирования EN и ES при выводе.

10.6.6. Описатели редактирования BN и BZ

Описатели редактирования BN и BZ можно использовать для задания интерпретации пробелов (кроме начальных) в числовых полях ввода. В начале выполнения каждого форматного оператора ввода эти символы пробела из файла, подсоединенного с помощью оператора OPEN, игнорируются или интерпретируются как нули в зависимости от значения спецификатора BLANK= (9.3.4.6), действующего в данный момент для этого устройства; внутренний файл трактуется как файл, открытый с BLANK='NULL'. Если в спецификации формата встретился описатель редактирования BN, все (кроме начальных) символы пробела в последующих полях ввода игнорируются. Действие этого режима можно представить следующим образом: из поля ввода удаляются все пробелы, оставшаяся часть поля сдвигается вправо, и все удаленные пробелы заменяются начальными. Однако поле, состоящее из одних пробелов, имеет значение 0. Если в спецификации формата встретился описатель редактирования BZ, все (кроме начальных) символы пробела в последующих числовых полях ввода трактуются как нули.

Описатели редактирования BN и BZ действуют только на описатели редактирования I, B, O, Z, F, E, EN, ES, D и G при выполнении операторов ввода. При выполнении операторов вывода они никакого действия не производят.

10.7. Описатели редактирования "символьная строка"

Этот описатель редактирования нельзя использовать при вводе.

10.7.1. Описатель редактирования "символьная константа"

Описатель редактирования "символьная константа" вызывает запись символов (включая пробелы), заключенных между ограничителями самого описателя редактирования. Заметим, что ограничителем должен быть символ "апостроф" или "кавычки".

Описатель редактирования "символьная константа" предполагает, что ширина поля равна числу символов, содержащихся между ограничивающими символами (не включая их). Внутри поля два смежных ограничивающих символа рассматриваются как один символ.

10.7.2. Описатель редактирования H

Описатель редактирования cH вызывает запись символьной информации из следующих c символов (включая пробелы), расположенных после буквы H самого описателя редактирования cH в *списке-элементов-формата*.

10.8. Форматирование, управляемое списком

Символы в одной или нескольких записях, управляемых списком, образуют последовательность значений и разделителей значений. Конец записи имеет тот же эффект, что символ пробела, если он не находится внутри символьной константы. Любая последовательность из двух или более пробелов трактуется как один пробел, если только она не находится внутри символьной константы.

Каждое значение есть либо пустое значение, либо одна из следующих форм:

```
c
r*c
r*
```

где *c* - литеральная константа или символьная константа без ограничивающих символов, а *r* - ненулевая целая литеральная константа без знака и без заданного параметра вида. Форма *r*c* эквивалентна *r* последовательным вхождениям константы *c*, а форма *r** эквивалентна *r* последовательным вхождениям пустых значений. Ни одна из этих форм не должна содержать промежуточных пробелов, кроме тех, которые разрешены в константе *c*.

Разделитель значений имеет одну из следующих форм:

1) запятая, перед и после которой могут находиться пробелы;
2) дробная черта, перед и после которой могут находиться пробелы;
3) один или несколько пробелов между двумя отличными от пробела значениями или после последнего отличного от пробела значения, где отличное от пробела значение - это константа, форма *r*c* или форма *r** .

10.8.1. Ввод, управляемый списком

Для ввода, управляемого списком, допустимы те формы вводимых значений, которые соответствуют форматным спецификациям для данного типа. Исключения приведены ниже. Форма вводимого значения должна быть допустимой для типа, который имеет следующий действующий элемент списка ввода. Пробелы никогда не используются как нули, кроме того, промежуточные пробелы недопустимы во всех константах, кроме символьных и комплексных (см. ниже). Заметим, что конец записи, если он не находится внутри символьной константы, имеет тот же эффект, что пробел.

Если следующий действующий элемент списка ввода имеет тип целый, значение записи при вводе интерпретируется так, как если бы был использован описатель редактирования I*w* с подходящим значением *w*.

Когда следующий действующий элемент списка ввода имеет тип вещественный, вводимые значения имеют вид числового поля ввода. Числовое поле ввода - это поле, пригодное для описателя редактирования F (10.5.1.2.1), т.е. предполагается, что цифры дробной части отсутствуют, кроме случая, когда в поле появляется десятичная точка.

Если следующий действующий элемент списка ввода имеет тип комплексный, вводимая информация состоит из левой скобки, за которой следует упорядоченная пара разделенных запятой числовых полей ввода, за которыми следует правая скобка. Первое числовое поле является действительной частью комплексной константы, второе поле - мнимая ее часть. До и после каждого из полей ввода могут находиться пробелы. Между действительной частью и запятой или между запятой и мнимой частью может находиться конец записи.

Если следующий действующий элемент списка ввода имеет тип логический, вводимая информация не может содержать среди необязательных символов, разрешенных для описателя редактирования L, ни запятых, ни пробелов, ни дробных черт.

Если следующий действующий элемент списка ввода имеет тип символьный, вводимая информация состоит из символьной литеральной константы с тем же параметром вида, что и следующий действующий элемент списка ввода. Символьная константа может быть продолжена от конца одной записи к началу следующей, но при этом конец записи не должен находиться между сдвоенными апострофами в константе, ограниченной апострофами, или между сдвоенными кавычками в константе, ограниченной кавычками. Конец записи не добавляет в константу ни пробелов, ни каких-либо других символов. Константа может занимать столько записей, сколько потребуется. В ней могут находиться символы: пробел, запятая и дробная черта.

Если следующий действующий элемент списка ввода имеет тип символьный и, кроме того

1) символьная константа не содержит разделяющих пробелов, запятых и дробных черт;
2) символьная константа не пересекает границу записи;
3) первый отличный от пробела символ не есть апостроф или кавычки;
4) первые символы не являются числовыми, за которыми бы следовала звездочка,

тогда ограничивающие апострофы или кавычки не являются необходимыми.

Если ограничители опущены, первый же символ пробел, запятая или дробная черта, либо конец записи заканчивают константу, а апострофы или кавычки внутри данного не надо дублировать.

Предположим, что *len* - длина следующего действующего элемента, а *w* - длина символьной константы. Если *len* меньше или равно *w*, в следующий действующий элемент передаются левые *len* символов константы. Если *len* больше *w*, константа передается в левые *w* символов следующего действующего элемента, а оставшиеся *len-w* символов следующего действующего элемента заполняются пробелами. Заметим, что эффект тот же, что и при присваивании значения константы следующему действующему элементу при помощи символьного оператора присваивания (7.5.1.4).

10.8.1.1. Пустые значения

Пустое значение задается:

1) формой *r* * ;
2) отсутствием символов между смежными разделителями значений или
3) отсутствием значения перед первым разделителем значений в записи, считываемой при каждом выполнении оператора ввода, управляемого списком.

Заметим, что конец записи, за которым следует любой другой разделитель (с разделяющими пробелами или без них), не задает пустого значения. Пустое значение не влияет на статус определенности следующего элемента списка ввода. Пустое значение нельзя использовать для представления действительной или мнимой части комплексной константы, но можно представить комплексную константу полностью в виде одного пустого значения.

Дробная черта в качестве разделителя значений во время выполнения оператора ввода, управляемого списком, вызывает завершение его выполнения после того, как будет присвоено предыдущее значение. Все символы, оставшиеся в текущей записи, игнорируются. Если остались еще элементы списка ввода, эффект такой, как если бы им соответствовали пустые значения. Любая переменная неявного цикла в списке ввода определена, даже если в оставшиеся элементы списка ввода засылаются пустые значения.

Заметим, что в записи для ввода, управляемого списком, все пробелы рассматриваются как часть некоторого разделителя значений, кроме следующих случаев:

1) пробелы, входящие в символьную константу;
2) промежуточные пробелы, окружающие действительную или мнимую часть комплексной константы;
3) начальные пробелы в первой записи, считываемой при каждом выполнении оператора ввода, управляемого списком, если за ними сразу не следует запятая или дробная черта.

10.8.1.2. Пример ввода, управляемого списком

```
INTEGER I; REAL X (8); CHARACTER (11) P;
COMPLEX Z; LOGICAL G
    . . .
READ *, I, X, P, Z, G
```

Записи ввода данных:

```
12345,12345,,2*1.5,4*
ISN'T_BOB'S,(123,0),.TEXAS$
```

Результаты:

Переменная	Значение
I	12345
X(1)	12345.0
X(2)	не меняется
X(3)	1.5
X(4)	1.5
X(5)-X(8)	не меняется
P	ISN'T_BOB'S
Z	(123.0,0.0)
G	"истина"

10.8.2. Вывод, управляемый списком

Форма вырабатываемых значений та же, что требуется при вводе, кроме специально оговоренных случаев. Значения (кроме расположенных рядом символьных констант без ограничителей) разделяются одним или несколькими пробелами, либо запятой, до и после которой могут находится пробелы.

Процессор может, если потребуется, начинать новые записи, но при этом внутрь констант, кроме символьных и комплексных, не должны попадать конец записи и пробелы.

Логические константы выводятся как T для значения "истина" и F - для значения "ложь".

Целые константы при выводе вырабатываются под действием описателя редактирования I*w*.

Вещественные константы вырабатываются под действием либо описателя редактирования F, либо описателя редактирования E в зависимости от величины *x* выводимого значения и диапазона $10^{d1} \le x < 10^{d2}$, где *d1* и *d2* - целые значения, зависящие от процессора. Если модуль *x* находится в этом диапазоне, константа вырабатывается с использованием описателя редактирования 0PF*w.d*, в противном случае используется описатель редактирования 1PE*w.dEe*.

При выводе чисел для каждой выводимой числовой константы используются допустимые целые значения *w*, *d* и *e*, зависящие от процессора.

Комплексные константы заключаются в скобки, запятая разделяет действительную и мнимую части, каждая из которых вырабатывается по описанным выше правилам для вещественных констант. Конец записи между запятой и мнимой частью допускается только в том случае, если полная длина константы равна или больше длины всей записи. Пробелы допускаются только между запятой и концом записи и один пробел в начале следующей записи.

Для внутреннего файла, либо файла, открытого без спецификатора DELIM = (9.3.4.9) или со значением спецификатора DELIM, равным NONE, вырабатываемые символьные константы:

1) не обрамляются апострофами или кавычками;
2) не отделяются друг от друга разделителями;
3) имеют во внешнем представлении для каждого внутреннего символа "апостроф" или "кавычки" один символ "апостроф" или "кавычки";
4) имеют в начале каждой записи, которая начинается с продолжения символьной константы из предыдущей записи, вставленный процессором пробел для управления печатью.

Если значением спецификатора DELIM при открытии файла является QUOTE, вырабатываемые символьные константы обрамляются кавычками, перед которыми может находиться *параметр-вида* и символ подчеркивания, все это обрамляется разделителями значений, а внутренние кавычки во внешнем представлении изображаются сдвоенными кавычками.

Если значением спецификатора DELIM при открытии файла является APOSTROPHE, вырабатываемые символьные константы обрамляются апострофами, перед которыми может находиться *параметр-вида* и символ подчеркивания, все это обрамляется разделителями значений, а внутренние апострофы во внешнем представлении изображаются сдвоенными апострофами.

Если два или более последовательных значений при выводе имеют одинаковые значения, процессор имеет режим выработки повторяемых констант в форме *r*c* вместо последовательности идентичных значений.

Форматирование, управляемое списком, не вырабатывает пустые значения и дробные черты в качестве разделителей значений.

Каждая запись вывода начинается с пробела для управления кареткой при печати, кроме случая продолжения символьной константы с ограничителями.

10.9. Форматирование по NAMELIST

Символы в одной или нескольких записях NAMELIST образуют **подпоследовательности имя-значение**, состоящие из имени, за которым следует знак равенства, а за ним одно или несколько значений и разделителей значений. С обеих сторон знака равенства могут находиться пробелы (некоторое количество, в том числе, нулевое). Конец записи, если он не внутри символьной константы, действует так же, как символ пробела. Любая последовательность пробелов везде, кроме символьной константы, воспринимается как один пробел .

Имя может быть любым именем из *списка-объектов-группы-NAMELIST* (5.4).

Каждое значение - это либо пустое значение, либо одна из форм:

c

r*c

r*

где *c* - литеральная константа, а *r* - ненулевая целая литеральная константа без знака и параметра вида.

Форма *r***c* эквивалентна *r* последовательным вхождениям константы *c*, а форма *r** эквивалентна *r* последовательным пустым значениям. Ни одна из этих форм не может содержать промежуточных пробелов, кроме тех, что допускаются внутри константы *c*.

Разделители значений для форматирования по NAMELIST те же, что при форматировании, управляемом списком (10.8).

10.9.1. Ввод по NAMELIST

Вводимая информация для оператора ввода по NAMELIST состоит из следующих элементов:

1) необязательных пробелов;
2) символа &, непосредственно за которым следует то же *имя-группы-NAMELIST*, что задано в операторе ввода по NAMELIST;
3) одного или нескольких пробелов;
4) последовательности из нулевого или более числа подпоследовательностей имя-значение, отделяемых разделителями значений;
5) дробной черты для завершения оператора ввода по NAMELIST.

В каждой подпоследовательности имя-значение имя должно быть именем элемента списка объектов группы NAMELIST, возможно, с индексами, причем имя с индексами не должно быть массивом нулевого размера или секцией массива нулевого размера, или символьной строкой нулевой длины.

Если процессор имеет возможность представления букв верхнего и нижнего регистров, регистр для имени не имеет значения.

10.9.1.1. Имена объектов группы NAMELIST

Каждое имя во вводимых данных должно соответствовать конкретному имени объекта группы NAMELIST. Индексные выражения, выражения для шага и диапазонов подстроки, используемые для обозначения имен объектов группы, должны быть целыми константами, возможно, со знаком и без параметра вида. Если имя объекта группы NAMELIST является именем массива, соответствующая ему вводимая запись может содержать либо имя массива, либо обозначение подобъекта этого массива, используя синтаксис обозначения подобъектов (R602). Если имя объекта группы NAMELIST является именем переменной производного типа, имя во вводимой записи может быть либо именем переменной, либо обозначением одного из ее компонентов, указанного путем дополнения имени переменной именем соответствующего компонента. Допустима дальнейшая квалификация в соответствии с конфигурацией и типом представляемой переменной.

Порядок имен во вводимых записях не обязательно должен соответствовать порядку элементов списка объектов группы NAMELIST. В записях ввода могут не содержаться все имена элементов группы объектов NAMELIST. Статус определенности тех имен из *списка-объектов-группы-NAMELIST*, которые отсутствуют во вводимой записи , остается неизменным. Пробелы могут находиться до и после имени во вводимой записи, но не внутри имени.

10.9.1.2. Значения данных, вводимых по NAMELIST

Данное *c* - любое вводимое значение, разрешенное спецификацией формата для заданного типа, исключение составляет форма вводимого значения, соответствующая элементам списка логического и целого типа (10.9.1.3). Форма вводимого значения должна быть допустимой для типа элемента списка объекта группы NAMELIST. Число и формы вводимых значений, которые могут следовать за знаком равенства в подпоследовательности имя-значение, зависят от формы и типа объекта, представленного именем во вводимой записи. Когда это имя является именем скалярной переменной встроенного типа, за знаком равенства не должно следовать более одного значения. Пробелы никогда не используются как нули, а в константах, кроме символьных и комплексных, пробелы запрещены (10.9.1.3).

Последовательности имя-значение обрабатываются последовательно слева-направо. Имя объекта группы NAMELIST или обозначение подобъекта может появиться в более чем одной последовательности имя-значение.

Когда имя во вводимой записи представляет переменную-массив или переменную производного типа, эффект такой же, как если бы представляемая переменная была развернута в последовательность скалярных элементов встроенного типа таким же образом, как это делается с элементами списка форматного ввода/вывода (9.4.2). Каждое вводимое значение, следующее за знаком равенства, должно быть допустимым для спецификации формата для элемента списка встроенного типа, находящегося в соответствующей позиции развернутой последовательности, кроме случаев, отмеченных в 10.9.1.3. Число значений, следующих за знаком равенства, не должно превышать числа элементов списка в развернутой последовательности, но может быть меньше; в этом последнем случае эффект такой же, как если бы оставшимся элементам списка в развернутой последовательности было поставлено в соответствие нужное количество пустых значений. Например, если имя во вводимой записи является именем целого массива размера 100, за знаком равенства могут следовать максимум 100 значений, каждое из которых есть либо строка цифр, либо пустое значение. Эти значения будут присваиваться элементам массива в стандартном для массива порядке.

Если в качестве разделителя значений встретилась дробная черта, оператор ввода по NAMELIST завершается после выполнения присваивания предыдущего значения. Если остались еще элементы в списке объектов группы NAMELIST, эффект такой, как если бы им были переданы пустые значения.

Последовательные записи NAMELIST читаются с помощью ввода по NAMELIST до тех пор, пока не встретится дробная черта. Остаток записи игнорируется.

10.9.1.3. Элементы списка объектов группы NAMELIST

Если следующий действующий элемент списка ввода имеет тип вещественный, формой ввода для вводимого значения является числовое поле ввода. Числовое поле ввода - это поле, подходящее для описателя редактирования F (10.5.1.2.1), которое при отсутствии в поле ввода десятичной точки предполагает отсутствие цифр дробной части.

Если следующий действующий элемент списка имеет тип комплексный, форма ввода для вводимого значения состоит из левой скобки, за которой следует упорядоченная пара полей числового ввода, разделенных запятой, а затем правая скобка. Первое числовое поле ввода - действительная часть комплексной константы, второе - ее мнимая часть. До и после каждого из полей могут находиться пробелы. Между действительной частью и запятой или между запятой и мнимой частью может находиться конец записи.

Если следующий действующий элемент списка ввода имеет тип логический, форма ввода для вводимого значения не должна содержать среди необязательных символов, разрешенных для описателя редактирования L (10.5.2), ни запятых, ни дробных черт, ни пробелов.

Если следующий действующий элемент списка имеет тип целый, значение во вводимой записи интерпретируется таким образом, как если бы использовался описатель редактирования I*w* с подходящим значением *w*.

Если следующий действующий элемент списка имеет тип символьный, формой ввода является символьная литеральная константа того же вида, что соответствующий элемент списка. Символьные константы могут быть продолжены от конца одной записи в начало следующей, но конец записи не должен попадать ни между сдвоенными апострофами в константе, обрамленной апострофами, ни между сдвоенными кавычками в константе, обрамленной кавычками. Конец записи не добавляет в константу ни пробелов, ни каких-либо иных символов. Константа может занимать столько записей, сколько потребуется. Символы: пробел, запятая, знак равенства и дробная черта могут находиться в символьных константах.

Предположим, что *len* - длина следующего действующего элемента списка, а *w* - длина символьной константы. Если *len* <= *w*, левые *len* символов константы передаются в следующий действующий элемент списка. Если *len* > *w*, константа передается в левые *w* символов следующего действующего элемента списка, а остальные *len*-*w* его символов заполняются пробелами. Заметим, что тот же эффект можно получить при присваивании константы следующему действующему элементу списка посредством символьного оператора присваивания (7.5.1.4).

10.9.1.4. Пустые значения

Пустое значение задается:

1) формой *r* * ;
2) пробелами между двумя последовательными разделителями значений, расположенными после знака равенства;
3) некоторым числом (включая нулевое) пробелов перед первым разделителем значений, следующим за знаком равенства;
4) двумя последовательными отличными от пробела разделителями значений.

Пустое значение не влияет на статус определенности соответствующего элемента списка ввода. Если элемент списка объектов группы NAMELIST определен, он сохранит свое предыдущее значение, если не определен, таковым и остается. Пустое значение нельзя использовать как действительную или мнимую часть комплексной константы, но одно пустое значение может представлять комплексную константу целиком.

Заметим, что конец записи, следующий за любым другим разделителем, с пробелами или без них, не задает пустого значения.

10.9.1.5. Пробелы

Все пробелы в записи, вводимой по NAMELIST, рассматриваются как часть некоторого разделителя значений, кроме следующих:

1) пробелов, включенных в символьную константу;
2) пробелов, окружающих действительную или мнимую часть комплексной константы;
3) начальных пробелов после знака равенства, если за ними не следует запятая или дробная черта;
4) пробелов между именем и знаком равенства.

10.9.1.6. Пример ввода по NAMELIST

```
INTEGER I; REAL X (8); CHARACTER (11) P; COMPLEX Z;
LOGICAL G
NAMELIST / TODAY / G, I, P, Z, X
READ (*, NML = TODAY)
```

Вводимые записи:

```
&TODAY I = 12345, X(1) = 12345, X(3:4) = 2*1.5, I=6,
P = "ISN'T_BOB'S", Z = (123,0)/
```

Результаты:

Переменная	Значение
I	6
X(1)	12345.0
X(2)	не меняется
X(3)	1.5
X(4)	1.5
X(5)-X(18)	не меняется
P	ISN'T_BOB'S
Z	(123.0,0.0)
G	не меняется

10.9.2. Вывод по NAMELIST

Вырабатываемые формы для вывода те же, что требуются при вводе, исключая формы для вещественных, символьных и логических констант. Если процессор может представить буквы и верхнего, и нижнего регистра, выводимое имя находится на верхнем регистре. Все значения, кроме смежных символьных констант без ограничителей, разделяются либо одним или несколькими пробелами, либо запятой, до и после которой могут находиться пробелы.

Если понадобится, процессор может начинать новые записи, но при этом конец записи не должен попадать в имена и константы, кроме символьных и комплексных, а пробелы не должны попадать ни в константы, ни в имена.

10.9.2.1. Редактирование вывода по NAMELIST

Логические выводимые константы - это T для значения "истина" и F - для значения "ложь".

Выводимые константы целого типа получаются под действием описателя редактирования I*w*.

Вещественные константы получаются под действием описателей редактирования F или E, в зависимости от абсолютной величины значения *x* и диапазона $10^{d1} \le x < 10^{d2}$, где *d1* и *d2* - целые значения, зависящие от процессора. Если абсолютная величина *x* находится в этом диапазоне, константа вырабатывается с использованием описателя редактирования 0PF*w.d*, в остальных случаях используется описатель редактирования 1PE*w.dEe*.

При выводе чисел для каждого вывода числовой константы используются соответствующие значения *w*, *d* и *e*, зависящие от процессора.

Комплексные константы заключаются в скобки, запятая разделяет действительную и мнимую части, каждая из которых вырабатывается по приведенным выше правилам для вещественных констант. Конец записи может появляться между запятой и мнимой частью только в том случае, когда константа имеет длину, большую или равную длине всей записи. Пробелы разрешены только между запятой и концом записи и один пробел в начале следующей записи.

Если файл был открыт без спецификатора DELIM = (9.3.4.9) или со значением DELIM, равным NONE, то:

1) вырабатываемые константы не обрамляются апострофами или кавычками;
2) они не отделяются друг от друга разделителями значений;
3) каждый внутренний апостроф (кавычки) во внешнем представлении изображается одним апострофом (кавычками);
4) в начало каждой записи, продолжающей символьную контанту из предыдущей записи, процессор вставляет пробел для управления печатью.

Если файл был открыт со значением DELIM, равным QUOTE, вырабатываемые символьные константы обрамляются кавычками, перед ними могут быть помещены *параметр-вида* и символ подчеркивания, все вместе обрамляется разделителями значений, а внутренние кавычки во внешнем представлении изображаются сдвоенными кавычками.

Если файл был открыт со значением DELIM, равным APOSTROPHE, вырабатываемые символьные константы обрамляются апострофами, перед ними могут быть помещены *параметр-вида* и символ подчеркивания, все вместе обрамляется разделителями значений, а внутренний апостроф во внешнем представлении изображается сдвоенным апострофом.

10.9.2.2. Записи вывода по NAMELIST

Если два или более последовательных значений в массиве в вырабатываемой записи вывода имеют одинаковые значения, процессор имеет режим выдачи повторяемых констант в виде `r*c` вместо последовательности одинаковых значений.

Имя каждого элемента списка объектов группы NAMELIST помещается в выводимую запись, за ним следует знак равенства, а за ним - список значений элементов списка объектов группы NAMELIST; однако, если элемент списка объектов группы NAMELIST является массивом нулевого размера или символьным объектом нулевой длины, в выводимую запись ничего не заносится.

В начале первой выводимой записи форматирование по NAMELIST вырабатывает символ "&" , за которым следует *имя-группы-NAMELIST* для того, чтобы указать, какая конкретно группа данных выводится. Для обозначения конца форматирования по NAMELIST вырабатывается дробная черта.

Пустые значения форматирование по NAMELIST не вырабатывает.

Каждая выводимая запись (кроме продолжения символьной константы с ограничителями) начинается с символа пробела, обеспечивающего управление кареткой при печати записи.

Глава 11. ПРОГРАММНЫЕ ЕДИНИЦЫ

Термины и основные понятия программных единиц были введены в разделе 2.2. Программная единица может быть главной программной единицей, внешней процедурой, модулем или программной единицей - блоком данных.

Данная глава содержит описание всех этих программных единиц за исключением внешних процедур, описанных в главе 12.

11.1. Главная программная единица

Главная программная единица - это программная единица, первым оператором которой не является ни один из операторов: SUBROUTINE, FUNCTION, MODULE или BLOCK DATA.

R1101 *главная-программная-единица* **есть** [*оператор-PROGRAM*]
[*раздел-спецификаций*]
[*раздел-выполняемых-конструкций*]
[*раздел-внутренних-процедур*]
оператор-конца-программы

R1102 *оператор-PROGRAM* **есть** PROGRAM *имя-программы*

R1103 *оператор-конца-программы* **есть** END[PROGRAM[*имя-программы*]]

Ограничение: *раздел-выполняемых-конструкций главной-программной-единицы* не должен включать операторы RETURN и ENTRY.

Ограничение: *имя-программы* может быть включено в *оператор-конца-программы*, только если используется необязательный *оператор-PROGRAM*, и оно должно быть идентично *имени-программы*, специфицированному в этом *операторе-PROGRAM*.

Ограничение: в *разделе-спецификаций* (R204) главной программной единицы не должно быть автоматически размещаемых объектов.

Имя программы является глобальным для выполняемой программы, оно не должно совпадать с именем какой-либо другой программной единицы, внешней процедуры или общего блока в выполняемой программе, либо с каким-либо локальным именем в главной программной единице.

Пример главной программной единицы:

```
PROGRAM ANALYSE
    REAL A,B,C(10,10)          ! Раздел спецификаций
    CALL FIND                  ! Раздел выполняемых конструкций
CONTAINS
    SUBROUTINE FIND            ! Внутренняя процедура
    ...
    END SUBROUTINE FIND
END PROGRAM ANALYSE
```

11.1.1. Спецификации главной программной единицы

Спецификации главной программной единицы не должны включать операторов OPTIONAL, INTENT, ENTRY, PUBLIC и PRIVATE или эквивалентных им атрибутов (5.1.2). Оператор SAVE не оказывает влияния на главную программную единицу.

11.1.2. Раздел выполняемых конструкций главной программной единицы

Последовательность операторов раздела выполняемых конструкций специфицирует действия главной программной единицы во время выполнения

программы. Выполнение выполняемой программы (R201) начинается с первой выполняемой конструкции главной программной единицы.

Главная программная единица не может быть рекурсивной, т.е. на нее нельзя ссылаться из какой-либо программной единицы в этой выполняемой программе (в том числе, из главной программной единицы).

Выполнение выполняемой программы завершается при выполнении *оператора-конца-программы* в главной программной единице или при выполнении оператора STOP в любой программной единице выполняемой программы.

11.1.3. Внутренние процедуры главной программной единицы

Все описания процедур, внутренних по отношению к главной программной единице, должны следовать за оператором CONTAINS. Внутренние процедуры описываются в 12.1.2.2. Главная программная единица называется **объемлющим контекстом** для своих внутренних процедур.

11.2. Внешние процедуры

Внешние процедуры описаны в главе 12.

11.3. Модули

Модуль содержит спецификации и описания, которые должны быть доступны для других программных единиц.

R1104 *модуль* **есть** *оператор-MODULE*
[*раздел-спецификаций*]
[*раздел-модульных-процедур*]
оператор-конца-модуля

R1105 *оператор-MODULE* **есть** MODULE *имя-модуля*

R1106 *оператор-конца-модуля* **есть** END [MODULE [*имя-модуля*]]

Ограничение: если в *операторе-конца-модуля* задано *имя-модуля*, оно должно совпадать с именем в *операторе-MODULE*.

Ограничение: *раздел-спецификаций* модуля не должен содержать *описаний-операторных-функций* и операторов ENTRY и FORMAT.

Ограничение: в *разделе-спецификаций* (R204) модуля не должны находиться автоматически размещаемые объекты.

Имя модуля в выполняемой программе является глобальным и не может совпадать ни с именем какой-либо другой программной единицы, внешней процедуры или общего блока в выполняемой программе, ни с каким-либо локальным именем в самом модуле.

Заметим, что хотя описания операторных функций и операторы ENTRY и FORMAT запрещены в разделе спецификаций модуля, они могут появиться в разделе спецификаций модульных процедур, содержащихся в этом модуле.

Заметим, что модуль является объемлющим контекстом для модульных процедур (12.1.2.2), которые он содержит, и поэтому объекты модуля доступны модульным процедурам через связь по объемлющему контексту.

11.3.1. Ссылка на модуль

Оператор USE, содержащий имя модуля, является **ссылкой на модуль**. Во время обработки ссылки на модуль общедоступные объекты этого модуля становятся доступными. Модуль не может ссылаться сам на себя, ни прямо, ни косвенно.

Возможность доступа (общедоступный или приватный) к спецификациям и описаниям модуля для контекстной области программной единицы, ссылающейся на модуль, управляется как модулем, так и контекстной областью, ссылающейся на него. В модуле для управления доступом к объектам модуля вне этого модуля используются опе-

раторы PUBLIC и PRIVATE (5.2.3), эквивалентные им атрибуты (5.1.2.2) и оператор PRIVATE в описании производного типа (4.4.1).

Для дальнейшего ограничения доступа к общедоступным объектам модуля в контекстной области, содержащей ссылку на модуль, можно использовать необязательную спецификацию ONLY в операторе USE.

11.3.2. Оператор USE и связь по использованию

Оператор USE обеспечивает средства доступа контекстной области к именованным объектам данных, производным типам, блокам интерфейса, процедурам, обобщенным идентификаторам (12.3.2.1) и группам NAMELIST в модуле. В этом случае говорят, что объекты контекстной области **связаны по использованию** с объектами из модуля. Используемые объекты имеют атрибуты, специфицированные в модуле.

R1107 *оператор-USE* — **есть** USE *имя-модуля* [,*список-переименований*]
или USE *имя-модуля*, ONLY: [*список-объектов-ONLY*]

R1108 *переименование* — **есть** *локальное-имя* => *используемое-имя*

R1109 *объект-ONLY* — **есть** *доступный-идентификатор*
или [*локальное-имя* =>] *используемое-имя*

Ограничение: каждый *доступный-идентификатор* должен быть общедоступным объектом в модуле.

Ограничение: каждое *используемое-имя* должно быть общедоступным объектом в модуле.

Если в *список-переименований* или в *список-объектов-ONLY* входит *локальное-имя*, это имя является локальным для объекта, специфицированного с помощью *используемого-имени*, в противном случае локальным именем является *используемое-имя*.

Оператор USE без спецификации ONLY обеспечивает доступ ко всем общедоступным объектам данного модуля.

Оператор USE со спецификацией ONLY обеспечивает доступ только к тем объектам, которые входят в *список-объектов-ONLY* в качестве *доступного-идентификатора* или *используемого-имени*.

В контекстной области может находиться несколько операторов USE, ссылающихся на данный модуль. Если один из них без спецификации ONLY, все общедоступные объекты в модуле являются доступными, а все *списки-переименований* и *списки-объектов-ONLY* интерпретируются как сцепленные в один *список-переименований*. Если же все операторы USE имеют спецификацию ONLY, доступными являются только те объекты, которые названы в одном или нескольких *списках-объектов-ONLY*, т.е. все *списки-объектов-ONLY* интерпретируются как сцепленные в один *список-объектов-ONLY*.

Если в контекстной области доступны два или более обобщенных интерфейса с одним и тем же именем, с одним и тем же знаком операции (либо оба являются присваиваниями), они интерпретируются как один обобщенный интерфейс. Два или более доступных объектов (отличных от обобщенных интерфейсов) могут иметь одно имя при условии, что в этой контекстной области нет ссылок на это имя. В остальных случаях локальное имя любого объекта, ставшего доступным посредством оператора USE, должно отличаться от локальных имен всех других объектов, доступных в контекстной области через операторы USE или иным способом. Заметим, что доступ к объекту может быть осуществлен более чем через одно локальное имя.

Локальное имя объекта, ставшего доступным через оператор USE, не может входить ни в один оператор спецификации, который мог бы вызвать изменение атрибутов (5.1.2) объекта в контекстной области, содержащей оператор USE; допускается только включение такого объекта в операторы PUBLIC или PRIVATE в контекстной области модуля. Заметим, что это запрещает включение такого локального имени в операторы COMMON или EQUIVALENCE, но разрешает включать его в списки объектов группы NAMELIST. Появление такого локального имени в операторе PUBLIC в модуле делает объект, доступный через оператор USE, общедоступным объектом этого модуля. Если имя находится в модуле в операторе PRIVATE, объект не является общедоступным объектом модуля. Если же

локальное имя не входит ни в оператор PUBLIC, ни в оператор PRIVATE, подразумевается значение атрибута доступности по умолчанию (5.2.3) в этой контекстной области.

Примеры.

Оператор

```
USE STATS_LIB
```

обеспечивает доступ ко всем общедоступным объектам в модуле STATS_LIB.

```
USE MATH_LIB; USE STATS_LIB, SPROD => PROD
```

В этом примере становятся доступными все общедоступные объекты модулей MATH_LIB и STATS_LIB. Если MATH_LIB содержит объект с именем PROD, он доступен под своим собственным именем, тогда как объект PROD из STATS_LIB доступен под именем SPROD.

```
USE STATS_LIB, ONLY: YPROD; USE STATS_LIB, ONLY : PROD
```

В этом фрагменте становятся доступными общедоступные объекты YPROD и PROD из модуля STATS_LIB.

Оператор

```
USE STATS_LIB, ONLY : YPROD; USE STATS_LIB
```

делает доступными все общедоступные объекты из `STATS_LIB`.

11.3.3. Примеры использования модулей

11.3.3.1. Идентичные общие блоки

Общий блок и все связанные с ним операторы спецификаций могут быть помещены в модуль с именем, например, MY_COMMON, тогда доступ к нему обеспечивается оператором USE вида:

USE MY_COMMON

который обеспечивает доступ ко всему модулю без каких-либо переименований. Это обеспечивает идентичность всех экземпляров этого общего блока. Модуль MY_COMMON мог бы содержать несколько общих блоков.

11.3.3.2. Глобальные данные

Модуль может содержать только объекты данных, например:

```
MODULE DATA_MODULE
    SAVE
    REAL A(10), B, C(20,20)
    INTEGER :: I=0
    INTEGER, PARAMETER :: J=10
    COMPLEX D(J,J)
END MODULE
```

Заметим, что объекты данных, которые стали глобальными таким способом, могут иметь любую комбинацию типов данных.

Доступ к некоторым из них может быть осуществлен посредством оператора USE со спецификацией ONLY, например:

USE DATA_MODULE, ONLY: A,B,D

а доступ к ним ко всем можно организовать с помощью следующего оператора USE:

USE DATA_MODULE

Доступ ко всем данным с некоторым переименованием во избежание конфликта имен можно осуществить так:

USE DATA_MODULE, AMODULE => A, DMODULE => D

11.3.3.3. Производные типы

Производный тип может быть описан в модуле и стать доступным для нескольких программных единиц. Например:

```
MODULE SPARSE
  TYPE NONZERO
    REAL A
    INTEGER I,J
  END TYPE
END MODULE
```

Описывается тип, состоящий из одного вещественного и двух целых компонентов, для хранения числового значения ненулевого элемента матрицы, номеров его столбца и строки.

11.3.3.4. Глобальные динамически размещаемые массивы

Часто в программе нужны большие глобальные динамически размещаемые массивы, размеры которых до выполнения программы неизвестны. Простая форма такой программы:

```
PROGRAM GLOBAL_WORK
  CALL CONFIGURE_ARRAYS          ! Производит соответствующие размещения
  CALL COMPUTE                   ! Использует массивы в вычислениях
END PROGRAM GLOBAL_WORK

MODULE WORK_ARRAYS               ! Пример набора рабочих массивов
  INTEGER N
  REAL, ALLOCATABLE, SAVE :: A(:),B(:,:),C(:,:,:)
END MODULE WORK_ARRAYS

SUBROUTINE CONFIGURE_ARRAYS      ! Размещение рабочих массивов
  USE WORK_ARRAYS
  READ (*, *) N
  ALLOCATE (A(N), B(N,N), C(N,N,2*N))
END SUBROUTINE CONFIGURE_ARRAYS

SUBROUTINE COMPUTE
  USE WORK_ARRAYS
  ...                            ! Вычисления, включающие массивы A, B и C
END SUBROUTINE COMPUTE
```

Как правило, доступ к рабочим массивам нужен во многих процедурах, и все эти процедуры должны содержать оператор:

```
USE WORK_ARRAYS
```

11.3.3.5. Библиотеки процедур

Блоки интерфейса для внешних процедур библиотеки можно накапливать в модуле. Это позволяет использовать необязательные аргументы и аргументы с ключевыми словами и дает возможность статически проверять обращения. Для разных приложений могут быть сформированы разные версии с использованием общеупотребительных ключевых слов для каждого приложения.

Пример такого библиотечного модуля:

```
MODULE LIBRARY_LLS
    INTERFACE
        SUBROUTINE LLS (X, A, F, FLAG)
          REAL X(:, :)
                                  ! SIZE в следующем операторе - встроенная функция
          REAL, DIMENSION (SIZE (X, 2)):: A, F
          INTEGER FLAG
        END SUBROUTINE LLS
        ...
    END INTERFACE
    ...
END MODULE LIBRARY_LLS
```

Этот модуль позволяет вызвать подпрограмму LLS следующим образом:

```
USE LIBRARY_LLS
...
CALL LLS (X = ABC, A = D, F = XX, FLAG = IFLAG)
...
```

11.3.3.6. Расширение семантики знаков операций

Чтобы расширить смысл знака встроенной операции, можно поместить в модуль блок интерфейса, специфицирующий этот знак операции в спецификации OPERATOR оператора INTERFACE. Например, операцию // можно расширить так, чтобы она выполняла конкатенацию двух объектов производного типа, употребляемых как символьные строки переменной длины, а знак операции "+" можно расширить для спецификации сложения матриц для типа MATRIX или для сложения объектов типа INTERVAL в интервальной арифметике.

Модуль может содержать несколько таких блоков интерфейса. Можно определить операцию с помощью внешней функции (Фортрана или некоторого другого языка), а ее интерфейс поместить в модуль.

11.3.3.7. Абстракция данных

Модуль может инкапсулировать описание производного типа и все процедуры, представляющие операции над значениями этого типа. В С.11.5 приведен пример операций над множествами.

11.3.3.8. Переименованные общедоступные объекты

Иногда может понадобиться переименование объектов, доступных через оператор USE. При этом следует быть внимательными, если модули, на которые они ссылаются, тоже содержат операторы USE. Следующий пример иллюстрирует возможности переименования в операторе USE:

```
MODULE J; REAL JX, JY, JZ; END MODULE
MODULE K
    USE J, ONLY: KX => JX, KY => JY
                                    ! KX и KY - локальные имена в модуле K
    REAL KZ                         ! KZ - локальное имя в модуле K
    REAL JZ                         ! JZ - локальное имя в модуле K
END MODULE
```

```
PROGRAM RENAME
    USE J; USE K
                    ! Объект JX модуля J доступен под именами JX и KX
                    ! Объект JY модуля J доступен под именами JY и KY
                    ! Объект KZ модуля K доступен под именем KZ
                    ! Объекты JZ в модуле J и в модуле K - разные объекты
                    ! и на них нельзя ссылаться
. . .
END PROGRAM RENAME
```

11.4. Программные единицы - блоки данных

Программная единица - блок данных используется для задания начальных значений для объектов данных из именованных общих блоков.

R1110 *блок-данных* **есть** *оператор-BLOCK-DATA*
[*раздел-спецификаций*]
оператор-конца-блока-данных

R1111 *оператор-BLOCK-DATA* **есть** BLOCK DATA [*имя-блока-данных*]

R1112 *оператор-конца-блока-данных* **есть** END [BLOCK DATA [*имя-блока-данных*]]

Ограничение: *имя-блока-данных* может быть включено в *оператор-конца-блока-данных* только в том случае, если оно есть в *операторе-BLOCK-DATA*, и тогда эти имена должны быть идентичны.

Ограничение: *раздел-спецификаций блока-данных* может содержать только операторы USE, операторы объявления типа, IMPLICIT, PARAMETER, описания производного типа и следующие операторы спецификации: COMMON, DATA, DIMENSION, EQUIVALENCE, INTRINSIC, POINTER, SAVE и TARGET.

Ограничение: операторы объявления типа в *разделе-спецификаций блока-данных* не должны содержать спецификаторов атрибутов ALLOCATABLE, EXTERNAL, INTENT, OPTIONAL, PRIVATE или PUBLIC.

Если объект в именованном общем блоке начально определен, все объекты, занимающие единицы памяти в последовательности памяти общего блока, должны быть специфицированы, даже если они не все начально определены. В одном блоке данных могут быть начально определены объекты из нескольких именованных общих блоков.

В блоке данных могут быть начально определены только объекты (не указатели) из именованного общего блока. Заметим, что объекты, связанные с объектом из общего блока, рассматриваются как находящиеся в этом общем блоке.

Один и тот же именованный общий блок может быть специфицирован не более чем в одном блоке данных выполняемой программы.

В выполняемой программе не может быть более одного неименованного блока данных.

Пример блока данных:

```
BLOCK DATA WORK
    COMMON /WRKCOM/ A, B, C (10,10)
    DATA A /1.0/, B /2.0/, C /100*0.0/
END BLOCK DATA WORK
```

ИЗДАТЕЛЬСТВО "ФИНАНСЫ И СТАТИСТИКА"

предлагает учебные пособия

В.В. Подбельский, С.С. Фомин
ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ Си: – 600 с.

Пособие состоит из трех разделов. Первый раздел – это полный курс программирования на стандартном языке Си. Рассматриваются все средства языка Си, не зависящие от реализации, существующие в компиляторах на ПЭВМ, рабочих станциях и мощных сетевых серверах. Второй раздел посвящен выполнению программ в разных операционных системах. Третий раздел – это практикум по языку Си. По каждой теме практикума приведены методические рекомендации и варианты заданий для организации индивидуальной работы в учебной группе и классе.

Для студентов и преподавателей вузов, а также для желающих освоить самостоятельно программирование на языке Си.

В.В. Подбельский
ЯЗЫК Си++. – 560 с.

Рассматриваются синтаксис, семантика и методы использования объектно-ориентированного языка Си++. Приводятся программы, иллюстрирующие процедурные возможности и особенности языка. В отличие от других книг, в которых язык Си++ излагается как расширение своего предшественника языка Си, в данном учебном пособии он рассматривается как самостоятельный язык программирования. Поэтому для его изучения желательно только, чтобы пользователь владел основами информатики и навыками программирования на любом алгоритмическом языке.

Для широкого круга студентов и специалистов, желающих самостоятельно изучить язык Си++.

Е.М. Воробьев
ВВЕДЕНИЕ В СИСТЕМУ "МАТЕМАТИКА". – 264 с.

Рассмотрены вопросы использования пакета "Математика" в качестве символьного, графического и численного калькулятора, а также языка программирования высокого уровня, позволяющего программировать в функциональном стиле. Использование пакета освобождает пользователей от многих рутинных математических операций.

Книги Вы можете приобрести в издательстве по адресу:

101000, Москва, ул. Покровка, 7

(метро "Китай-город", выход на ул. Маросейка)

Тел.: **(095) 925-35-02, 923-18-68, 923-80-42**

Факс **(095) 925-09-57**

Глава 12. ПРОЦЕДУРЫ

Понятие процедуры было введено в 2.2.3. Данная глава содержит полное описание процедур. Действия, специфицированные процедурой, производятся, когда процедура вызывается при выполнении ссылки на нее. Ссылка может идентифицировать в качестве фактических аргументов объекты, которые во время выполнения ссылки на процедуру связываются с соответствующими формальными аргументами в описании процедуры.

12.1. Классификации процедур

Процедуры классифицируются в соответствии с формой ссылки на них и способом их описания.

12.1.1. Классификация процедур по ссылке

Описание процедуры указывает, является ли процедура функцией или подпрограммой. Ссылка на функцию либо явно появляется в выражении в качестве первичного, либо подразумевается в виде определяемой операции, входящей в выражение. Ссылкой на подпрограмму является оператор CALL или определяемый оператор присваивания (7.5.1.3).

Процедура классифицируется как **поэлементная**, если она является встроенной процедурой, на которую можно ссылаться поэлементно (12.4.3, 12.4.5).

12.1.2. Классификация процедур по способу описания

Процедура может быть либо встроенной процедурой, внешней процедурой, модульной процедурой, внутренней процедурой, формальной процедурой, либо операторной функцией.

12.1.2.1. Встроенные процедуры

Процедура, которая предоставляется как неотъемлемая часть процессора, является **встроенной процедурой**.

12.1.2.2. Внешние, внутренние и модульные процедуры

Внешней процедурой является процедура, которая описывается либо с помощью внешней процедуры Фортрана, либо средствами, отличными от Фортрана.

Внутренней процедурой является процедура, описанная посредством внутренней процедуры Фортрана. Внутренние процедуры могут находиться в главной программной единице, во внешней или модульной процедуре. Внутренние процедуры не могут находиться в других внутренних процедурах. Внутренние процедуры имеют тот же вид, что и внешние, за исключением следующего: имя внутренней процедуры не является глобальным, во внутренней процедуре не должно быть операторов ENTRY, имя внутренней процедуры не может быть аргументом, связанным через аргумент с формальной процедурой (12.4.1.2) и внутренняя процедура имеет доступ к объектам своей объемлющей области через связь по объемлющему контексту.

Модульная процедура - это процедура, описанная как модульная.

Если процедура содержит один или несколько операторов ENTRY, она определяет по одной процедуре на каждый оператор ENTRY и одну процедуру для оператора FUNCTION или SUBROUTINE.

12.1.2.2.1. Связь по объемлющему контексту

Внутренние и модульные процедуры и описания производных типов имеют доступ к именованным объектам своей объемлющей области через **связь по объемлющему контексту**. Эти объекты получают те же имена и атрибуты, которые они имеют внутри объемлющей области. К ним относятся переменные, константы, процедуры, включая интерфейсы, производные типы, параметры типов, компоненты производных типов и группы NAMELIST.

Если имя объекта, доступного через связь по объемлющему контексту, совпадает с именем объекта из объемлющей области, последний становится недоступным. Имя, которое находится в контекстной области как *внешнее-имя* в операторе EXTERNAL, является глобальным именем, и любой объект из объемлющей области, имеющий это имя как необобщенное, является недоступным. Имя, которое появилось в контекстной области как

1) имя типа в описании производного типа;
2) имя функции в операторе FUNCTION, в описании операторной функции, в объявлении объекта в операторе объявления типа;
3) имя подпрограммы в операторе SUBROUTINE;
4) имя входа в операторе ENTRY;
5) имя объекта в объявлении объекта в операторе объявления типа, в операторе POINTER, в операторе SAVE или в операторе TARGET;
6) именованная константа в описании именованной константы в операторе PARAMETER;
7) имя массива в операторе ALLOCATABLE или в операторе DIMENSION;
8) имя переменной, являющейся объектом общего блока в операторе COMMON;
9) имя переменной, полностью или частично инициализированной в операторе DATA;
10) имя объекта, полностью или частично эквивалентного в операторе EQUIVALENCE;
11) имя формального аргумента в операторе FUNCTION, в операторе SUBROUTINE, в операторе ENTRY или в описании операторной функции;
12) имя результата в операторе FUNCTION или в операторе ENTRY;
13) имя встроенной процедуры в операторе INTRINSIC;
14) имя группы NAMELIST в операторе NAMELIST;
15) обобщенное-имя в обобщенном-спецификаторе в операторе-INTERFACE,

является именем локального объекта, и любой объект из объемлющей области, который имеет это имя как необобщенное, является недоступным. Локальные (14.1.2) объекты процедуры недоступны для ее объемлющей области.

Если объект из объемлющей области недоступен только потому, что одноименный локальный объект инициализирован (полностью или частично) в операторе DATA, этот локальный объект нельзя определять и нельзя на него ссылаться до оператора DATA.

Отметим, что тело интерфейса не имеет доступа к именованным объектам через связь по объемлющему контексту, но может иметь доступ к объектам через связь по использованию (11.3.2).

Если процедура получает доступ к указателю через связь по объемлющему контексту, связь указателя с адресатом, которая является текущей во время вызова процедуры, остается таковой внутри процедуры. В процедуре эта связь указателя может быть изменена. Когда исполнение процедуры завершается, текущая связь указателя сохраняется, кроме тех случаев, когда завершение делает неопределенным адресат (п.4 из 14.7.6). В этих случаях статус связанности указателя, связанного по объемлющему контексту, становится неопределенным.

12.1.2.2.2. Связь по использованию и по объемлющему контексту

Объемлющая процедура и внутренняя процедура могут содержать одинаковые или разные объекты, связанные по использованию, что иллюстрирует приведенный ниже пример.

```
MODULE B; REAL BX, Q; INTEGER IX, JX; END MODULE
MODULE C; REAL CX; END MODULE
MODULE D; REAL DX, DY, DZ; END MODULE
MODULE E; REAL EX, EY, EZ; END MODULE
MODULE F; REAL FX; END MODULE
MODULE G; USE F; REAL GX; END MODULE

PROGRAM A
USE B; USE C; USE D
   . . .
CONTAINS
   SUBROUTINE INNER_PROC (Q)
      USE C                    ! Не обязателен
      USE B, ONLY: BX          ! Доступные объекты: BX, IX, JX, если
                               !     нет других объектов IX или JX,
                               !     которые доступны в INNER_PROC
                               ! Q локально в INNER_PROC, т.к. Q -
                               !     формальный аргумент
      USE D, X => DX           ! Доступные объекты: DX, DY, DZ
                               ! X - локальное имя для DX в INNER_PROC
                               ! X и DX обозначают один объект,
                               !     если нет другого DX, который
                               !     является локальным объектом
                               !     в INNER_PROC
      USE E, ONLY: EX          ! EX доступно в INNER_PROC, но не в
                               !   главной программной единице A
                               ! EY и EZ недоступны в INNER_PROC и A
      USE G                    ! FX и GX доступны в INNER_PROC
      . . .
   END SUBROUTINE INNER_PROC
END PROGRAM A
```

Заметим, что поскольку главная программная единица A содержит оператор

```
USE B
```

все объекты модуля B, кроме Q, доступны в INNER_PROC, несмотря на то, что в INNER_PROC есть оператор

```
USE B, ONLY: BX
```

Оператор USE с ключевым словом ONLY означает, что данный конкретный оператор делает доступным только названный объект, а не то, что только эта переменная из модуля доступна в данной контекстной области.

12.1.2.3. Формальные процедуры

Формальный аргумент, который специфицируется как процедура или входит в ссылку на процедуру, является **формальной процедурой.**

12.1.2.4. Операторные функции

Функция, определяемая отдельным оператором, является **операторной функцией** (12.5.4).

12.2. Характеристики процедур

Характеристики процедуры - это классификация процедуры как функции или подпрограммы, характеристики ее аргументов и характеристики значения ее результата в случае функции.

12.2.1. Характеристики формальных аргументов

Каждый формальный аргумент - это либо формальный объект данных или формальная процедура, либо звездочка (указатель альтернативного возврата). Формальный аргумент - не звездочка может быть специфицирован с атрибутом OPTIONAL. Этот атрибут означает, что данный формальный аргумент не обязательно связан с фактическим для каждой конкретной ссылки на процедуру.

12.2.1.1. Характеристики формальных объектов данных

Характеристиками формального объекта данных являются его тип, параметры типа (если они есть), конфигурация, назначение (5.1.2.3, 5.2.1), является ли он обязательным (5.1.2.6, 5.2.2) и является ли он указателем (5.1.2.7, 5.2.7) или адресатом (5.1.2.8, 5.2.8). Если параметром типа объекта или границей массива является выражение, которое зависит от значения или атрибутов другого объекта, точная зависимость от других объектов является характеристикой. Если конфигурация, размер или длина символьного являются неопределенными, это тоже характеристика.

12.2.1.2. Характеристики формальных процедур

Характеристиками формальной процедуры является наличие у нее явного интерфейса (12.3.1), характеристики ее как процедуры, если интерфейс явный, и является ли она обязательным аргументом (5.1.2.6, 5.2.2).

12.2.1.3. Характеристики формального аргумента - звездочки

Формальный аргумент - звездочка характеристик не имеет.

12.2.2. Характеристики результатов функции

Характеристиками результата функции являются его тип, параметры типа (если они есть), размерность и является ли он указателем. Если результат функции - массив, но не указатель, его конфигурация является характеристикой. Если параметр типа или граница массива не являются константным выражением, точная их зависимость от объектов в выражении тоже является характеристикой. Если длина символьного объекта является неопределенной, это тоже характеристика.

12.3. Интерфейс процедуры

Интерфейс процедуры определяет формы ссылок, посредством которых можно ее вызвать. Интерфейс состоит из характеристик процедуры, ее имени, имени и характеристик каждого формального аргумента и обобщенных идентификаторов процедуры (если они есть). Характеристики процедуры фиксированы, но остальные составляющие интерфейса могут быть разными в разных контекстных областях.

12.3.1. Явный и неявный интерфейсы

Если процедура доступна в контекстной области, ее интерфейс в этой контекстной области является либо **явным**, либо **неявным**. Интерфейс внутренних, встроенных и модульных процедур всегда является явным в такой контекстной области. Интерфейс рекурсивной подпрограммы или рекурсивной функции с отдельным результатом

является явным в той процедуре, которая его определяет. Интерфейс операторной функции всегда неявный. Для внешней или формальной процедуры в контекстной области, отличной от ее собственной, интерфейс является явным, если для процедуры описан или доступен блок интерфейса (12.3.2.1), иначе интерфейс будет неявным. Например, подпрограмма LLS из 11.3.3.5 имеет явный интерфейс.

12.3.1.1. Явный интерфейс

Процедура должна иметь явный интерфейс, если выполняется хотя бы одно из следующих условий.

1. Ссылка на процедуру появляется:
   а) с ключевым словом аргумента (12.4.1);
   б) как определяемое присваивание (только подпрограммы);
   в) в выражении как определяемая операция (только функции);
   г) как ссылка посредством ее обобщенного имени (12.3.2.1);
2. Процедура имеет
   а) необязательный формальный аргумент;
   б) результат, значением которого является массив (только функции);
   в) формальный аргумент, который является массивом неопределенной конфигурации, указателем или адресатом;
   г) результат, для которого значение параметра длины не является ни неопределенной длиной , ни константой (только для символьных функций);
   д) результат, который является указателем (только для функций).

12.3.2. Спецификация интерфейса процедуры

Интерфейс для внешней, внутренней, формальной или модульной процедуры специфицируется с помощью операторов FUNCTION, SUBROUTINE или ENTRY и операторов спецификации для формальных аргументов и результата функции. Эти операторы могут находиться в описании процедуры, в блоке интерфейса, либо и там, и там (кроме ENTRY, который запрещен в блоке интерфейса). Заметим, что в блоке интерфейса не должно быть внутренних процедур.

12.3.2.1. Блок интерфейса процедуры

R1201 *блок-интерфейса*	**есть** *оператор-INTERFACE* [*тело-интерфейса*]... [*оператор-модульных-процедур*]... *оператор-конца-интерфейса*
R1202 *оператор-INTERFACE*	**есть** INTERFACE [*обобщенная-спецификация*]
R1203 *оператор-конца-интерфейса*	**есть** END INTERFACE
R1204 *тело-интерфейса*	**есть** *оператор-FUNCTION* [*раздел-спецификаций*] *оператор-конца-функции* **или** *оператор-SUBROUTINE* [*раздел-спецификаций*] *оператор-конца-подпрограммы*
R1205 *оператор-модульных-процедур*	**есть** MODULE PROCEDURE *список-имен-процедур*
R1206 *обобщенная-спецификация*	**есть** *обобщенное-имя* **или** OPERATOR (*знак-определяемой-операции*) **или** ASSIGNMENT (=)

Ограничение: *тело-интерфейса* не должно содержать операторов ENTRY, FORMAT, DATA и описаний операторной функции.

Ограничение: спецификация MODULE PROCEDURE разрешена только если *блок-интерфейса* имеет *обобщенную-спецификацию* и объемлющий контекст его является модулем или имеет доступ к модулю через связь по использованию. Каждое *имя-процедуры* должно быть именем модульной процедуры, доступной в объемлющей области.

Ограничение: *блок-интерфейса* не может находиться в программной единице - блоке данных.

Ограничение: *блок-интерфейса* не может включать *тело-интерфейса* для процедуры, описываемой данной программной единицей.

Описание внешней или модульной процедуры задает **специфический интерфейс** для определяемой им процедуры. Такой специфический интерфейс является явным для модульных процедур и неявным - для внешних. **Тело интерфейса** в блоке интерфейса определяет явный интерфейс для существующей внешней или формальной процедуры. Если имя в заголовке процедуры в блоке интерфейса совпадает с именем формального аргумента процедуры, содержащей этот блок интерфейса, блок интерфейса объявляет, что этот формальный аргумент должен быть формальной процедурой с указанным интерфейсом; иначе это имя объявляется как имя внешней процедуры с указанным интерфейсом.

Тело интерфейса специфицирует все характеристики процедуры, которые должны согласовываться с характеристиками из описания процедуры. Заметим, что имена формальных аргументов могут отличаться. Раздел спецификаций тела интерфейса может задавать атрибуты или определять значения для объектов данных, которые не определяют характеристики процедуры. Такие спецификации никакого эффекта не имеют. Блок интерфейса не должен содержать операторов ENTRY, но можно специфицировать интерфейс для ENTRY, используя имя входа в качестве имени процедуры в теле интерфейса. Процедура должна иметь в данной контекстной области не более одного явного специфического интерфейса.

Пример блока интерфейса без обобщенной спецификации:

```
INTERFACE
    SUBROUTINE EXT1 (X, Y, Z)
        REAL, DIMENSION (100, 100) :: X, Y, Z
    END SUBROUTINE EXT1

    SUBROUTINE EXT2 (X, Z)
        REAL X
        COMPLEX (KIND = 4) Z (2000)
    END SUBROUTINE EXT2

    FUNCTION EXT3 (P, Q)
        LOGICAL EXT3
        INTEGER P (1000)
        LOGICAL Q (1000)
    END FUNCTION EXT3
END INTERFACE
```

Этот блок интерфейса задает явные интерфейсы для трех внешних процедур: EXT1, EXT2 и EXT3. Все эти процедуры могут использовать вызовы с ключевыми словами, например:

```
EXT3 (Q = P_MASK (N+1 : N+1000), P = ACTUAL_P)
```

Блок интерфейса с обобщенной спецификацией задает **обобщенный интерфейс** для всех процедур в этом блоке. Если объемлющая область является модулем или ссылается на модуль через связь по использованию, в спецификации MODULE PROCEDURE перечисляются те модульные процедуры, имеющие этот обобщенный

интерфейс, которые либо описаны в этом модуле, либо доступны через оператор USE. Характеристики модульных процедур не определяются в блоках интерфейса, а передаются из описаний модульных процедур. Обобщенный интерфейс всегда явный.

На процедуру всегда можно сослаться через ее специфический интерфейс, а также через обобщенный, если он есть. Обобщенное имя, знак определяемой операции или равенства в *обобщенной спецификации* являются ***обобщенным идентификатором*** для всех процедур в блоке интерфейса. Правила, по которым должны различаться две процедуры с одинаковым обобщенным идентификатором, приведены в 14.1.2.3. Они обеспечивают, чтобы конкретный обобщенный вызов был применим не более чем к одной специфической процедуре.

Обобщенное имя специфицирует одно имя для ссылки на все имена процедур в блоке интерфейса. Обобщенное имя может совпадать с любым из имен процедур в блоке интерфейса либо с любым доступным обобщенным именем.

Пример обобщенного процедурного интерфейса:

```
INTERFACE SWITCH
    SUBROUTINE INT_SWITCH (X, Y)
        INTEGER, INTENT (INOUT) :: X, Y
    END SUBROUTINE INT_SWITCH

    SUBROUTINE REAL_SWITCH (X, Y)
        REAL, INTENT (INOUT) :: X, Y
    END SUBROUTINE REAL_SWITCH

    SUBROUTINE COMPLEX_SWITCH (X, Y)
        COMPLEX, INTENT (INOUT) :: X, Y
    END SUBROUTINE COMPLEX_SWITCH
END INTERFACE
```

На все эти три процедуры (INT_SWITCH, REAL_SWITCH и COMPLEX_SWITCH) можно ссылаться через обобщенное имя SWITCH так же, как и через специфическое имя. Например, ссылка на INT_SWITCH может иметь следующий вид:

```
CALL SWITCH (MAX_VAL, LOC_VAL)          ! MAX_VAL и LOC_VAL - целого типа
```

12.3.2.1.1. Определяемые операции

Если в обобщенной спецификации задана спецификация OPERATOR, все процедуры в блоке интерфейса должны быть функциями, на которые можно ссылаться как на определяемые операции (12.4). В случае функции с двумя аргументами подразумевается инфиксная нотация знака бинарной операции. В случае функции с одним аргументом подразумевается префиксная нотация знака операции. Нельзя задавать спецификацию OPERATOR для функции без аргументов или для функции, имеющей более двух аргументов. Формальные аргументы должны быть формальными объектами данных без атрибута OPTIONAL, но с атрибутом INTENT(IN), а результат функции не должен иметь неопределенной длины (для символьных данных). Если операция является *знаком-встроенной-операции* (R310), число аргументов функции должно соответствовать использованию этой операции как встроенной.

Определяемая операция трактуется как ссылка на функцию. Для унарной определяемой операции операнд соответствует формальному аргументу функции; для бинарной операции левый операнд соответствует первому аргументу функции, правый операнд - второму аргументу.

Пример использования обобщенной спецификации:

```
INTERFACE OPERATOR (*)
    FUNCTION BOOLEAN_AND (B1, B2)
        LOGICAL, INTENT (IN) :: B1 (:), B2 (SIZE(B1))
        LOGICAL :: BOOLEAN_AND (SIZE(B1))
    END FUNCTION BOOLEAN_AND
END INTERFACE
```

Это позволяет, например, такую запись:

```
SENSOR (1:N) * ACTION (1:N)
```

как альтернативу вызову функции:

```
BOOLEAN_AND(SENSOR(1:N),ACTION(1:N))        ! SENSOR и ACTION имеют
                                            ! тип LOGICAL
```

Данная определяемая операция, как в случае с обобщенными именами, применима более чем к одной функции, и тогда она является обобщенной точно так же, как в случае с обобщенными процедурными именами. Для знаков встроенных операций свойство обобщенности включает и представляемую им встроенную операцию. Поскольку обе формы для знаков операций отношения имеют одинаковую интерпретацию (7.3), расширение одной из форм (такой, как «=») равносильно определению обеих форм («=» и .LE.).

12.3.2.1.2. Определяемые присваивания

Если в операторе INTERFACE задана спецификация ASSIGNMENT, все процедуры в блоке интерфейса должны быть подпрограммами, на которые можно ссылаться как на определяемые присваивания (7.5.1.3). Каждая такая подпрограмма должна иметь два формальных аргумента, которые не могут быть необязательными. Первый аргумент может иметь атрибут INTENT(OUT) или INTENT(INOUT), а второй аргумент должен иметь атрибут INTENT(IN). Определяемое присваивание трактуется, как ссылка на подпрограмму, с левой частью в качестве первого аргумента и заключенной в скобки правой частью в качестве второго аргумента. Обобщенная спецификация ASSIGNMENT означает, что присваивание расширено или переопределено, если обе части равенства относятся к одному производному типу.

Пример использования обобщенной спецификации ASSIGNMENT:

```
INTERFACE ASSIGNMENT (=)
    SUBROUTINE BIT_TO_NUMERIC (N, B)
        INTEGER, INTENT (OUT) :: N
        INTEGER, INTENT (IN) :: B (:)
    END SUBROUTINE BIT_TO_NUMERIC

    SUBROUTINE CHAR_TO_STRING (S, C)
        USE STRING_MODULE                   ! Содержит описание типа STRING
        TYPE (STRING),INTENT (OUT)::S       ! Строка переменной длины
        CHARACTER (*), INTENT (IN) C
    END SUBROUTINE CHAR_TO_STRING
END INTERFACE
```

Примеры присваивания:

```
KOUNT=SENSOR(J:K)      ! CALL BIT_TO_NUMERIC (KOUNT, (SENSOR(J:K)))
NOTE = '89AB'          ! CALL CHAR_TO_STRING (NOTE, ('89AB'))
```

12.3.2.2. Оператор EXTERNAL

Оператор EXTERNAL задает список имен, обладающих атрибутом EXTERNAL. Имя с атрибутом EXTERNAL представляет внешнюю или формальную процедуру или программную единицу - блок данных. Задание имени внешней или формальной процедуры в операторе EXTERNAL позволяет использовать его в качестве фактического аргумента.

R1207 *оператор-EXTERNAL* **есть** EXTERNAL *список-внешних-имен*

Каждое *внешнее-имя* должно быть именем внешней процедуры, формального аргумента или программной единицы - блока данных.

Появление имени формального аргумента в операторе EXTERNAL означает, что данный аргумент является формальной процедурой.

Появление в операторе EXTERNAL имени, которое не является именем формального аргумента, означает, что это имя является именем внешней процедуры или программной единицы - блока данных. Если имя внешней или формальной процедуры используется как фактический аргумент, оно либо должно быть включено в данной контекстной области в оператор EXTERNAL или в оператор объявления типа с атрибутом EXTERNAL, либо должно быть объявлено как процедура с помощью блока интерфейса в этой контекстной области. Появление в операторе EXTERNAL имени встроенной процедуры приводит к тому, что это имя становится именем некоторой внешней процедуры, а встроенная процедура с этим именем в данной контекстной области недоступна.

Разрешено только одно вхождение имени во все операторы EXTERNAL одной контекстной области.

Пример оператора EXTERNAL:

```
SUBROUTINE SUB (FOCUS)
  EXTERNAL FOCUS
```

12.3.2.3. Оператор INTRINSIC

Оператор INTRINSIC специфицирует список имен, имеющих атрибут INTRINSIC. Имя с атрибутом INTRINSIC представляет встроенную процедуру (глава 13). Атрибут INTRINSIC позволяет использовать в качестве фактического аргумента имя, представляющее специфическую встроенную функцию.

R1208 *оператор-INTRINSIC* **есть** INTRINSIC *список-имен-встроенных-процедур*

Ограничение: каждое *имя-встроенной-процедуры* должно быть именем встроенной процедуры.

Появление имени в операторе INTRINSIC означает, что это имя является именем встроенной процедуры. Включение в оператор INTRINSIC обобщенного имени встроенной функции (13.10) не вызывает утраты именем его родовых свойств.

Если специфическое имя (13.12) встроенной функции используется в качестве фактического аргумента, оно должно быть включено в этой контекстной области в оператор INTRINSIC или в оператор описания типа с атрибутом INTRINSIC.

Разрешено только одно вхождение имени во все операторы INTRINSIC в одной контекстной области. Заметим, что в одной контекстной области одно и то же имя не может входить и в оператор EXTERNAL, и в оператор INTRINSIC.

12.3.2.4. Спецификация неявного интерфейса

В программной единице, где интерфейс функции является неявным, тип и параметры типа результата функции задаются неявной или явной спецификацией типа для имени функции. Тип, параметры типа и конфигурация формальных аргументов при обращении из контекстной области, где интерфейс процедуры является неявным, должны быть такими, чтобы фактические аргументы согласовывались с характеристиками формальных.

12.4. Ссылка на процедуру

Форма ссылки на процедуру зависит от интерфейса процедуры, но не зависит от способов описания этой процедуры. Формы ссылок на процедуры:

R1209 *ссылка-на-функцию* **есть** *имя-функции* ♦
♦ ([*список-специф-фактических-аргументов*])

Ограничение: *список-специф-фактических-аргументов* в ссылке на функцию не должен включать *специф-альтерн-возврата*.

R1210 *оператор-CALL* **есть** CALL *имя-подпрограммы* ♦
♦ ([([*список-специф-фактических-аргументов*])])

Кроме того, на функцию можно ссылаться как на определяемую операцию, а на подпрограмму - как на определяемое присваивание.

12.4.1. Список фактических аргументов

R1211 *специф-фактического-аргумента* **есть** [*ключевое-слово* =] ♦
♦ *фактический-аргумент*

R1212 *ключевое-слово* **есть** *имя-формального-аргумента*

R1213 *фактический-аргумент* **есть** *выражение*
или *переменная*
или *имя-процедуры*
или *специф-альтерн-возврата*

R1214 *специф-альтерн-возврата* **есть** **metka*

Ограничение: если интерфейс процедуры в данной контекстной области неявный, конструкция "*ключевое-слово* =" не разрешена.

Ограничение: "*ключевое-слово*=" может быть опущено в *специф-фактического-аргумента* только тогда, когда опущено "*ключевое-слово*=" в каждом из предшествующих *специф-фактического-аргумента* в списке аргументов.

Ограничение: каждое *ключевое-слово* должно быть именем формального аргумента в явном интерфейсе процедуры.

Ограничение: *имя-процедуры* в качестве *фактического-аргумента* не может быть именем внутренней процедуры или операторной функции и не может быть обобщенным именем процедуры (12.3.2.1, 13.1).

Ограничение: метка в *специф-альтерн-возврата* должна быть меткой оператора-адресата для оператора ветвления, находящегося в одной контекстной области с оператором CALL.

В любой ссылке на подпрограмму или на функцию список фактических аргументов идентифицирует соответствие между представленными фактическими аргументами и формальными аргументами процедуры. При отсутствии ключевого слова фактический аргумент связывается с тем формальным аргументом, который занимает соответствующую позицию в списке формальных аргументов, т.е. первый фактический аргумент будет связан с первым формальным аргументом, второй фактический аргумент будет связан со вторым формальным аргументом и так далее. Если ключевое слово присутствует, фактический аргумент будет связан с тем формальным аргументом, имя которого совпадает с ключевым словом (используя имена формальных аргументов из того блока интерфейса, который доступен в контекстной области, содержащей ссылку на процедуру). Ровно один фактический аргумент должен быть связан с каждым обязательным формальным аргументом. Не больше одного фактического аргумента может быть связано с каждым необязательным формальным аргументом. Каждый фактический аргумент должен быть связан с формальным аргументом. Например, процедура

```
SUBROUTINE SOLVE (FUNCT, SOLUTION, METHOD, STRATEGY, PRINT)
    INTERFACE
        FUNCTION FUNCT(X)
            REAL FUNCT,X
        END FUNCTION FUNCT
    END INTERFACE
    REAL SOLUTION
    INTEGER, OPTIONAL :: METHOD, STRATEGY, PRINT
    . . .
```

может быть вызвана следующим образом:

```
CALL SOLVE (FUN, SOL, PRINT=6)
```

при условии, что ее интерфейс является явным; когда интерфейс задается с помощью блока интерфейса, именем последнего аргумента должно быть PRINT.

12.4.1.1. Аргументы, связанные с формальными объектами данных

Если формальный аргумент является формальным объектом данных, связанный с ним фактический аргумент должен быть выражением или объектом данных того же типа. Значение параметра вида для фактического аргумента должно согласовываться с параметром вида формального аргумента. Значение параметра длины фактического аргумента нестандартного символьного типа должно согласовываться с длиной формального аргумента. Если формальный аргумент - массив неопределенной конфигурации стандартного символьного типа, значение параметра длины фактического аргумента должно согласовываться с длиной формального аргумента.

Если скалярный фактический аргумент имеет стандартный символьный тип, длина *len* формального аргумента может быть меньше или равна длине фактического аргумента. Тогда формальный аргумент связывается с левыми *len* символами фактического аргумента. Если формальный аргумент - массив стандартного символьного типа, ограничение на длину относится не к каждому элементу массива, а к массиву целиком. Длина элемента в формальном аргументе - массиве может отличаться от длины элемента в связанном с ней фактическом аргументе - массиве, элементе массива или подстроке элемента массива, но при этом формальный массив не должен выходить за пределы фактического.

Кроме случая поэлементной ссылки на процедуру (12.4.3, 12.4.5), каждый элемент фактического аргумента со значением-массивом или элемент последовательности в случае связи по последовательности (12.4.1.4) связывается с элементом формального массива, занимающим ту же позицию в обычном порядке элементов массива (6.2.2.2). Заметим, что для связи по памяти стандартного символьного типа интерпретация элемента приведена в 12.4.1.4.

Если формальный аргумент является указателем, фактический аргумент тоже должен быть указателем, а их типы, значения параметров типа и размерности должны согласовываться.

При вызове процедуры формальный аргумент - указатель получает статус связанности фактического аргумента. Если фактический аргумент в текущий момент связан, формальный аргумент связывается с этим же адресатом. Статус связанности может быть изменен во время выполнения процедуры.

Когда выполнение процедуры заканчивается, статус связанности формального аргумента становится неопределенным, если он связан с формальным аргументом процедуры, имеющим атрибут TARGET, либо с адресатом, который становится неопределенным (14.7.6); после этого статус связанности фактического аргумента становится таким же, как у формального аргумента.

Если фактический аргумент имеет атрибут TARGET, все связанные с ним указатели при вызове процедуры не связываются с соответствующим формальным аргументом, а сохраняют связь с фактическим аргументом. Если формальный аргумент имеет атрибут

TARGET, все связанные с ним указатели становятся неопределенными при завершении исполнения процедуры.

Если фактический аргумент - скаляр, соответствующий формальный аргумент должен быть скаляром, если только фактический аргумент не является элементом массива неопределенной конфигурации или указателя на массив, либо подстрокой элемента такого массива. Если на процедуру ссылаются через обобщенное имя либо как на определяемую операцию или определяемое присваивание, размерности фактических аргументов и соответствующих формальных аргументов должны согласовываться.

Если формальный аргумент имеет атрибут INTENT (OUT) или INTENT (INOUT), фактический аргумент должен быть определимым. Если формальный аргумент имеет атрибут INTENT (OUT), соответствующий фактический аргумент становится неопределенным во время установления связи.

Если фактический аргумент - секция массива с векторным индексом, формальный аргумент не определим и не может иметь атрибута INTENT (OUT) или INTENT (INOUT).

Если формальный аргумент - массив неопределенной конфигурации, фактический аргумент не должен быть массивом неопределенного размера, обозначением элемента массива или его подстроки, скаляром (включая обозначение элемента массива или подстроки элемента массива).

Скалярный формальный аргумент может быть связан только со скалярным фактическим аргументом.

12.4.1.2. Аргументы, связанные с формальными процедурами

Если формальный аргумент является формальной процедурой, связываемый с ним фактический аргумент должен быть специфическим именем внешней, формальной или встроенной процедуры либо модульной процедуры. Из встроенных процедур разрешены только те, которые перечислены в 13.12 и не помечены звездочкой. Если специфическое имя является также и обобщенным именем, с формальным аргументом связывается только специфическая процедура.

Если интерфейс процедуры явный, характеристики связанной фактической процедуры должны совпадать с характеристиками формальной процедуры (12.2).

Если интерфейс формальной процедуры неявный и либо явно задан тип формальной процедуры, либо процедура вызывается как функция, тогда на формальную процедуру нельзя ссылаться как на подпрограмму, а фактический аргумент должен быть функцией или формальной процедурой.

Если интерфейс формальной процедуры неявный, а ссылка на процедуру является ссылкой на подпрограмму, фактический аргумент должен быть подпрограммой или формальной процедурой.

12.4.1.3. Аргументы, связанные с указателями альтернативного возврата

Если формальный аргумент - звездочка (12.5.2.3), связанный с ним фактический аргумент должен быть указателем альтернативного возврата. Метка в указателе альтернативного возврата должна идентифицировать выполняемую конструкцию в контекстной области, содержащей ссылку на процедуру.

12.4.1.4. Связь по последовательности

Фактический аргумент представляет **последовательность элементов**, если он является выражением-массивом, обозначением элемента массива, обозначением подстроки элемента массива. Если фактический аргумент является выражением-массивом, последовательность элементов состоит из элементов массива, расположенных в принятом для массива порядке. Если фактический аргумент является обозначением элемента массива, последовательность элементов состоит из этого элемента и всех элементов, следующих за ним в обычном для массива порядке.

Если фактический аргумент - стандартного символьного типа и является выражением-массивом, элементом массива или обозначением подстроки элемента массива, последовательность элементов образуется из символьных единиц памяти, начиная с первой символьной единицы памяти фактического аргумента и до конца массива. Символьные единицы памяти для подстроки элемента массива рассматриваются как элементы массива, образованные последовательными группами символьных единиц памяти, каждая из которых имеет длину символьного формального аргумента. Заметим, что некоторые элементы в последовательности элементов могут быть образованы единицами памяти из разных элементов исходного массива.

Фактический аргумент, который представляет последовательность элементов и соответствует формальному аргументу - массиву, связывается по последовательности с формальным аргументом, если формальный аргумент является массивом явной конфигурации или неопределенного размера. Размерность и конфигурация фактического аргумента не обязательно должны быть теми же, что у формального аргумента, но число элементов формального аргумента не должно превышать числа элементов в последовательности элементов фактического аргумента. Если формальный аргумент неопределенного размера, число его элементов равно числу элементов в последовательности элементов фактического аргумента.

12.4.2. Ссылка на функцию

Функция вызывается во время вычисления выражения с помощью ссылки на функцию или посредством определяемой операции (7.1.3). Во время вызова вычисляются все выражения фактических аргументов, затем аргументы связываются и затем выполняется функция. Когда завершится выполнение функции, значение ее результата становится доступным для использования в выражении, вызвавшем функцию. Характеристики результата функции (12.2.2) определяются интерфейсом функции.

12.4.3. Поэлементная ссылка на встроенную функцию

Ссылка на поэлементную встроенную функцию является **поэлементной ссылкой**, если один или несколько ее фактических аргументов являются массивами и все аргументы-массивы имеют одинаковую конфигурацию. Результат имеет ту же конфигурацию, что аргументы-массивы, а значение каждого элемента результата получается вычислением функции, использующей скалярные аргументы и соответствующие элементы аргументов-массивов. Например, если X и Y - массивы с конфигурацией (m, n),

```
MAX (X, 0.0, Y)
```

является выражением-массивом с конфигурацией (m,n), элементы которого имеют значения:

```
MAX (X(i,j), 0.0, Y(i,j)), i=1,2,...,m, j=1,2,...n
```

12.4.4. Ссылка на подпрограмму

Подпрограмма вызывается при выполнении оператора CALL или оператора определяемого присваивания (7.5.1.3). Когда подпрограмма вызывается, сначала вычисляются все выражения фактических аргументов, затем связываются аргументы, а потом выполняется подпрограмма. Когда заканчиваются действия, специфицированные подпрограммой, выполнение оператора CALL или оператора определяемого присваивания тоже завершается. Если среди аргументов оператора CALL есть один или несколько спецификаторов альтернативного возврата, управление может быть передано на один из указанных ими операторов в зависимости от действий, специфицированных подпрограммой.

12.4.5. Поэлементная ссылка на встроенную подпрограмму

Ссылка на поэлементную встроенную подпрограмму является поэлементной ссылкой, если все фактические аргументы, соответствующие формальным аргументам с

атрибутом INTENT (OUT) и INTENT (INOUT), являются массивами одинаковой конфигурации, а остальные фактические аргументы согласуются с ними. Значения элементов массивов, соответствующих формальным аргументам с атрибутом INTENT (OUT) и INTENT (INOUT), такие, какие были бы получены, если бы подпрограмма применялась отдельно к соответствующим элементам каждого аргумента.

12.5. Определение процедуры

12.5.1. Определение встроенной процедуры

Встроенные процедуры определяются как составная часть процессора. Соответствующий стандарту процессор должен включать встроенные процедуры, описанные в главе 13, но может включать и другие. Однако соответствующая стандарту программа не должна использовать встроенные процедуры, отличные от тех, что описаны в главе 13.

12.5.2. Процедуры, определяемые средствами Фортрана

Когда вызывается процедура, определяемая средствами Фортрана, создается и выполняется экземпляр (12.5.2.4) процедуры. Выполнение начинается с первой выполняемой конструкции, следующей за оператором FUNCTION, SUBROUTINE или ENTRY, задающим имя, по которому вызывается процедура, либо с оператора END, если нет других выполняемых конструкций.

12.5.2.1. Действие атрибута INTENT в процедурах

Атрибут INTENT для формальных объектов данных ограничивает способы их употребления в процедуре. Формальные объекты данных, имеющие атрибут INTENT (IN), нельзя определять или переопределять в процедуре. Формальный объект данных, имеющий атрибут INTENT (OUT), начально в подпрограмме не определен. На формальный объект данных с атрибутом INTENT (INOUT) можно ссылаться, его можно определять. Формальный объект данных, для которого не задан атрибут назначения, подчиняется ограничениям того объекта данных, который является связанным с ним фактическим аргументом; т.е. если фактический аргумент определен, на формальный объект данных можно ссылаться, и формальный аргумент можно определять, если фактический аргумент является определимым.

12.5.2.2. Процедура-функция

Процедура-функция - это процедура, в которой первым оператором является оператор FUNCTION.

R1215 *процедура-функция* **есть** *оператор-FUNCTION*
[*раздел-спецификаций*]
[*раздел-выполняемых-конструкций*]
[*раздел-внутренних-процедур*]
оператор-конца-функции

R1216 *оператор-FUNCTION* **есть** [*префикс*] FUNCTION *имя-функции* ♦
♦ ([*список-имен-формальных-аргументов*]) ♦
♦ [RESULT (*имя-результата*)]

Ограничение: если задано RESULT, *имя-функции* не должно входить ни в один оператор спецификации в контекстной области функции.

R1217 *префикс* **есть** *специф-типа* [RECURSIVE]
или RECURSIVE [*специф-типа*]

R1218 *оператор-конца-функции* **есть** END [FUNCTION[*имя-функции*]]

Ограничение: если задано ключевое слово RESULT, *имя-результата* не должно совпадать с *именем-функции*.

Ограничение: ключевое слово FUNCTION обязательно для *оператора-конца-функции* в модульной или внутренней функции.

Ограничение: внутренняя функция не должна содержать операторов ENTRY.

Ограничение: во внутренней функции не должно быть *раздела-внутренних-процедур*.

Ограничение: если в *операторе-конца-функции* есть *имя-функции*, оно должно соответствовать *имени-функции* в *операторе-FUNCTION*.

Тип и параметры типа (если они есть) результата функции, определяемой процедурой-функцией, могут быть заданы посредством спецификации типа в операторе FUNCTION или включением имени результирующей переменной в оператор объявления типа в разделе спецификаций функции. Нельзя их задавать одновременно обоими способами. Если же тип не задан ни одним из этих способов, его можно определить по правилам неявной типизации, действующим внутри процедуры-функции. Если функция имеет в качестве результата массив или указатель, это должно быть задано в спецификациях имени функции внутри тела функции. Спецификации атрибутов результата функции, атрибутов ее формальных аргументов вместе с информацией в заголовке процедуры определяют интерфейс функции (12.3).

Если функция прямо или косвенно вызывает себя или функцию, описанную в ней же оператором ENTRY, ключевое слово **RECURSIVE** должно присутствовать. Оно должно присутствовать и в тех случаях, когда функция, описанная через ENTRY, прямо или косвенно вызывает саму себя или другую функцию, описанную в операторе FUNCTION или в другом операторе ENTRY в этой же процедуре.

Именем функции является *имя-функции*.

Если специфицирован префикс RESULT, именем результирующей переменной функции является *имя-результата*, ее характеристиками являются характеристики результата функции и все вхождения имени функции в операторы *раздела-выполняемых-конструкций* в контекстной области являются рекурсивными вызовами функции. Если же RESULT отсутствует, именем результирующей переменной является *имя-функции*, и все вхождения этого имени в операторы *раздела-выполняемых-конструкций* контекстной области являются ссылками на результирующую переменную. Значение результирующей переменной при завершении выполнения функции - это и есть значение, выдаваемое функцией. Если результат функции был объявлен как указатель, конфигурация выдаваемого функцией значения определяется конфигурацией результата функции в момент окончания выполнения функции. Если результирующая переменная не является указателем, ее значение должно быть в функции определено. Если результат функции - указатель, функция должна либо связать результирующую переменную-указатель с адресатом, либо определить статус размещенности этого указателя как несвязанный.

Если заданы и RECURSIVE, и RESULT, интерфейс функции внутри процедуры-функции определяется как явный.

Пример рекурсивной функции:

```
RECURSIVE FUNCTION CUMM_SUM (ARRAY) RESULT(C_SUM)
    REAL ARRAY (:), C_SUM (SIZE (ARRAY))
    INTENT (IN) ARRAY     ! Характеристики CUMM_SUM те же, что у C_SUM
    INTEGER N
    N = SIZE (ARRAY)
    IF (N.LE.1) THEN
        C_SUM = ARRAY
    ELSE
        N = N/2
        C_SUM (:N) = CUMM_SUM (ARRAY (:N))
        C_SUM (N+1:) = C_SUM(N) + CUMM_SUM (ARRAY (N+1:))
    END IF
END FUNCTION CUMM_SUM
```

12.5.2.3. Подпрограмма

R1219 *процедура-подпрограмма* **есть** *оператор-SUBROUTINE*
[*раздел-спецификаций*]
[*раздел-выполняемых-конструкций*]
[*раздел-внутренних-процедур*]
оператор-конца-подпрограммы

R1220 *оператор-SUBROUTINE* **есть** [RECURSIVE] SUBROUTINE ♦
♦ *имя-подпрограммы* [([*список-формальных-аргументов*])]

R1221 *формальный-аргумент* **есть** *имя-формального-аргумента*
или *

R1222 *оператор-конца-подпрограммы* **есть** END [SUBROUTINE [*имя-подпрограммы*]]

Ограничение: в *операторе-конца-подпрограммы* внутренней или модульной подпрограммы слово SUBROUTINE должно присутствовать.

Ограничение: внутренняя подпрограмма не должна содержать операторов ENTRY.

Ограничение: во внутренней подпрограмме не должно быть *раздела-внутренних-процедур*.

Ограничение: если в *операторе-конца-подпрограммы* есть *имя-подпрограммы*, оно должно совпадать с *именем-подпрограммы* в *операторе-SUBROUTINE*.

Если подпрограмма прямо или косвенно вызывает самое себя или подпрограмму, определенную в ней же оператором ENTRY, ключевое слово RECURSIVE должно присутствовать. Оно должно присутствовать и в тех случаях, когда подпрограмма, описанная через ENTRY, прямо или косвенно вызывает самое себя или другую подпрограмму, описанную оператором SUBROUTINE или другим оператором ENTRY в этой же процедуре.

Если задано ключевое слово RECURSIVE, интерфейс подпрограммы определяется в ней как явный.

Именем подпрограммы является *имя-подпрограммы*.

12.5.2.4. Экземпляры процедуры

Когда вызывается функция или подпрограмма, определяемая процедурой Фортрана, создается **экземпляр** этой процедуры.

Каждый экземпляр имеет независимую последовательность выполнения и независимый набор формальных аргументов и несохраняемых локальных объектов данных. Если внутренняя процедура или операторная функция, содержащиеся в процедуре, вызываются непосредственно из экземпляра процедуры или из внутренней процедуры или операторной функции, имеющих доступ к объектам этого экземпляра, созданный экземпляр внутренней процедуры или операторной функции тоже имеет доступ к объектам этого экземпляра объемлющей области.

Все другие объекты являются общими для всех экземпляров процедуры. Например, значение сохраняемого объекта данных, входящего в один экземпляр процедуры, может быть определено либо в предыдущем экземпляре, либо при инициализации в операторе DATA или в операторе объявления типа.

12.5.2.5. Оператор ENTRY

Оператор ENTRY позволяет ссылаться на процедуру, начиная с какого-то конкретного выполняемого оператора процедуры, в которой находится оператор ENTRY.

R1223 *оператор-ENTRY* **есть** ENTRY *имя-входа* ♦
♦ [([*список-формальных-аргументов*])] ♦
♦ [RESULT (имя-результата)]]

Ограничение: если задан префикс RESULT, *имя-входа* не должно входить ни в один оператор спецификации в контекстной области процедуры-функции.

Ограничение: *оператор-ENTRY* может находиться только во *внешней-процедуре* или в *модульной-процедуре. Оператор-ENTRY* не должен находиться внутри *выполняемой-конструкции* .

Ограничение: префикс RESULT может присутствовать только в операторе ENTRY, расположенном в *процедуре-функции* .

Ограничение: в процедуре, содержащей оператор ENTRY, *имя-входа* не должно находиться в качестве формального аргумента в операторах FUNCTION или SUBROUTINE и в других операторах ENTRY и не может входить в оператор EXTERNAL или INTRINSIC.

Ограничение: формальный-аргумент может быть указателем альтернативного возврата только в случае, если оператор ENTRY находится в подпрограмме.

Ограничение: если задано RESULT, *имя-результата* не должно совпадать с *именем-входа.*

В процедуре может быть один или более операторов ENTRY.

Если оператор ENTRY находится в процедуре-функции, эта процедура определяет дополнительную функцию. Именем этой функции является *имя-входа,* а ее результирующей переменной - *имя-результата* или *имя-входа,* если *имя-результата* отсутствует. Характеристики результата функции задаются с помощью спецификаций результирующей переменной. Формальные аргументы функции - те, которые специфицированы в операторе ENTRY. Если характеристики результата функции, указанной в операторе ENTRY, совпадают с характеристиками результата функции, указанной в операторе FUNCTION, их результирующие переменные идентифицируют одну переменную, хотя имена у них могут быть разными. Иначе они связываются по памяти и должны все быть скалярами без атрибута POINTER и все стандартного символьного типа с одинаковыми длинами либо все они должны быть скалярами без атрибута POINTER и все одного из следующих типов: стандартный целый, стандартный вещественный, вещественный двойной точности, стандартный комплексный или стандартный логический.

Если в операторе ENTRY есть спецификация RESULT, а в операторе FUNCTION - RECURSIVE, интерфейс функции, определяемой оператором ENTRY, является в процедуре-функции явным.

Если оператор ENTRY находится в подпрограмме, эта подпрограмма определяет еще дополнительную подпрограмму. Именем этой подпрограммы является *имя-входа,* а формальными аргументами - аргументы, специфицированные в операторе ENTRY.

Если в операторе SUBROUTINE есть спецификация RECURSIVE, интерфейс подпрограммы, определяемой оператором ENTRY, является в программной единице явным.

Порядок, количество, типы, параметры типа и имена формальных аргументов в операторе ENTRY могут отличаться от соответствующих характеристик формальных аргументов оператора FUNCTION или SUBROUTINE в программной единице, содержащей этот оператор ENTRY.

Поскольку оператор ENTRY определяет дополнительную функцию или подпрограмму, ссылаются на нее так же, как на любую другую функцию или подпрограмму (12.4).

Имя, входящее в оператор ENTRY в качестве формального аргумента, может входить в выполняемые операторы, предшествующие ENTRY, только в том случае, когда оно входит в оператор FUNCTION или SUBROUTINE, либо в оператор ENTRY, предшествующий данному выполняемому оператору.

Имя, входящее в оператор ENTRY в качестве формального аргумента, может входить в выражение операторной функции только в том случае, когда оно является еще и формальным аргументом операторной функции, входит в оператор FUNCTION или SUBROUTINE, либо в оператор ENTRY, предшествующий описанию операторной функции.

Если формальный аргумент входит в выполняемый оператор, выполнение последнего разрешается во время выполнения ссылки на функцию или подпрограмму только в том случае, если формальный аргумент входит в список формальных аргументов той процедуры, имя которой указано в ссылке.

Если формальный аргумент используется в выражении спецификации для задания границы массива или длины символьного объекта, появление объекта в операторе, выполняемом во время ссылки на процедуру, разрешается только в том случае, когда формальный аргумент входит в список формальных аргументов той процедуры, имя которой указано в ссылке, и этот аргумент присутствует (12.5.2.8).

Контекстная область, содержащая ссылку на процедуру, определяемую в операторе ENTRY, может иметь доступ к телу интерфейса для процедуры. Заголовок процедуры для тела интерфейса должен быть оператором FUNCTION, если это вход в функцию, либо оператором SUBROUTINE, если это вход в подпрограмму.

Ключевое слово RECURSIVE в операторе ENTRY не используется. Его наличие или отсутствие в начальном операторе FUNCTION или SUBROUTINE влияет на то, может ли процедура, определяемая оператором ENTRY, ссылаться на самое себя.

12.5.2.6. Оператор RETURN

R1224 *оператор-RETURN* **есть** RETURN [*целое-скалярное-выражение*]

Ограничение: *оператор-RETURN* должен находиться в контекстной области процедуры-функции или процедуры-подпрограммы.

Ограничение: *целое-скалярное-выражение* разрешено только в контекстной области подпрограммы.

Выполнение **оператора RETURN** завершает выполнение экземляра процедуры, в котором он находится. Если в нем есть выражение и его значение n находится между 1 и числом звездочек в списке формальных аргументов, то после оператора CALL, вызвавшего подпрограмму, управление передается на оператор, идентифицированный n-m спецификатором альтернативного возврата в списке фактических аргументов. Если же выражения нет или его значение не попадает в требуемый диапазон, передачи управления на альтернативный возврат нет.

Выполнение *оператора-конца-функции* и *оператора-конца-подпрограммы* эквивалентно выполнению оператора RETURN без выражения.

12.5.2.7. Оператор CONTAINS

R1225 *оператор-CONTAINS* **есть** CONTAINS

Оператор CONTAINS отделяет тело главной программной единицы, модуля или процедуры от любой внутренней или модульной процедуры, которую они могут содержать. Оператор CONTAINS является невыполняемым.

12.5.2.8. Ограничения на отсутствующие формальные аргументы

Формальный аргумент **присутствует** в экземпляре процедуры, если он связан с фактическим аргументом, который в вызывающей процедуре либо также является присутствующим формальным аргументом, либо вообще не является формальным аргументом. Формальный аргумент, который не является необязательным, должен присутствовать. Необязательный формальный аргумент, который отсутствует, подчиняется следующим ограничениям:

1) если это формальный объект данных, на него нельзя ссылаться и нельзя его определять;
2) если это формальная процедура, ее нельзя вызывать;
3) его нельзя использовать как фактический аргумент, соответствующий обязательному формальному аргументу, кроме случая его использования в качестве аргумента встроенной функции PRESENT;
4) подобъект такого формального аргумента не должен использоваться как фактический аргумент, соответствующий необязательному формальному аргументу.

Этот формальный аргумент может использоваться как фактический аргумент, соответствующий необязательному формальному аргументу, который затем тоже рассматривается как не связанный с фактическим аргументом.

12.5.2.9. Ограничения на объекты, связанные с формальными аргументами

Пока объект связан с формальным аргументом, действуют следующие ограничения.

1. Недопустимы любые действия, которые повлияли бы на значение или доступность объекта или какой-либо его части иначе, чем через формальный аргумент. В примере:

```
SUBROUTINE OUTER
   REAL, POINTER :: A(:)
   ...
   ALLOCATE (A(1:N))
   ...
   CALL INNER(A)
   ...
CONTAINS
   SUBROUTINE INNER(B)
      REAL :: B(:)
      ...
   END SUBROUTINE INNER

   SUBROUTINE SET (C,D)
      REAL, INTENT (OUT) :: C
      REAL, INTENT (IN) :: D
      C=D
   END SUBROUTINE SET
END SUBROUTINE OUTER
```

оператор присваивания, такой, как

```
A(1) = 1.0
```

недопустим во время выполнения INNER, поскольку он может изменить A без использования B, но такие операторы, как

```
B(1) = 1.0
```

или

```
CALL SET(B(1),1.0)
```

были бы допустимы. Аналогично, оператор

```
DEALLOCATE(A)
```

недопустим, поскольку влияет на доступность A без использования B. В этом случае

```
DEALLOCATE(B)
```

тоже запрещено, но было бы допустимо, если бы B было об'явлено с атрибутом POINTER.

Заметим, что если есть частичное или полное перекрытие между фактическими аргументами, связанными с двумя разными аргументами одной и той же процедуры, перекрывающиеся их части не должны определяться, переопределяться или становиться неопределенными во время выполнения процедуры. Например, в операторе

```
CALL SUB(A(1:5),A(3:9))
```

A(3:5) не должно определяться, переопределяться или становиться неопределенным через первый формальный аргумент, поскольку эта часть аргумента связана со вторым формальным аргументом, и не должно определяться, переопределяться или становиться неопределенным через второй формальный аргумент, т.к. эта часть аргумента связана с первым формальным

аргументом. A(1:2) остается определимым через первый, а A(6:9) - через второй формальный аргумент.

Это ограничение применимо и к адресатам указателей. В примере:

```
REAL, DIMENSION (10), TARGET :: A
REAL, DIMENSION (:), POINTER :: B, C
B => A (1:5)
C => A (3:9)
CALL SUB (B, C)
```

B(3:5) нельзя определять, т.к. эта часть аргумента связана со вторым формальным аргументом. C(1:3) нельзя определять, т.к. эта часть аргумента связана с первым формальным аргументом. A(1:2) [она же B(1:2)] остается определимой через первый формальный аргумент, а A(6:9) [она же C(4:7)] остается определимой через второй формальный аргумент.

Заметим, что поскольку формальный аргумент, объявленный с атрибутом IN, нельзя использовать для изменения связанного с ним фактического аргумента, последний остается постоянным во время выполнения процедуры.

2. Если какая-либо часть объекта определяется через формальный аргумент, тогда в любой момент выполнения процедуры — до и после определения — ссылаться на нее можно только через этот формальный аргумент. В примере:

```
MODULE DATA
   REAL :: W,X,Y,Z
END MODULE DATA

PROGRAM MAIN
   USE DATA
   ...
   CALL INIT(X)
   ...
END PROGRAM MAIN

SUBROUTINE INIT(V)
   USE DATA
   ...
   READ(*,*) V
   ...
END SUBROUTINE INIT
```

на переменную X нельзя ссылаться прямо во время выполнения INIT, т.к. она определяется через формальный аргумент V. На X можно (косвенно) сослаться через V. На W, Y и Z можно ссылаться непосредственно. Конечно же, после завершения выполнения INIT прямые ссылки на X разрешены.

12.5.3. Определение процедур способом, отличным от Фортрана

Отличные от Фортрана способы описания процедур зависят от процессора. Ссылка на такие процедуры оформляется так же, как если бы они были описаны внешними процедурами Фортрана. Описание процедуры, написанной не на Фортране, не должно входить в программную единицу Фортрана , а программная единица Фортрана не должна содержать описания процедуры, написанной не на Фортране. Интерфейс к такой процедуре может быть задан в блоке интерфейса.

12.5.4. Операторная функция

Операторная функция - это функция, описанная с помощью одного оператора.

R1226 *описание-операторной-функции* **есть** *имя-функции* ♦
♦ ([*список-имен-формальных-аргументов*]) = *скалярное-выражение*

Ограничение: *скалярное-выражение* может быть образовано только из констант (литеральных и именованных), ссылок на скалярные переменные и элементы массива, ссылок на функции и формальные процедуры-функции и из встроенных операций. Если в *скалярном-выражении* появилась ссылка на другую операторную функцию, она должна быть ранее описана в контекстной области и при этом не должна быть именем описываемой операторной функции.

Ограничение: именованные константы в *скалярном-выражении* должны быть предварительно объявлены в этой контекстной области, либо должны быть доступны через связь по объемлющему контексту или по использованию. Если в *скалярном-выражении* есть элементы массива, массив-основа должен быть предварительно объявлен как массив в этой контекстной области, либо должен быть доступен через связь по объемлющему контексту или по использованию. Если скалярная переменная, элемент массива, ссылка на функцию или формальную функцию типизируются по правилам неявной типизации, все их последующие вхождения в операторы объявления типа должны согласовываться с неявно заданными типом и значениями его параметров.

Ограничение: *имя-функции* и все *имена-формальных-аргументов* должны быть специфицированы явно или неявно как скалярные объекты данных.

Ограничение: данное *имя-формального-аргумента* может появиться в *списке-имен-формальных-аргументов* только один раз.

Ограничение: каждая ссылка на скалярную переменную в *скалярном-выражении* должна быть ссылкой либо на формальный аргумент операторной функции, либо на локальную переменную из той же контекстной области, где находится описание операторной функции.

Описанию операторной функции с тем же именем, что у доступного объекта из объемлющей области, должно предшествовать объявление ее типа в операторе объявления типа.

Областью действия формальных аргументов является описание операторной функции. Каждый формальный аргумент имеет тот же тип и параметры типа, что и объект с тем же именем в контекстной области, содержащей операторную функцию.

Операторную функцию нельзя использовать в качестве аргумента процедуры.

Значение операторной функции получается вычислением выражения с использованием значений фактических аргументов для значений соответствующих формальных аргументов и, если необходимо, с преобразованием результата к объявленному типу и параметрам типа функции.

Ссылка на функцию в скалярном выражении не должна вызывать определения формального аргумента операторной функции или утраты им определенности.

ИЗДАТЕЛЬСТВО "ФИНАНСЫ И СТАТИСТИКА"

предлагает учебные пособия

СТАТИЧЕСКИЕ И ДИНАМИЧЕСКИЕ ЭКСПЕРТНЫЕ СИСТЕМЫ/Э.В. Попов, И.Б. Фоминых, Е.Б. Кисель, М.Д. Шапот – 320 с.

Учебное пособие посвящено одной из центральных проблем искусственного интеллекта – проблеме построения экспертных систем (ЭС). Рассмотрено современное состояние исследований в области искусственного интеллекта. Обоснована важность ЭС как подхода к разработке интегрированных прикладных программ, объединяющих технологии инженерии знаний и традиционного программирования. Приведена классификация ЭС и инструментальных средств. Описаны принципы, методы и средства построения ЭС. Особое внимание уделено новому классу экспертных систем – динамическим системам реального времени.

Для студентов и преподавателей вузов, аспирантов и научных работников.

ПАКЕТЫ ПРОГРАММ ОФИСНОГО НАЗНАЧЕНИЯ/Под ред. проф. С.В. Назарова – 320 с.

Рассматриваются вопросы автоматизации офисной деятельности и пакеты прикладных программ офисного назначения: текстовый процессор, электронная таблица, система управления базами данных. Изложение ведется на примере наиболее распространенного комплекта Microsoft office. Даются сведения по работе в системе Windows и примеры организации офисных систем.

Для студентов экономических специальностей и широкого круга специалистов, использующих персональную вычислительную технику.

Книги Вы можете приобрести в издательстве по адресу:
101000, Москва, ул. Покровка, 7
(метро "Китай-город", выход на ул. Маросейка)

Тел.: **(095) 925-35-02, 923-80-42**
Факс **(095) 925-09-57**

Глава 13. ВСТРОЕННЫЕ ПРОЦЕДУРЫ

Имеется четыре класса встроенных процедур: справочные функции (функции запроса), поэлементные функции, функции преобразования и подпрограммы. Одна из встроенных подпрограмм является поэлементной.

13.1. Встроенные функции

Встроенная функция может быть справочной функцией, поэлементной функцией или функцией преобразования. **Справочной функцией (функцией запроса)** является функция, результат которой зависит от характеристик ее основного аргумента, а не от его значения; фактически значение аргумента может быть неопределенным. **Поэлементная функция** - это функция, специфицированная для скалярных аргументов, но может быть применима и к аргументам-массивам, как это описано в разделе 13.2. Все остальные функции - это **функции преобразования**; почти все они имеют один или несколько аргументов со значением-массивом или результат со значением-массивом.

Обобщенные (общие) имена встроенных функций перечислены в 13.10. В большинстве случаев такие функции воспринимают аргументы более чем одного типа, а тип результата совпадает с типом аргументов. **Специфические имена** встроенных функций вместе с соответствующими обобщенными именами перечислены в 13.12.

Если встроенная функция передается в качестве фактического аргумента внешней процедуры, то должно быть использовано ее специфическое имя, и на него можно ссылаться при вызове процедуры только со скалярными аргументами. Если у встроенной функции нет специфического имени, ее нельзя использовать в качестве фактического аргумента (12.4.1.2).

13.2. Поэлементные встроенные процедуры

13.2.1. Аргументы и результаты поэлементной встроенной функции

Если общее или специфическое имя используется для ссылки на поэлементную встроенную функцию, конфигурация результата совпадает с конфигурацией аргумента с наибольшей размерностью. Если все аргументы являются скалярами, результат также является скаляром. Для тех поэлементных встроенных функций, которые имеют более одного аргумента, все аргументы должны быть согласованы. В случае значения-массива значения элементов результата (если они есть) являются такими же, какие были бы получены, если бы функция со скалярным значением была применена в произвольном порядке отдельно к соответствующему элементу каждого аргумента. Аргумент, соответствующий формальному аргументу с ключевым словом KIND, должен быть специфицирован как скалярное целое выражение инициализации и должен задавать способ представления результата функции, существующий в процессоре.

13.2.2. Аргументы встроенной поэлементной подпрограммы

Поэлементная подпрограмма - это подпрограмма, описанная для скалярных аргументов, но может быть применима и к аргументам-массивам. В ссылке на поэлементную встроенную подпрограмму либо все фактические аргументы должны быть скалярами, либо все фактические аргументы с атрибутами INTENT (OUT) и INTENT (INOUT) должны быть массивами одинаковой конфигурации, а остальные фактические аргументы должны быть согласованы с ними. В случае, когда аргументы с атрибутами INTENT (OUT) и INTENT (INOUT) - массивы, значения элементов результата (если они есть) таковы, какие были бы получены, если бы процедура со скалярными аргументами была применена в произвольном порядке к соответствующим элементам каждого аргумента отдельно.

13.3. Позиционные аргументы и аргументы с ключевыми словами

Все встроенные процедуры могут вызываться либо с **позиционными аргументами**, либо с **ключевыми словами аргумента**. В описаниях в 13.13 приведены имена ключевых слов и позиционная последовательность. Ключевое слово является обязательным для аргумента, только когда опущен предшествующий необязательный аргумент.

13.4. Функция запроса о наличии аргументов

Справочная функция PRESENT позволяет сделать запрос о присутствии фактического аргумента, связанного с формальным аргументом, который имеет атрибут OPTIONAL.

13.5. Числовые, математические, символьные, логические , битовые процедуры и процедуры для параметра вида

13.5.1. Числовые функции

Поэлементные функции INT, REAL, DBLE и CMPLX производят преобразования типа. Поэлементные функции AIMAG, CONJG, AINT, ANINT, NINT, ABS, MOD, SIGN, DIM, DPROD, MODULO, FLOOR, CEILING, MAX и MIN выполняют простые числовые операции.

13.5.2. Математические функции

Поэлементные функции SQRT, EXP, LOG, LOG10, SIN, COS, TAN, ASIN, ACOS, ATAN, ATAN2, SINH, COSH и TANH вычисляют значения элементарных математических функций.

13.5.3. Символьные функции

Поэлементные функции ICHAR, CHAR, LGE, LGT, LLE, LLT, IACHAR, ACHAR, INDEX, VERIFY, ADJUSTL, ADJUSTR, SCAN и LEN_TRIM выполняют символьные операции. Функция преобразования REPEAT выдает повторяющуюся конкатенацию символьного аргумента. Функция преобразования TRIM выдает аргумент с удаленными конечными пробелами.

13.5.4. Символьная справочная функция

Справочная функция LEN выдает длину символьного объекта. Значение аргумента не обязательно должно быть определено. Процессор может не вычислять аргумент этой функции, если ее значение можно определить иначе.

13.5.5. Функции для параметра вида

Справочная функция KIND выдает значение параметра вида для целых, вещественных, комплексных, логических или символьных объектов. Значение аргумента этой функции может быть не определено. Функция преобразования SELECTED_REAL_KIND выдает такое значение параметра вида для вещественных, которое обеспечивает по меньшей мере такие же десятичную точность и диапазон порядка, как у аргументов функции. Функция преобразования SELECTED_INT_KIND выдает такое значение параметра вида для целых, которое обеспечивает по меньшей мере такой же диапазон десятичного порядка, как у аргумента.

13.5.6. Логическая функция

Поэлементная функция LOGICAL выполняет приведение объектов логического типа с разными значениями параметра вида.

13.5.7. Процедуры обработки и справочные функции для битовых данных

Процедуры для действий над битовыми данными - это набор из десяти функций и одной подпрограммы. Логические операции над битами обеспечиваются функциями IOR, IAND, NOT и IEOR; операции сдвига - функциями ISHFT и ISHFTC; сослаться на битовые подполя можно с помощью функции IBITC и подпрограммы MVBITS; однобитовая обработка обеспечивается функциями BTEST, IBSET и IBCLR.

Для использования этих процедур бит определяется как двоичная цифра w, расположенная в позиции k неотрицательного целого скалярного объекта; модель неотрицательного целого определяется как

$$j = \sum_{k=0}^{s-1} w_k * 2^k$$

где w_k должно иметь значение 0 или 1. Пример модельного целого, совместимого с примерами из 13.7.1, должен иметь s=32, т.к. определяет 32-битовое целое.

Справочная функция BIT_SIZE позволяет определять параметр s модели. Значение аргумента этой функции не обязательно должно быть определено. Процессор может не вычислять аргумент этой функции, если значение функции можно определить и без этого.

Эта модель определяет целый объект, состоящий из s битов, последовательно пронумерованных справа налево числами от 0 до s-1. Данная модель является правильной только в контексте использования таких объектов, как аргумент или результат одной из процедур манипуляций с битами. Во всех остальных контекстах применяется модель, определенная для целых в 13.7.1. В частности, несмотря на то, что модели идентичны при $w_{s-1} = 0$, они не соответствуют при $w_{s-1} = 1$ и интерпретация битов в таких объектах зависит от процессора.

13.6. Функция передачи типа

Функция TRANSFER специфицирует, что физическое представление первого аргумента должно трактоваться как обладающее типом и параметрами типа второго аргумента без преобразования.

13.7. Функции обработки и справочные функции для числовых данных

Функции обработки и справочные функции для числовых данных описываются в терминах модели представления чисел в процессоре. Модель имеет параметры, которые определяются так, чтобы сделать ее наиболее пригодной для машины, на которой исполняется выполняемая программа.

13.7.1. Модели для целых и вещественных данных

Модельное множество для целого i определяется равенством

$$i = s * \sum_{k=1}^{Q} w_k * R^{k-1}$$

где R - целое, превышающее единицу, Q - положительное целое, каждое w_k является неотрицательным целым, меньшим, чем R, а s есть +1 или -1. Модельное множество для вещественного x определяется как

$x = 0$

или

$$x = s * b^{\varepsilon} \sum_{k=1}^{p} f_k * b^{-k}$$

где p и b - целые числа, превышающие единицу; каждое f_k является неотрицательным целым, меньшим b, а f_1 к тому же не равно нулю; s есть +1 или -1; а ε - целое, которое расположено между некоторым целым максимумом ε_{MAX} и некоторым целым минимумом ε_{MIN} включительно. Для $x=0$ его экспонента ε и цифры f_k определяются равными нулю. Целые параметры r и q определяют множество модельных целых чисел, а целые параметры b, p, ε_{MIN} и ε_{MAX} определяют множество модельных чисел с плавающей запятой. Параметры целой и вещественных моделей пригодны для каждого целого и вещественного типа данных, реализованного процессором. Параметры характеризуют множество допустимых чисел в определении модели. Числовые функции для манипуляций и запросов обеспечивают значения, связанные с параметрами, и другие константы, относящиеся к ним. Примеры этих функций в данной главе используют модели:

$$i = s * \sum_{k=1}^{31} w_k * 2^{k-1}$$

и

$$x = 0 \text{ или } s * 2^{\varepsilon} \left(1/2 + \sum_{k=1}^{24} f_k * 2^{-k} \right), \qquad -126 \le \varepsilon \le 127$$

13.7.2. Числовые справочные функции

Справочные функции RADIX, DIGITS, MINEXPONENT, MAXEXPONENT, HUGE, EPSILON, TINY, PRECISION и RANGE выдают скалярные значения, относящиеся к параметрам модели, связанным с типом аргументов и его параметрами. Значения аргументов таких функций могут быть не определены, аргументы-указатели могут быть не связаны, а аргументы-массивы могут быть не размещены.

13.7.3. Функции обработки данных с плавающей запятой

Поэлементные функции EXPONENT, SCALE, NEAREST, FRACTION, SET_EXPONENT, SPACING и RRSPACING выдают значения, относящиеся к компонентам модельных значений (13.7.1), связанным с фактическими значениями аргументов.

13.8. Встроенные функции для массивов

Встроенные функции для массивов производят следующие операции над массивами: векторное и матричное умножение, числовые или логические вычисления, понижающие размерность, запрос о структуре массива, конструирование массива, манипуляции с массивом и геометрическое расположение.

13.8.1. Конфигурация массивов-аргументов

Встроенные функции преобразования для массивов оперируют с каждым массивом-аргументом как с единым целым. Конфигурация соответствующего фактического аргумента должна быть поэтому определена, то есть фактический аргумент должен быть секцией массива, массивом неопределенной конфигурации, массивом явной конфигурации, указателем, связанным с адресатом, динамически размещаемым массивом, который уже размещен, или выражением со значением-массивом. Он не может быть массивом неопределенного размера.

Некоторые справочные встроенные функции воспринимают аргументы-массивы, конфигурация которых может быть не определена. Для этих функций в качестве аргументов можно использовать массивы неопределенного размера. Это функция LBOUND и в некоторых случаях ссылки на SIZE и UBOUND.

13.8.2. Аргументы-маски

Некоторые встроенные функции для массивов имеют необязательный аргумент MASK, используемый функцией для выбора тех элементов одного или нескольких аргументов, с которыми потом должна оперировать функция. Элементы, не выбираемые маской, могут быть не определены во время вызова функции.

Аргумент MASK влияет только на значение функции, но не влияет на вычисление (перед вызовом функции) аргументов, представленных выражениями-массивами.

Аргумент MASK должен быть логического типа.

13.8.3. Функции векторного и матричного умножения

Функция матричного умножения MATMUL оперирует с двумя матрицами либо с одной матрицей и одним вектором и выдает произведение матрица-матрица, вектор-матрица или матрица-вектор. Аргументы функции MATMUL - это массивы числового (целого, вещественного, комплексного) или логического типа. Над логическими матрицами и векторами функция MATMUL производит булевское матричное умножение.

Функция скалярного произведения DOT_PRODUCT оперирует с двумя векторами и выдает их скалярное произведение. Вектора должны быть одного типа (числового или логического), как для MATMUL. Для логических векторов DOT_PRODUCT выдает булевское скалярное произведение.

13.8.4. Функции редукции

Функции редукции SUM, PRODUCT, MAXVAL, MINVAL, COUNT, ANY, ALL производят числовые, логические и счетные операции над массивом. Они могут быть применены либо ко всему массиву для получения скалярного результата, либо к заданному измерению для получения результата с уменьшенной на единицу размерностью. С помощью подходящей логической маски, которая согласована с заданным массивом, вычисление может быть ограничено любым подмножеством массива (например, положительными элементами).

13.8.5. Справочные функции для массивов

Справочная функция ALLOCATED выдает значение "истина", если массив-аргумент в данный момент размещен, и значение "ложь" - если не размещен.

Справочные функции для массивов SIZE, SHAPE, LBOUND и UBOUND выдают, соответственно, количество элементов, конфигурацию, нижние и верхние границы индексов для каждого измерения. Размер, конфигурация и границы должны быть определены.

Значения аргументов-массивов для этих функций могут быть не определены.

13.8.6. Функции конструирования массивов

Функции MERGE, SPREAD, PACK и UNPACK создают новые массивы из элементов существующих массивов. MERGE объединяет два согласованных массива в один поэлементным выбором в соответствии с логической маской. SPREAD конструирует массив из нескольких копий фактического аргумента (SPREAD делает это добавлением нового измерения, как при формировании книги из копий одной страницы). PACK и UNPACK, соответственно, "собирает" и "разбирает" элементы одномерного массива из позиций (в позиции) другого массива, где позиции заданы логической маской.

13.8.7. Функция изменения конфигурации массива

Функция RESHAPE вырабатывает из тех же элементов массив другой конфигурации.

13.8.8. Функции обработки массивов

Функции TRANSPOSE, EOSHIFT и CSHIFT производят различные манипуляции над массивами. TRANSPOSE производит операцию транспонирования матрицы, заданной двумерным массивом. Функции сдвига оставляют конфигурацию массива неизменной, но сдвигают позиции элементов параллельно заданному измерению массива. Эти сдвиги либо циклические (CSHIFT), в этом случае элементы выдвигаются из одного конца и появляются на другом, либо такие сдвиги (EOSHIFT), когда элементы в заданных границах сдвигаются на освободившиеся позиции (сдвиг типа end-off).

13.8.9. Функции определения расположения в массиве

Функции MAXLOC и MINLOC выдают положение (индексы) элемента массива, который имеет максимальное (или минимальное) значение соответственно. С помощью необязательной логической маски, согласованной с данным массивом, редукция может быть ограничена любым подмножеством массива.

13.8.10. Справочная функция для запроса о статусе связанности указателя

Функция ASSOCIATED проверяет, связан ли ее аргумент в данный момент с каким-либо адресатом, с конкретным адресатом или с тем же адресатом, что другой указатель.

13.9. Встроенные подпрограммы

Встроенные подпрограммы обеспечиваются процессором и имеют специальные определения, приведенные в 13.11 и 13.13. На встроенную подпрограмму ссылаются посредством оператора CALL, который использует ее имя явно. Имя встроенной подпрограммы не должно использоваться в качестве фактического аргумента. Действие ссылки на подпрограмму описано в 13.13.

13.9.1. Подпрограммы даты и времени

Подпрограммы DATE_AND_TIME и SYSTEM_CLOCK выдают целое значение даты и часов реального времени. Выдаваемое время является местным, но есть средства для нахождения разности между местным временем и координированным универсальным временем (по Гринвичу).

13.9.2. Псевдослучайные числа

Подпрограмма RANDOM_NUMBER выдает псевдослучайное число или массив псевдослучайных чисел. Подпрограмма RANDOM_SEED инициализирует или перезапускает генератор псевдослучайных чисел.

13.9.3. Подпрограмма копирования битов

Поэлементная подпрограмма MVBITS копирует битовое поле из заданных позиций одного целого объекта в заданные позиции другого целого объекта.

13.10. Обобщенные встроенные функции

Для всех встроенных процедур аргументы обозначаются именами, которые следует использовать как ключевые слова, когда используется форма фактических аргументов с ключевыми словами. Например, ссылка на CMPLX может быть записана в виде: CMPLX (A, B, M) или в виде CMPLX (Y=B, KIND=M, X=A).

Многие из этих ключевых слов имеют имена, указывающие на способ их использования. Например:

KIND	описывает параметр вида для результата
STRING, STRING_A	произвольная символьная строка
BACK	указывает, что сканирование строки должно производиться справа налево (назад)
MASK	маска, которая может быть применена к аргументам
DIM	выбранное измерение аргумента-массива

13.10.1. Справочная функция для запроса о наличии аргументов

PRESENT(A)	присутствие аргумента

13.10.2. Числовые функции

ABS(A)	абсолютное значение
AIMAG(Z)	мнимая часть комплексного числа
AINT(A, KIND) необязателен KIND	усечение до целого числа
ANINT(A, KIND) необязателен KIND	ближайшее целое число
CEILING(A)	наименьшее целое, большее или равное числу
CMPLX(X, Y, KIND) необязательны Y, KIND	преобразование в комплексный тип
CONJG(Z)	сопряженное комплексное число
DBLE(A)	преобразование к вещественному типу двойной точности
DIM(X, Y)	положительная разность
DPROD(X, Y)	произведение вещественное двойной точности
FLOOR(A)	наибольшее целое, меньшее или равное числу
INT(A, KIND) необязателен KIND	преобразование в целый тип
MAX(A1, A2, A3, ...) необязательны A3, ...	максимальное значение
MIN(A1, A2, A3, ...) необязательны A3, ...	минимальное значение
MOD(A, P)	функция остатка
MODULO(A, P)	функция "по модулю"
NINT(A, KIND) необязателен KIND	ближайшее целое
REAL(A, KIND) необязателен KIND	преобразование в вещественный тип
SIGN(A, B)	передача знака

13.10.3. Математические функции

ACOS(X)	арккосинус
ASIN(X)	арксинус
ATAN(X)	арктангенс
ATAN2(Y, X)	арктангенс
COS(X)	косинус
COSH(X)	гиперболический косинус
EXP(X)	экспонента
LOG(X)	натуральный логарифм
LOG10(X)	общий (десятичный) логарифм
SIN(X)	синус
SINH(X)	гиперболический синус
SQRT(X)	корень квадратный
TAN(X)	тангенс
TANH(X)	гиперболический тангенс

13.10.4. Символьные функции

ACHAR(I)	символ в заданной позиции кода ASCII
ADJUSTL (STRING)	выравнивание влево
ADJUSTR (STRING)	выравнивание вправо
CHAR (I, KIND) необязателен KIND	символ в заданной позиции в кодировке процессора
IACHAR (C)	позиция символа в кодировке ASCII
ICHAR (C)	позиция символа в кодировке процессора
INDEX (STRING, SUBSTRING, BACK) необязателен BACK	начальная позиция подстроки
LEN_TRIM (STRING)	длина без концевых пробелов
LGE (STRING_A, STRING_B)	лексически больше или равно
LGT (STRING_A, STRING_B)	лексически больше
LLE (STRING_A, STRING_B)	лексически меньше или равно
LLT (STRING_A, STRING_B)	лексически меньше
REPEAT (STRING, NCOPIES)	повторяемая конкатенация
SCAN (STRING, SET, BACK) необязателен BACK	просмотр строки для поиска символа в наборе символов
TRIM (STRING)	удаление конечных пробелов
VERIFY(STRING, SET, BACK) необязателен BACK	проверка множества символов в строке

13.10.5. Символьная справочная функция

LEN (STRING)	длина символьного объекта

13.10.6. Справочные функции вида

KIND (X)	значение параметра вида
SELECTED_INT_KIND (R)	значение параметра вида для целых в заданном диапазоне
SELECTED_REAL_KIND (P, R) необязательны P или R	значение параметра вида для вещественных с заданными точностью и диапазоном

13.10.7. Логическая функция

LOGICAL (L, KIND) необязателен KIND	преобразование объектов логического типа с разными параметрами вида

13.10.8. Числовые справочные функции

DIGITS (X)	число значащих цифр в модели
EPSILON (X)	число, почти не отличающееся от заданного
HUGE(X)	наибольшее число в модели
MAXEXPONENT(X)	максимальная экспонента в модели
MINEXPONENT(X)	минимальная экспонента в модели
PRECISION(X)	десятичная точность
RADIX(X)	основание модели
RANGE(X)	значение десятичной экспоненты
TINY(X)	наименьшее положительное число в модели

13.10.9. Битовая справочная функция

BIT_SIZE(I)	число битов в модели

13.10.10. Функции для операций с битами

BTEST (I, POS)	проверка битов
IAND (I, J)	логическое AND
IBCLR (I, POS)	обнуление бита
IBITS (I, POS, LEN)	выделение бита
IBSET (I, POS)	установка бита
IEOR (I, J)	исключительное OR
IOR (I, J)	включительное OR
ISHFT (I, SHIFT)	логический сдвиг
ISHFTC (I, SHIFT, SIZE) необязателен SIZE	циклический сдвиг
NOT (I)	логическое дополнение

13.10.11. Функция передачи типа

TRANSFER (SOURCE, MOLD, SIZE) необязателен SIZE	трактует первый аргумент, как если бы он имел тип второго аргумента

13.10.12. Функции для операций с плавающей точкой

EXPONENT (X)	экспонентная часть модельного числа
FRACTION (X)	дробная часть числа
NEAREST (X, S)	ближайшее, отличное от заданного, процессорное число в данном направлении
RRSPASING (X)	обратная величина от относительной окрестности модельных чисел около данного числа
SCALE (X, I)	умножить вещественное число на его основание, возведенное в целую степень
SET_EXPONENT (X, I)	установить экспонентную часть числа
SPACING (X)	абсолютная окрестность модельных чисел около данного числа

13.10.13. Функции векторного и матричного умножения

DOT_PRODUCT(VECTOR_A, VECTOR_B)	скалярное произведение двух одномерных массивов
MATMUL (MATRIX_A, MATRIX_B)	умножение матриц

13.10.14. Функции редукции

ALL (MASK, DIM) необязателен DIM	"истина", если все значения - "истина"
ANY (MASK, DIM) необязателен DIM	"истина", если какое-либо из значений - "истина"
COUNT (MASK, DIM) необязателен DIM	количество элементов массива со значением - "истина"
MAXVAL (ARRAY, DIM, MASK) необязательны DIM, MASK	максимальное значение в массиве
MINVAL (ARRAY, DIM, MASK) необязательны DIM, MASK	минимальное значение в массиве
PRODUCT (ARRAY, DIM, MASK) необязательны DIM, MASK	произведение элементов массива
SUM (ARRAY, DIM, MASK) необязательны DIM, MASK.	сумма элементов массива

13.10.15. Функции запроса о массиве

ALLOCATED (ARRAY)	статус размещенности массива
LBOUND (ARRAY, DIM) необязателен DIM	нижние границы измерений массивов
SHAPE (SOURCE)	конфигурация массива или скаляра
SIZE (ARRAY, DIM) необязателен DIM	общее число элементов в массиве
UBOUND (ARRAY, DIM) необязателен DIM	верхние границы измерений массива

13.10.16. Функции конструирования массива

MERGE (TSOURCE, FSOURCE, MASK)	слияние по маске
PACK (ARRAY, MASK, VECTOR) необязателен VECTOR	упаковка массива в вектор по маске
SPREAD (SOURCE, DIM, NCOPIES)	копирование массива добавлением измерений
UNPACK (VECTOR, MASK, FIELD)	распаковка вектора в массив по маске

13.10.17. Функция изменения конфигурации массива

RESHAPE (SHAPE, SOURCE, PAD, ORDER) необязательны PAD и ORDER	изменение конфигурации массива

13.10.18. Функции манипуляции с массивами

CSHIFT (ARRAY, SHIFT, DIM) необязателен DIM	циклический сдвиг
EOSHIFT (ARRAY, SHIFT, BOUNDARY, DIM) необязательны BOUNDARY, DIM	нециклический сдвиг (типа end-off)
TRANSPOSE (MATRIX)	транспонирование матрицы

13.10.19. Функции определения положения в массиве

MAXLOC (ARRAY, MASK) — положение максимального значения в массиве
необязателен MASK

MINLOC (ARRAY, MASK) — положение минимального значения в массиве
необязателен MASK

13.10.20. Функция запроса о статусе связанности указателя

ASSOCIATED (POINTER, TARGET) — статус связанности или сравнение
необязателен TARGET

13.11. Встроенные подпрограммы

DATE_AND_TIME (DATE, TIME, ZONE, VALUES) — получение даты и времени
необязательны DATE, TIME, ZONE, VALUES

MVBITS (FROM, FROMPOS, LEN, TO, TOPOS) — копирует биты из одного целого в другое

RANDOM_NUMBER (HARVEST) — выдает псевдослучайное число

RANDOM_SEED (SIZE, GET, PUT) — инициализирует или перезапускает генератор псевдослучайных чисел
необязательны SIZE, GET, PUT

SYSTEM_CLOCK (COUNT, COUNT_RATE, COUNT_MAX) — получение даты из системных часов
необязательны COUNT, COUNT_RATE, COUNT_MAX

13.12. Специфические имена для встроенных функций

Специфическое имя	*Обобщенное имя*	*Тип аргументов*
ABS(A)	ABS(A)	стандартный вещественный
ACOS(X)	ACOS(X)	стандартный вещественный
AIMAG(Z)	AIMAG(Z)	стандартный комплексный
AINT(A)	AINT(A)	стандартный вещественный
ALOG(X)	LOG(A)	стандартный вещественный
ALOG10(X)	LOG10(X)	стандартный вещественный
• AMAX0(A1, A2, A3, ...) необязательны A3, ...	REAL(MAX(A1, A2, A3, ...))	стандартный целый
• AMAX1(A1, A2, A3, ...) необязательны A3, ...	MAX(A1, A2, A3, ...)	стандартный вещественный
• AMIN0(A1, A2, A3, ...) необязательны A3, ...	REAL(MIN(A1, A2, A3, ...))	стандартный целый
• AMIN1(A1, A2, A3, ...) необязательны A3, ...	MIN(A1, A2, A3, ...)	стандартный вещественный
AMOD(A, P)	MOD(A, P)	стандартный вещественный
ANINT(A)	ANINT(A)	стандартный вещественный
ASIN(X)	ASIN(X)	стандартный вещественный
ATAN(X)	ATAN(X)	стандартный вещественный
ATAN2(Y, X)	ATAN(Y, X)	стандартный вещественный
CABS(A)	ABS(A)	стандартный комплексный
CCOS(X)	COS(X)	стандартный комплексный
CEXP(X)	EXP(X)	стандартный комплексный
• CHAR(I)	CHAR(I)	стандартный целый
CLOG(X)	LOG(X)	стандартный комплексный
CONJG(Z)	CONJG(Z)	стандартный комплексный

COS(X)	COS(X)	стандартный вещественный
COSH(X)	COSH(X)	стандартный вещественный
CSIN(X)	SIN(X)	стандартный комплексный
CSQRT(X)	SQRT(X)	стандартный комплексный
DABS(A)	ABS(A)	двойной точности
DACOS(X)	ACOS(X)	двойной точности
DASIN(X)	ASIN(X)	двойной точности
DATAN(X)	ATAN(X)	двойной точности
DATAN2(Y, X)	ATAN2(Y, X)	двойной точности
DCOS(X)	COS(X)	двойной точности
DCOSH(X)	COSH(X)	двойной точности
DDIM(X, Y)	DIM(X, Y)	двойной точности
DEXP(X)	EXP(X)	двойной точности
DIM(X, Y)	DIM(X, Y)	стандартный вещественный
DINT(A)	AINT(A)	двойной точности
DLOG(X)	LOG(X)	двойной точности
DLOG10(X)	LOG10(X)	двойной точности
• DMAX1(A1, A2, A3, ...) необязательны A3, ...	MAX(A1, A2, A3, ...)	двойной точности
• DMIN1(A1, A2, A3, ...) необязательны A3, ...	MIN(A1, A2, A3, ...)	двойной точности
DMOD(A, P)	MOD(A, P)	двойной точности
DNINT(A)	ANINT(A)	двойной точности
DPROD(X, Y)	DPROD(X, Y)	стандартный вещественный
DSIGN(A, B)	SIGN(A, B)	двойной точности
DSIN(X)	SIN(X)	двойной точности
DSINH(X)	SINH(X)	двойной точности
DSQRT(X)	SQRT(X)	двойной точности
DTAN(X)	TAN(X)	двойной точности
DTANH(X)	TANH(X)	двойной точности
EXP(X)	EXP(X)	стандартный вещественный
• FLOAT(A)	REAL(A)	стандартный целый
IABS(A)	ABS(A)	стандартный целый
• ICHAR(C)	ICHAR(C)	стандартный символьный
IDIM(X, Y)	DIM(X, Y)	стандартный целый
• IDINT(A)	INT(A)	двойной точности
IDNINT(A)	NINT(A)	двойной точности
• IFIX(A)	INT(A)	стандартный вещественный
INDEX(STRING, SUBSTRING)	INDEX(STRING, SUBSTRING)	стандартный символьный
• INT(A)	INT(A)	стандартный вещественный
ISIGN(A, B)	SIGN(A, B)	стандартный целый
LEN(STRING)	LEN(STRING)	стандартный символьный
• LGE(STRING_A, STRING_B)	LGE(STRING_A, STRING_B)	стандартный символьный
• LGT(STRING_A, STRING_B)	LGT(STRING_A, STRING_B)	стандартный символьный
• LLE(STRING_A, STRING_B)	LLE(STRING_A, STRING_B,	стандартный символьный
• LLT(STRING_A, STRING_B)	LLT(STRING_A, STRING_B)	стандартный символьный

• MAX0(A1, A2, A3, ...) необязательны A3, ...	MAX(A1, A2, A3, ...)	стандартный целый
• MAX1(A1, A2, A3, ...) необязательны A3, ...	INT(MAX(A1, A2, A3, ...)	стандартный вещественный
• MIN0(A1, A2, A3, ...) необязательны A3, ...	MIN(A1, A2, A3, ...)	стандартный целый
• MIN1(A1, A2, A3, ...) необязательны A3, ...	INT(MIN(A1, A2, A3, ...)	стандартный вещественный
MOD(A, P)	MOD(A, P)	стандартный целый
NINT(A)	NINT(A)	стандартный вещественный
• REAL(A)	REAL(A)	стандартный целый
SIGN(A, B)	SIGN(A, B)	стандартный вещественный
SIN(X)	SIN(X)	стандартный вещественный
SINH(X)	SINH(X)	стандартный вещественный
• SNGL(A)	REAL(A)	двойной точности
SQRT(X)	SQRT(X)	стандартный вещественный
TAN(X)	TAN(X)	стандартный вещественный
TANH(X)	TANH(X)	стандартный вещественный

• Эти специфические имена встроенных функций нельзя использовать в качестве фактических аргументов.

13.13. Спецификации встроенных процедур

Этот раздел содержит детальные спецификации всех встроенных процедур в алфавитном порядке.

13.13.1. ABS (A)

Описание: абсолютное значение.

Класс: поэлементная функция.

Аргумент: должен быть целого, вещественного или комплексного типа.

Тип результата и параметр типа: те же, что у A, с той разницей, что если A - комплексного типа, то результат - вещественного.

Значение результата: если тип A - целый или вещественный, значением результата является |A|; если A - комплексное со значением (X,Y), результат равен процессорно-зависимой аппроксимации значения $(X^2+Y^2)^{1/2}$.

Пример: ABS((3.0,4.0)) имеет значение 5.0.

13.13.2. ACHAR (I)

Описание: выдает символ в заданной позиции кодировки ASCII. Функция, обратная к функции IACHAR.

Класс: поэлементная функция.

Аргумент: I должно быть целого типа.

Тип результата и параметр типа: символьный, длина равна единице, значение параметра вида KIND('A').

Значение результата: если I имеет значение в интервале 0 <= I <= 127, результатом будет символ из I-й позиции в кодировке ASCII при условии, что этот символ представим в процессоре, иначе результат зависит от процессора. Если процессор не располагает возможностью представления и строчных, и прописных букв, а I соответствует букве в регистре, не представляемом процессором, результатом является буква регистра,

представимого процессором. ACHAR (ICHAR (C)) должно иметь значение C для любого символа C, представимого процессором.

Пример: ACHAR(88) имеет значение 'X'.

13.13.3. ACOS (X)

Описание: арккосинус (функция, обратная косинусу).

Класс: поэлементная функция.

Аргумент: X должен быть вещественного типа. Его значение должно удовлетворять неравенству |X|«1.

Тип результата и параметр типа: те же, что у X.

Значение результата: результат имеет значение, равное процессорно-зависимой аппроксимации ARCCOS(X), выраженное в радианах. Оно находится в интервале 0 « ACOS(X) « *п*.

Пример: ACOS(0.54030231) имеет значение 1.0 (приближенно).

13.13.4. ADJUSTL (STRING)

Описание: сдвиг к левому краю за счет удаления начальных пробелов и добавления конечных пробелов.

Класс: поэлементная функция.

Аргумент: STRING должно быть символьного типа.

Тип результата: символьный с той же длиной и с тем же параметром вида, что и STRING.

Значение результата: значение результата совпадает со STRING с той разницей, что исключены все начальные пробелы и добавлено точно такое же число конечных пробелов.

Пример: ADJUSTL(' WORD') имеет значение 'WORD '.

13.13.5. ADJUSTR (STRING)

Описание: сдвиг к правому краю за счет удаления конечных пробелов и добавления начальных.

Класс: поэлементная функция.

Аргумент: STRING должно быть символьного типа.

Тип результата: символьный с той же длиной и с тем же параметром вида, что и STRING.

Значение результата: значение результата совпадает со STRING с той разницей, что исключены все конечные пробелы и включено точно такое же число начальных пробелов.

Пример: ADJUSTR('WORD ') имеет значение ' WORD'.

13.13.6. AIMAG(Z)

Описание: мнимая часть комплексного числа.

Класс: поэлементная функция.

Аргумент: Z должно быть комплексного типа.

Тип результата и параметр типа: вещественный с тем же параметром вида, что Z.

Значение результата: если Z имеет значение (X,Y), то результат имеет значение Y.

Пример: AIMAG ((2.0,3.0)) имеет значение 3.0.

13.13.7. AINT (A,KIND)

Необязательный аргумент: KIND.

Описание: усечение до целого числа.

Класс: поэлементная функция.

Аргументы:

A	должен быть вещественным.
KIND (необязательный)	должен быть скалярным целым выражением инициализации.

Тип результата и параметр типа: результат - вещественного типа. Если задан KIND, параметр вида определяется им, иначе параметр вида тот же, что у A.

Значение результата: если $|A|<1$, AINT(A) имеет значение 0, если $|A|\geq 1$, то AINT(A) имеет значение, равное целому, модуль которого есть наибольшее целое, не превышающее модуля A, а знак совпадает со знаком A.

Примеры: AINT (2.783) имеет значение 2.0, AINT (-2.783) имеет значение -2.0.

13.13.8. ALL (MASK,DIM)

Необязательный аргумент: DIM.

Описание: определяет, все ли элементы MASK по измерению DIM имеют значение "истина".

Класс: функция преобразования.

Аргументы:

MASK	должен быть логического типа и не должен быть скаляром.
DIM (необязательный)	должен быть скаляром целого типа со значением в интервале $1 \leq DIM \leq n$, где n - размерность MASK. Соответствующий фактический аргумент не должен быть необязательным формальным аргументом.

Тип результата, параметр типа и конфигурация: результат имеет логический тип с тем же параметром вида, что у аргумента MASK. Результат является скаляром, если DIM отсутствует или MASK имеет размерность 1; иначе он является массивом размерности $n-1$ и конфигурации $(d_1, d_2,...,d_{DIM-1}, d_{DIM+1},...,d_n)$, где $(d_1, d_2,...,d_n)$ - конфигурация MASK.

Значение результата.

Случай 1: результат ALL(MASK) имеет значение "истина", если все элементы MASK имеют значение "истина" или если MASK имеет нулевой размер, и значение результата - "ложь", если какой-либо элемент имеет значение "ложь".

Случай 2: если MASK имеет размерность 1, ALL (MASK,DIM) имеет значение, равное значению ALL(MASK). Иначе значение элемента $(s_1,s_2,...,s_{DIM-1},s_{DIM+1},...,s_n)$ функции ALL (MASK,DIM) равно ALL (MASK $(s_1,s_2,..., s_{DIM-1}, :, s_{DIM+1}, ..., s_n)$).

Примеры:

Случай 1: значением функции ALL ((/.TRUE.,.FALSE.,.TRUE./)) является "ложь".

Случай 2: если B является массивом $\begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix}$, а C - массив $\begin{bmatrix} 0 & 3 & 5 \\ 7 & 4 & 8 \end{bmatrix}$,

тогда ALL (B .NE. C, DIM = 1) равно [истина, ложь, ложь],

а ALL (B.NE.C, DIM = 2) равно [ложь, ложь].

13.13.9. ALLOCATED (ARRAY)

Описание: определяет, размещен ли в памяти или в данный момент не размещен динамически размещаемый массив.

Класс: справочная функция.

Аргумент: ARRAY должен быть динамически размещаемым массивом.

Тип, параметр типа и конфигурация результата: результат является скаляром стандартного логического типа.

Значение результата: результат имеет значение "истина", если ARRAY в настоящий момент размещен, и "ложь", если ARRAY в данный момент не размещен. Если же статус размещенности (14.8) массива не определен, результат тоже не определен.

13.13.10. ANINT (A,KIND)

Необязательный аргумент: KIND.

Описание: ближайшее целое число.

Класс: поэлементная функция.

Аргументы:

A	должен иметь тип вещественный.
KIND (необязательный)	должен быть скалярным целым выражением инициализации.

Тип и параметр типа результата: результат вещественного типа. Параметр вида либо тот, который специфицирован посредством KIND, либо (если KIND отсутствует) тот же, что у A.

Значение результата: если A > 0, ANINT (A) имеет значение AINT(A+0.5); если A <= 0, ANINT(A) имеет значение AINT(A-0.5).

Примеры: ANINT (2.783) имеет значение 3.0, ANINT (-2.783) имеет значение -3.0.

13.13.11. ANY (MASK,DIM)

Необязательный аргумент: DIM.

Описание: определяет, имеет ли какой-либо из элементов MASK по измерению DIM значение "истина".

Класс: функция преобразования.

Аргументы:

MASK	должен быть логического типа и не должен быть скаляром.
DIM (необязательный)	должен быть скаляром целого типа и иметь значение в интервале $1 <= DIM <= N$, где N - размерность MASK. Соответствующий фактический аргумент не должен быть необязательным формальным аргументом.

Тип результата, параметр типа и конфигурация: результат имеет логический тип с тем же значением параметра вида, что у MASK. Он является скаляром, если DIM отсутствует или MASK имеет размерность 1; иначе он является массивом размерности $N-1$ и конфигурации $(d_1, d_2, ..., d_{DIM-1}, d_{DIM+1}, ..., d_N)$, где $(d_1, d_2, ..., d_N)$ - конфигурация MASK.

Значение результата:

Случай 1: результат функции ANY (MASK) имеет значение "истина", если хотя бы один из элементов MASK имеет значение "истина", и значение функции - "ложь", если ни один из элементов не имеет значение "истина" или если MASK имеет нулевой размер.

Случай 2: если MASK имеет размерность 1, ANY (ARRAY, DIM) имеет значение, равное значению ANY (MASK). Иначе значение элемента $(s_1, s_2, ..., s_{DIM-1}, s_{DIM+1}, ..., s_N)$ функции ANY (MASK, DIM) равно ANY (MASK $(s_1, s_2, ..., s_{DIM-1}, :, s_{DIM+1}, ..., s_N)$).

Примеры:

Случай 1: значением функции ANY ((/.TRUE., .FALSE., .TRUE./)) является "истина".

Случай 2: если B является массивом $\begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix}$, а C - массив $\begin{bmatrix} 0 & 3 & 5 \\ 7 & 4 & 8 \end{bmatrix}$,

то ANY (B .NE. C, DIM=1) есть [истина, ложь,истина],

а ANY (B.NE.C, DIM=2) есть [истина, истина]

13.13.12. ASIN (X)

Описание: арксинус (функция, обратная синусу).

Класс: поэлементная функция.

Аргумент: X должен быть вещественного типа. Его значение должно удовлетворять неравенству |X| <= 1.

Тип результата и параметр типа: те же, что у X.

Значение результата: результат имеет значение, равное процессорно-зависимой аппроксимации ARCSIN(X), выраженной в радианах. Оно находится в интервале $-\pi/2$ <= ASIN(X) <= $\pi/2$.

Пример: ASIN(0.84147098) имеет значение 1.0 (приближенно).

13.13.13. ASSOCIATED (POINTER,TARGET)

Необязательный аргумент: TARGET.

Описание: выдает статус связанности своего аргумента-указателя или проверяет, связан ли указатель с адресатом.

Класс: справочная функция.

Аргументы:

POINTER	должен быть указателем и может быть любого типа. Его статус связанности не должен быть неопределенным.
TARGET (необязательный)	должен быть указателем или адресатом. Если это указатель, его статус связанности не должен быть неопределенным.

Тип результата. Результат стандартного логического типа.

Значение результата.

Случай 1: при отсутствии TARGET результат - "истина", если POINTER в данный момент связан с адресатом, иначе - "ложь".

Случай 2: если TARGET присутствует и является адресатом, результат - "истина", если POINTER в данный момент связан с этим адресатом, иначе - "ложь".

Случай 3: если TARGET присутствует и является указателем, результат - "истина", если и POINTER, и TARGET в данный момент связаны с одним и тем же адресатом, иначе результат - "ложь". Если либо POINTER, либо TARGET не связаны, результат - "ложь"

Примеры: ASSOCIATED (CURRENT, HEAD) - "истина", если CURRENT указывает на HEAD. После выполнения

```
A_PART => A (:N)
```

ASSOCIATED(A_PART,A) -"истина", если N равно UBOUND (A, DIM=1). После выполнения

```
NULLIFY (CUR); NULLIFY (TOP)
```

ASSOCIATED(CUR, TOP) - "ложь"

13.13.14. ATAN (X)

Описание: арктангенс (функция, обратная тангенсу).

Класс: поэлементная функция.

Аргумент: X должен быть вещественного типа.

Тип результата и параметр типа: *те же, что у X.*

Значение результата: результат имеет значение, равное процессорно-зависимой аппроксимации ARCTAN(X), выраженное в радианах. Оно находится в интервале $-\pi/2 \leq$ ATAN(X) $\leq \pi/2$.

Пример: ATAN(1.5574077) имеет значение 1.0 (приближенно).

13.13.15. ATAN2 (Y,X)

Описание: функция арктангенс (обратная функции тангенс). Результат - это главное значение для аргумента, который является ненулевым комплексным числом (X,Y).

Класс: поэлементная функция.

Аргументы:

Y	должен быть вещественного типа.
X	должен быть того же типа и с тем же параметром вида, что Y. Если Y имеет значение нуль, то X не должен быть нулем.

Тип результата и параметр типа: те же, что у X.

Значение результата: результат имеет значение, равное процессорно-зависимой аппроксимации для главного значения комплексного аргумента (X,Y), выраженное в радианах. Оно находится в интерале $-\pi <$ ATAN2 (Y,X) $\leq \pi$ и равно процессорно-зависимой аппрокиации значения ARCTAN (Y/X), если X $\neq$ 0. Если Y > 0, результат положительный. Если Y = 0, то результат равен нулю, если X > 0, и равен π, если X < 0. Если Y < 0, результат *отрицательный. Если X=0, абсолютное значение результата равно* $\pi/2$.

Примеры: ATAN2 (1.5574077,1.0) имеет значение 1.0 (приближенно).

Если Y имеет значение $\begin{bmatrix} 1 & 1 \\ -1 & -1 \end{bmatrix}$, а X имеет значение $\begin{bmatrix} -1 & 1 \\ -1 & 1 \end{bmatrix}$,

значение ATAN2(Y,X) приблизительно $\begin{bmatrix} 3\pi/4 & \pi/4 \\ -3\pi/4 & -\pi/4 \end{bmatrix}$.

13.13.16. BIT_SIZE (I)

Описание: выдает число битов s, определяемое моделью, описанной в 13.5.7.

Класс: справочная функция.

Аргумент: I должно быть целого типа.

Тип результата, параметр типа и конфигурация: скаляр целого типа с тем же параметром вида, что для I.

Значение результата: значением результата является количество битов s в модельном целом, определенном для операций с битовыми данными в 13.5.7.

Пример: BIT_SIZE(1) имеет значение 32, если s в модели равно 32.

13.13.17. BTEST (I, POS)

Описание: проверяет бит в целом значении.

Класс: поэлементная функция.

Аргументы:

I	должно быть целого типа.
POS	должно быть целого типа, неотрицательным и должно быть меньше, чем BIT_SIZE (I).

Тип результата: результат стандартного логического типа.

Значение результата: результат имеет значение "истина", если POS-й бит в I имеет значение 1, и "ложь", если POS-й бит в I имеет значение 0. Модель для интерпретации целого значения как последовательности битов приведена в 13.5.7.

Примеры: BTEST (8, 3) имеет значение "истина". Если A имеет значение $\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$,

значение BTEST(A,2) есть $\begin{bmatrix} \text{ложь} & \text{ложь} \\ \text{ложь} & \text{истина} \end{bmatrix}$,

а значение BTEST (2,A) есть $\begin{bmatrix} \text{истина} & \text{ложь} \\ \text{ложь} & \text{ложь} \end{bmatrix}$.

13.13.18. CEILING (A)

Описание: выдает наименьшее целое, которое больше или равно аргументу.

Класс: поэлементная функция.

Аргумент: должен быть вещественным.

Тип результата и параметр типа: стандартный целый тип.

Значение результата: результат имеет значение, равное наименьшему целому, которое больше или равно A. Результат не определен, если процессор не может представить такое значение стандартного целого типа.

Примеры: CEILING (3.7) имеет значение 4. CEILING (-3.7) имеет значение -3.

13.13.19. CHAR (I,KIND)

Описание: выдает символ в данной позиции в соответствии с кодировкой процессора. Функция, обратная к функции ICHAR.

Класс: поэлементная функция.

Аргументы:

I	должен быть целого типа со значением в интервале 0 «= I «= n-1, где n - количество символов в кодировке, связанной с заданным параметром вида.
KIND (необязательный)	должен быть скалярным целым выражением инициализации.

Тип результата и параметры типа: символьный единичной длины. Параметр вида либо тот, который задан с помощью KIND, либо (при отсутствии KIND) тот, который определен для стандартного символьного типа.

Значение результата: результатом является символ в позиции I кодировки, связанной с заданным параметром вида. ICHAR (CHAR (I, KIND(C))) должно иметь значение I для 0 «= I «= n-1, а CHAR (ICHAR (C), KIND (C)) должно иметь значение C для любого символа C, представимого в процессоре.

Пример: CHAR(88) имеет значение X в процессоре, использующем кодировку ASCII.

13.13.20. CMPLX (X,Y,KIND)

Необязательные аргументы: Y, KIND.

Описание: преобразование в комплексный тип.

Класс: поэлементная функция.

Аргументы:

X	должен иметь тип целый, вещественный или комплексный.
Y (необязательный)	должен иметь тип целый или вещественный. Он должен отсутствовать, если X - комплексного типа.
KIND (необязательный)	должен быть скалярным целым выражением инициализации.

Тип результата и параметр типа: результат имеет комплексный тип. Если KIND присутствует, параметр вида задается им, иначе параметр вида тот же, что у стандартного вещественного типа.

Значение результата: если Y отсутствует и X - не комплексное, это то же самое, что присутствие Y с нулевым значением. Если X - комплексное, это то же самое, что наличие Y со значением AIMAG(X). CMPLX(X,Y,KIND) имеет комплексное значение, вещественная часть которого - это REAL(X, KIND), а мнимая - REAL(Y, KIND).

Пример: CMPLX(-3) имеет значение (-3.0, 0.0).

13.13.21. CONJG(Z)

Описание: сопряженное к данному комплексное число.

Класс: поэлементная функция.

Аргумент: Z должен быть комплексного типа.

Тип результата и параметр типа: те же, что у Z.

Значение результата: если Z имеет значение (x,y), то результат имеет значение (x, -y).

Пример: CONJG((2.0, 3.0)) имеет значение (2.0, -3.0).

13.13.22. COS (X)

Описание: функция косинус.

Класс: поэлементная функция.

Аргумент: X должен быть вещественного типа или комплексного.

Тип результата и параметр типа: те же, что у X.

Значение результата: результат имеет значение, равное процессорно-зависимой аппроксимации cos(X). Если X имеет тип вещественный, он рассматривается как имеющий значение в радианах. Если X - комплексного типа, его действительная часть рассматривается как значение, выраженное в радианах.

Пример: COS(1.0) имеет значение 0.54030231 (приближенно).

13.13.23. COSH (X)

Описание: функция гиперболический косинус.

Класс: поэлементная функция.

Аргумент: X должен быть вещественного типа.

Тип результата и параметр типа: те же, что у X.

Значение результата: результат имеет значение, равное процессорно-зависимой аппроксимации cosH(X).

Пример: COS(1.0) имеет значение 1.5430806 (приближенно).

13.13.24. COUNT (MASK,DIM)

Необязательный аргумент: DIM.

Описание: подсчитывает количество элементов со значением "истина" в массиве MASK по измерению DIM.

Класс: функция преобразования.

Аргументы:

MASK — должен быть логического типа и не должен быть скаляром.

DIM (необязательный) — должен быть скаляром целого типа со значением в интервале 1 <= DIM <= N, где N - размерность MASK. Соответствующий фактический аргумент не должен быть необязательным формальным аргументом.

Тип результата, параметр типа и конфигурация: результат имеет стандартный целый тип; это скаляр, если либо DIM отсутствует, либо MASK имеет размерность, равную единице; иначе - это массив размерности N-1 и конфигурации $(d_1, d_2, ..., d_{DIM-1}, d_{DIM+1}, ..., d_N)$, где $(d_1, d_2, ..., d_N)$ - конфигурация MASK.

Значение результата:

Случай 1: результат функции COUNT(MASK) имеет значение, равное количеству элементов MASK, имеющих значение "истина", или нулю, если MASK имеет нулевой размер.

Случай 2: если MASK имеет размерность, равную единице, COUNT(MASK,DIM) имеет значение, равное значению COUNT(MASK). Иначе значение элемента $(s_1, s_2, ..., s_{DIM-1}, s_{DIM+1}, ..., s_N)$ массива COUNT(MASK,DIM) равно COUNT(MASK $(s_1, s_2, ..., s_{DIM-1}, :, s_{DIM+1}, ..., s_N)$).

Примеры:

Случай 1: значение COUNT ((/.TRUE., .FALSE., .TRUE./)) равно 2.

Случай 2: если B - массив $\begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix}$, а C - массив $\begin{bmatrix} 0 & 3 & 5 \\ 7 & 4 & 8 \end{bmatrix}$,

COUNT(B.NE.C, DIM=1) равно [2,0,1], а COUNT(B.NE.C, DIM=2) равно [1,2].

13.13.25. CSHIFT (ARRAY,SHIFT,DIM)

Необязательный аргумент: DIM.

Описание: производит циклический сдвиг выражения-массива размерности 1 или циклические сдвиги во всех полных одномерных секциях по заданному измерению выражения-массива с размерностью два или больше. Элементы, выдвинутые с одного конца секции, помещаются в другой ее конец. В различных секциях могут быть сдвиги на различное число элементов и в различных направлениях.

Класс: функция преобразования.

Аргументы:

ARRAY — может быть любого типа, но не должен быть скаляром.

SHIFT — должен быть скаляром целого типа, если ARRAY имеет размерность 1; иначе SHIFT должен быть скаляром или массивом размерности N-1 и конфигурации $(d_1, d_2, ..., d_{DIM-1}, d_{DIM+1}, ..., d_N)$, где $(d_1, d_2, ..., d_N)$- конфигурация ARRAY.

DIM(необязательный) — является скаляром целого типа со значением в интервале 1 <= DIM <= N, где N - размерность ARRAY. Если DIM опущен, подразумевается значение 1.

Тип результата, параметр типа и конфигурация: результат имеет конфигурацию, тип и параметры типа те же, что ARRAY.

Значение результата:

Случай 1: если ARRAY имеет размерность 1, элемент *i* результата равен ARRAY (1 + MODULO (*i* + SHIFT-1, SIZE (ARRAY))).

Случай 2: если ARRAY имеет размерность больше 1, секция $(s_1, s_2, \ldots, s_{DIM-1}, :, s_{DIM+1}, \ldots, s_N))$ результата имеет значение, равное CSHIFT (ARRAY $(s_1, s_2, \ldots, s_{DIM-1}, :, s_{DIM+1}, \ldots, s_N))$ *sh*, 1), где *sh* есть SHIFT или SHIFT$(s_1, s_2, \ldots, s_{DIM-1}, s_{DIM+1}, \ldots, s_N)$

Примеры:

Случай 1: если V - массив [1,2,3,4,5,6], эффект сдвига V циклически влево на две позиции достигается с помощью CSHIFT (V, SHIFT=2), которое имеет значение [3,4,5,6,1,2]; CSHIFT (V, SHIFT=-2) выполняет циклический сдвиг вправо на 2 позиции,и результат имеет значение [5,6,1,2,3,4].

Случай 2: строки массива размерности 2 могут быть все сдвинуты на одинаковое или на разное число позиций. Если M - массив $\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$, значением функции CSHIFT(M, SHIFT=-1, DIM=2) будет $\begin{bmatrix} 3 & 1 & 2 \\ 6 & 4 & 5 \\ 9 & 7 & 8 \end{bmatrix}$.

а значение CSHIFT(M, SHIFT = (/ -1, 1, 0 /), DIM=2) равно $\begin{bmatrix} 3 & 1 & 2 \\ 5 & 6 & 4 \\ 7 & 8 & 9 \end{bmatrix}$.

13.13.26. DATE_AND_TIME (DATE, TIME, ZONE, VALUES)

Необязательные аргументы: DATE, TIME, ZONE, VALUES.

Описание: выдает дату по часам реального времени и дату, совместимую с представлением, определенным в ISO 8601:1988.

Класс: подпрограмма.

Аргументы:

DATE (необязательный) — должен быть скаляром стандартного символьного типа, длина его должна быть, как минимум, 8 для того, чтобы поместилось полное значение. Это аргумент с атрибутом INTENT (OUT). Его левым 8 символам присваивается значение, имеющее вид: *CCYYMMDD*, где *CC* - век, *YY* - год в веке, *MM* - месяц в году, *DD* - день в месяце. Если дата недоступна, они заполняются пробелами.

TIME (необязательный) — должен быть скаляром стандартного символьного типа длиной, как минимум, 10 для того, чтобы поместилось полное значение. Это аргумент с атрибутом INTENT (OUT). Его левым 10 символам присваивается значение, имеющее вид: *HHMMSS.SSS*, где *HH* - час дня, *MM* - минуты в часу, *SS.SSS* - секунды и миллисекунды в минуте. Если часы недоступны, эти позиции заполняются пробелами.

ZONE (необязательный) — должен быть скаляром стандартного символьного типа длиной, как минимум, 5 для того, чтобы поместилось полное значение. Это аргумент с атрибутом INTENT (OUT). Его левым 5 символам присваивается значение, имеющее вид: *±HHMM*, где *HH* и *MM* - разница во времени по отношению к координатному универсальному времени (UTC) в часах и частях часа,

выраженных в минутах, соответственно. Если часы недоступны, эти позиции заполняются пробелами.

VALUES (необязательный) должен быть стандартного целого типа и размерности 1. Это аргумент с атрибутом INTENT (OUT). Его размер должен быть, как минимум, 8. Значениями, выдаваемыми в VALUES, являются следующие:

VALUES(1) год (например, 1990) или -HUGE(0), если дата недоступна.

VALUES(2) месяц года или -HUGE(0), если дата недоступна.

VALUES(3) день месяца или -HUGE(0), если дата недоступна.

VALUES(4) разница по времени относительно координатного универсального времени (UTC) или -HUGE(0), если эта информация недоступна.

VALUES(5) час дня в диапазоне 0 - 23 или -HUGE(0), если нет часов.

VALUES(6) минуты в часу в диапазоне 0 - 59 или -HUGE(0), если часов нет.

VALUES(7) секунды в минуте в диапазоне 0 - 60 или -HUGE(0), если часов нет.

VALUES(8) миллисекунды в секунде в диапазоне 0 - 999 или -HUGE(0), если часов нет.

Пример:

```
INTEGER DATE_TIME (8)
CHARACTER (LEN=10) BIG_BEN (3)
CALL DATE_AND_TIME (BIG_BEN (1), BIG_BEN (2),   &
                    BIG_BEN (3), DATE_TIME)
```

Если подпрограмма вызвана в Женеве 12 апреля 1985 г. в 15:27:35.5, BIG_BEN(1) присвоится значение 19850412, BIG_BEN(2) - значение 152735.500, BIG_BEN(3) - значение +0100 и массиву DATE_TIME - значения: 1985, 4, 12, 60, 15, 27, 35, 500.

Заметим, что UTC определено в CCIR Recommendation 460-2 (и известно также как время по Гринвичу).

13.13.27. DBLE (A)

Описание: преобразование к вещественному типу двойной точности.

Класс: поэлементная функция.

Аргумент: должен иметь тип целый, вещественный или комплексный.

Тип результата и параметр типа: вещественный двойной точности.

Значение результата:

Случай 1: если A - двойной точности, то DBLE(A)=A.

Случай 2: если A имеет тип целый или вещественный, результат имеет такую точность значащей части действительной части A, какую может содержать данное двойной точности.

Случай 3: если A - комплексного типа, результат получает такую точность действительной части A, какую может содержать данное двойной точности.

Пример: DBLE(-3) имеет значение -3.0D0.

13.13.28. DIGITS (X)

Описание: выдает количество значащих цифр в модели, представляющей числа того же типа и с тем же параметром вида, что аргумент.

Класс: справочная функция.

Аргумент: значением X должен быть скаляр или массив целого или вещественного типа.

Тип, параметр типа и конфигурация результата: скаляр стандартного целого типа.

Значение результата: результат имеет значение q, если X целого типа, и значение p если X вещественного типа, где q и p определены в 13.7.1 для модели, представляющей числа того же типа и с тем же параметром вида, что X.

Пример: DIGITS(X) имеет значение 24 для такого вещественного X, модель которого приведена в конце 13.7.1.

13.13.29. DIM (X,Y)

Описание: разность X-Y, если она положительна; иначе – 0.

Класс: поэлементная функция.

Аргументы:

X — должен иметь тип целый или вещественный.

Y — должен быть того же типа, что X, и с тем же параметром вида.

Тип результата и параметр типа: те же, что у X.

Значение результата: значение результата равно X-Y, если X > Y, иначе – 0.

Пример: DIM (-3.0,2.0) имеет значение 0.0.

13.13.30. DOT_PRODUCT (VECTOR_A, VECTOR_B)

Описание: вырабатывает скалярное произведение двух числовых или логических векторов.

Класс: функция преобразования.

Аргументы:

VECTOR_A — должен быть числового (целого, вещественного или комплексного) или логического типа. Он должен быть массивом размерности один.

VECTOR_B — должен быть числового типа, если VECTOR_A числового типа, и логического, если VECTOR_A имеет логический тип. Он должен быть массивом размерности 1. Он должен иметь тот же размер, что и VECTOR_A.

Тип результата, параметр типа и конфигурация: если аргументы имеют числовой тип, тип и параметр вида для результата те же, что для выражения VECTOR_A * VECTOR_B, определяемого типами аргументов в соответствии с 7.1.4. Если аргументы имеют логический тип, результат тоже логический с параметром вида для выражения VECTOR_A .AND. VECTOR_B в соответствии с 7.1.4. Результат является скаляром.

Значение результата:

Случай 1: если VECTOR_A целого или вещественного типа, результат имеет значение SUM (VECTOR_A * VECTOR_B). Если векторы имеют нулевой размер, результат имеет значение нуль.

Случай 2: если VECTOR_A имеет комплексный тип, результат имеет значение SUM (CONJG (VECTOR_A) * VECTOR_B). Если векторы имеют нулевой размер, результат имеет нулевое значение.

Случай 3: если VECTOR_A имеет логический тип, результат имеет значение ANY (VECTOR_A .AND. VECTOR_B). Если векторы имеют нулевой размер, результат имеет значение "ложь"

Пример: DOT_PRODUCT((/1,2,3/),(/2,3,4/)) имеет значение 20.

13.13.31. DPROD (X,Y)

Описание: произведение вещественных двойной точности.

Класс: поэлементная функция.

Аргументы:

X должен быть стандартного вещественного типа.

Y должен быть стандартного вещественного типа.

Тип и параметр типа результата: вещественный двойной точности.

Значение результата: результат имеет значение, равное процессорно-зависимой аппроксимации произведения X на Y.

Пример: DPROD(-3.0, 2.0) имеет значение -6.0D0.

13.13.32. EOSHIFT (ARRAY, SHIFT, BOUNDARY, DIM)

Необязательные аргументы: BOUNDARY, DIM.

Описание: производит сдвиг типа end-off (нециклический) выражения-массива размерности один или производит сдвиг типа end-off во всех полных секциях размерности один вдоль данного измерения выражения-массива с размерностью 2 или больше. Элементы выдвигаются в один конец секции, копии граничного значения помещаются в другой конец. Разные секции могут иметь разные граничные значения и могут быть сдвинуты на разное число позиций и в различных направлениях.

Класс: функция преобразования.

Аргументы:

ARRAY может быть любого типа. Он не должен быть скаляром.

SHIFT должен быть целого типа и должен быть скаляром, если ARRAY имеет размерность 1, иначе он должен быть скаляром или массивом размерности $N-1$ и конфигурации $(d_1, d_2, \ldots, d_{DIM-1}, d_{DIM+1}, \ldots, d_N)$, где $(d_1, d_2, \ldots, d_N)$ - конфигурация ARRAY.

BOUNDARY (необязательный)

должен быть того же типа и с теми же параметрами типа, что и ARRAY, и должен быть скаляром, если ARRAY имеет размерность 1, иначе он должен быть либо скаляром, либо массивом размерности $N-1$ и конфигурации $(d_1, d_2, \ldots, d_{DIM-1}, d_{DIM+1}, \ldots, d_N)$. BOUNDARY может быть опущен для типов данных, перечисленных в нижеследующей таблице, и в этом случае он как бы присутствует со скалярным значением, указанным в этой таблице.

Тип A	Значение BOUNDARY
целый	0
вещественный	0.0
двойной точности	0.0D0
комплексный	(0.0,0.0)
логический	ложь
символьный(*LEN*)	*LEN* пробелов

DIM(необязательный) должен быть скаляром целого типа со значением в интервале $1 <= DIM <= N$, где N - размерность ARRAY. Если DIM опущен, подразумевается его значение 1.

Тип результата, параметр типа и конфигурация: результат имеет конфигурацию, тип и параметр типа, как у ARRAY.

Значение результата: элемент $(s_1, s_2, \ldots, s_N)$ результата имеет значение такое же, как ARRAY $(s_1, s_2, \ldots, s_{DIM-1}, s_{DIM}, s_{DIM+1}, \ldots, s_N)$, где sH - это SHIFT или SHIFT $(s_1, s_2, \ldots, s_{SIM-1}, s_{SIM+1}, \ldots, s_N)$, при условии неравенства LBOUNS (ARRAY, SIM) <= s_{SIM} + sH <= UBOUNS (ARRAY, SIM), иначе - BOUNSARY или BOUNSARY$(s_1, s_2, \ldots, s_{SIM-1}, s_{SIM+1}, \ldots, s_N)$.

Примеры:

Случай 1: если V - массив (1,2,3,4,5,6), то в результате выполнения нециклического сдвига для V влево на 3 позиции с помощью функции EOSHIFT (V, SHIFT=3) получается значение (4,5,6,0,0,0); EOSHIFT(V, SHIFT=-2, BOUNDARY=99) выполняет нециклический сдвиг на две позиции вправо с граничным значением 99 и выдает результат (99,99,1,2,3,4).

Случай 2: строки двумерного массива могут быть сдвинуты либо на одно и то же число позиций, либо на различное число позиций, и граничные элементы могут быть либо одинаковыми, либо разными.

Если M - это массив $\begin{bmatrix} A & B & C \\ D & E & F \\ G & H & I \end{bmatrix}$, то значение функции

EOSHIFT (M, SHIFT=-1, BOUNDARY='*', DIM=2) - это $\begin{bmatrix} * & A & B \\ * & D & E \\ * & G & H \end{bmatrix}$, а значение

EOSHIFT (M, SHIFT = (/-1, 1, 0/), BOUNDARY =(/'* ','/','?'/), DIM = 2) - это $\begin{bmatrix} * & A & B \\ E & F & / \\ G & H & I \end{bmatrix}$.

13.13.33. EPSILON (X)

Описание: выдает положительное модельное число, пренебрежимо малое по сравнению с единицей в модели, представляющей числа того же типа и с тем же параметром вида, что и аргумент.

Класс: справочная функция.

Аргумент: X должен быть вещественного типа. Его значение может быть скаляром или массивом.

Тип результата, параметр типа и конфигурация: скаляр того же типа и с тем же параметром вида, что и X.

Значение результата: результат имеет значение b^{1-p}, где b и p такие, как определено в разделе 13.7.1 для модели, представляющей числа того же типа и с тем же параметром вида, что и X.

Пример: EPSILON(X) имеет значение 2^{-23} для вещественного X, модель которого описана в конце раздела 13.7.1.

13.13.34. EXP (X)

Описание: экспонента (с основанием e)

Класс: поэлементная функция.

Аргумент: X должен быть вещественного типа или комплексного.

Тип результата и параметр типа: те же, что у X.

Значение результата: результат имеет значение, равное процессорно-зависимой аппроксимации числа e^X. Если X комплексного типа, его мнимая часть рассматривается как значение в радианах.

Пример: EXP(1.0) имеет значение 2.7182818 (приближенно).

13.13.35. EXPONENT(X)

Описание: выдает экспонентную часть аргумента, представленного как модельное число.

Класс: поэлементная функция.

Аргумент: X должен быть вещественного типа.

Тип результата: стандартный целый.

Значение результата: результат имеет значение, равное экспоненте e модельного представления (13.7.1) для значения X при условии, что X отличен от нуля, а e находится внутри интервала для стандартного целого типа. Результат не определен, если процессор не может представить e как данное стандартного целого типа. EXPONENT(X) имеет значение 0 при X=0 .

Примеры: EXPONENT(1.0) имеет значение 1, а EXPONENT(4.1) имеет значение 3 для вещественных, чья модель представлена в конце раздела 13.7.1.

13.13.36. FLOOR (A)

Описание: выдает наибольшее целое, которое меньше или равно его аргументу.

Класс: поэлементная функция.

Аргумент: должен быть вещественного типа.

Тип результата и параметр типа: стандартный целый.

Значение результата: результат имеет значение наибольшего целого, которое меньше или равное A. Если процессор не может представить такое значение стандартного целого типа, результат не определен.

Пример: FLOOR(3.7) имеет значение 3. FLOOR(-3.7) имеет значение -4.

13.13.37. FRACTION (X)

Описание: выдает дробную часть модельного представления значения аргумента.

Класс: поэлементная функция.

Аргумент: X должен быть вещественного типа.

Тип результата и параметр типа: те же, что у X.

Значение результата: результат имеет значение $X*b^{-\varepsilon}$ где b и ε такие, как определено в 13.7.1 для модельного представления X. Если X имеет нулевое значение, результат также имеет значение нуль.

Пример: FRACTION (3.0) имеет значение 0.75 для вещественных чисел, модель для которых представлена в конце раздела 13.7.1.

13.13.38. HUGE(X)

Описание: выдает наибольшее число в модели, представляющей числа того же типа и с тем же параметром вида, что и аргумент.

Класс: справочная функция.

Аргумент: X должен быть целого или вещественного типа. Его значением может быть скаляр или массив.

Тип результата, параметр типа и конфигурация: скаляр того же типа и с тем же параметром вида, что и X.

Значение результата: результат имеет значение r^q-1, если X имеет целый тип, и $(1-b^{-p}) * b**\varepsilon_{max}$, если X имеет вещественный тип, где r, q, b, p, ε_{max} такие, как определено в разделе 13.7.1 для модели, представляющей числа того же типа и с тем же параметром типа, что и X.

Пример: HUGE(X) имеет значение $(1-2^{-24})*2^{127}$ для вещественного X, модель которого представлена в конце раздела 13.7.1.

13.13.39. IACHAR(C)

Описание: выдает позицию символа в кодировке ASCII.

Класс: поэлементная функция.

Аргумент: C должен быть символьного типа с длиной, равной единице.

Тип результата и параметр типа: стандартный целый.

Значение результата: если C входит в упорядоченную последовательность, определенную кодами, специфицированными в ISO 646:1983 (International Reference Version), результатом является позиция C в этой последовательности,и она удовлетворяет неравенству (0 <= IACHAR(C) <= 127). Если C нет в коде ASCII, выдается процессорно-зависимое значение. Результат должен согласовываться с функциями лексического сравнения LGE, LGT, LGT и LLE. Например, если LLE (C,D) - "истина", то IACHAR(C) .LE. IACHAR(D) тоже "истина", где C и D - любые два символа, представимые в процессоре.

Пример: IACHAR ('X') имеет значение 88.

13.13.40. IAND (I,J)

Описание: выполняет логический AND.

Класс: поэлементная функция.

Аргументы:

I — должно быть целым.

J — должно быть целым с тем же параметром вида, что у I.

Тип результата и параметр типа: те же, что у I.

Значение результата: результат имеет значение, которое получается поразрядной комбинацией I и J в соответствии со следующей таблицей истинности:

I	J	IAND (I,J)
1	1	1
1	0	0
0	1	0
0	0	0

Модель для интерпретации целого значения как последовательности битов приведена в 13.5.7.

Пример: IAND (1,3) имеет значение 1.

13.13.41. IBCLR (I,POS)

Описание: обнуление одного бита.

Класс: поэлементная функция.

Аргументы:

I — должно быть целым.

POS — должно быть неотрицательным целым, меньшим, чем BIT_SIZE(I).

Тип результата и параметр типа: те же, что у I.

Значение результата: значение результата - последовательность битов I, но POS-й бит I установлен в нуль. Модель для интерпретации целого значения как последовательности битов приведена в 13.5.7.

Примеры: IBCLR (14, 1) имеет результат 12. Если V имеет значение [1,2,3,4], значением IBCLR (POS=V, I=31) является [29,27,23,15].

13.13.42. IBITS (I, POS, LEN)

Описание: выделяет последовательность битов.

Класс: поэлементная функция.

Аргументы:

I	должно быть целым.
POS	должно быть неотрицательным целым и POS+LEN должно быть меньше или равно BIT_SIZE(I).
LEN	должно быть положительным целым.

Тип результата и параметр типа: те же, что у I.

Значение результата: значением результата является последовательность из LEN битов I, начиная с POS-го справа бита, причем все остальные биты - нулевые. Модель для интерпретации целого значения как последовательности битов приведена в 13.5.7.

Пример: IBITS(14,1,3) имеет значение 7.

13.13.43. IBSET (I,POS)

Описание: устанавливает один бит равным единице.

Класс: поэлементная функция.

Аргументы:

I	должно быть целого типа.
POS	должно быть неотрицательным целым, меньшим, чем BIT_SIZE(I).

Тип результата и параметр типа: те же, что у I.

Значение результата: значением результата является последовательность битов I, но POS-й бит I имеет значение - единица. Модель для интерпретации целого значения как последовательности битов приведена в 13.5.7.

Примеры: IBSET(12,1) имеет значение 14. Если V имеет значение [1,2,3,4], значение IBSET (POS=V, I=0) есть [2,4,8,16].

13.13.44. ICHAR (C)

Описание: выдает позицию символа в кодировке процессора, соответствующей параметру типа аргумента.

Класс: поэлементная функция.

Аргумент: C должен быть символьного типа с длиной, равной единице. Его значение должно быть символом, имеющим представление в процессоре.

Тип результата и параметр типа: стандартный целый.

Значение результата: результатом является позиция C в кодировке процессора, соответствующей параметру вида C. Результат находится в интервале 0 <= ICHAR(C) <= N-1, где N - количество символов в кодировке процессора. Для любых символов C и D, представимых в процессоре, C .LE. D истинно тогда и только тогда, когда истинно ICHAR(C) .LE. ICHAR(D) , а C .EQ. D истинно, только если истинно ICHAR(C) .EQ. ICHAR(D).

Пример: ICHAR('X') имеет значение 88 в процессоре, использующем кодировку ASCII для стандартного символьного типа.

13.13.45. IEOR (I,J)

Описание: выполняет исключающее OR.

Класс: поэлементная функция.

Аргументы:

I — должно быть целого типа.

J — должно быть целым с тем же параметром вида, что у I.

Тип результата и параметр типа: те же, что у I.

Значение результата: результат имеет значение, полученное поразрядной комбинацией I и J в соответствии со следующей таблицей истинности:

I	J	IEOR (I,J)
1	1	0
1	0	1
0	1	1
0	0	0

Модель для интерпретации целого значения как последовательности битов приведена в 13.5.7.

Пример: IEOR (1,3) имеет значение 2.

13.13.46. INDEX (STRING, SUBSTRING, BACK)

Необязательный аргумент: BACK.

Описание: выдает начальную позицию подстроки внутри строки.

Класс: поэлементная функция.

Аргументы:

STRING — должен быть символьного типа.

SUBSTRING — должен быть символьного типа и с тем же параметром типа, что у STRING.

BACK (необязательный) — должен быть логического типа.

Тип результата и параметр типа: стандартный целый.

Значение результата:

Случай 1: если BACK отсутствует или присутствует со значением "ложь", выдаваемый результат - это минимальное положительное значение I, такое, что STRING (I:I + LEN (SUBSTRING) - 1) = SUBSTRING, либо нуль, если такого значения нет. Если LEN(STRING) < LEN(SUBSTRING), выдается нуль; если LEN (SUBSTRING) = 0, выдается единица.

Случай 2: если присутствует BACK со значением "истина", результатом является максимальное значение I, меньше или равное LEN (STRING)-LEN (SUBSTRING)+1, такое, что STRING (I:I + LEN (SUBSTRING)-1)=SUBSTRING, либо нуль, если такого значения нет. Если LEN (STRING) < LEN (SUBSTRING), выдается нуль, а если LEN (SUBSTRING) = 0, выдается LEN (STRING) + 1.

Примеры: INDEX('FORTRAN','R') имеет значение 3.

INDEX('FORTRAN','R',BACK=.TRUE.) имеет значение 5.

13.13.47. INT (A, KIND)

Необязательный аргумент: KIND.

Описание: преобразование в целый тип.

Класс: поэлементная функция.

Аргументы:

A — должен быть целого, вещественного или комплексного типа.

KIND (необязательный) — должен быть скалярным целым выражением инициализации.

Тип результата и параметр типа: целый. Параметр вида задается в KIND, если он присутствует, иначе - такой, как для стандартного целого типа.

Значение результата:

Случай 1: если A имеет целый тип, то INT(A) = A.

Случай 2: если A-вещественного типа,то имеются два случая: если |A| < 1, INT(A) = 0, если |A| >= 1, то INT(A) - это целое, значение которого есть наибольшее целое, не превышающее модуля A, и знак которого совпадает со знаком A.

Случай 3: если A комплексного типа, INT(A) - это значение, полученное применением правила случая 2 к действительной части A.

Если процессор не может представить результат заданного целого типа, результат не определен.

Пример: INT(-3.7) имеет значение -3.

13.13.48. IOR (I,J)

Описание: выполняет включающее OR.

Класс: поэлементная функция.

Аргументы:

I — должно быть целого типа.

J — должно быть целым с тем же параметром вида, что у I.

Тип результата и параметр типа: те же, что у I.

Значение результата: результат имеет значение, полученное поразрядной комбинацией I и J в соответствии со следующей таблицей истинности:

I	J	IOR (I,J)
1	1	1
1	0	1
0	1	1
0	0	0

Модель для интерпретации целого значения как последовательности битов приведена в 13.5.7.

Пример: IOR (1,3) имеет значение 3.

13.13.49. ISHFT (I, SHIFT)

Описание: выполняет логический сдвиг.

Класс: поэлементная функция.

Аргументы:

I — должно быть целым.

SHIFT — должно быть целым. Абсолютное значение SHIFT должно быть меньше или равно BIT_SIZE(I)

Тип результата и параметр типа: те же, что у I.

Значение результата: результат имеет значение, получаемое сдвигом битов в I на SHIFT позиций. Если SHIFT положительно, сдвиг влево; если отрицательно - вправо; если SHIFT равно нулю,- сдвиг не выполняется. Биты, выдвинутые справа или слева, пропадают. На противоположный конец выдвигаются нули. Модель для интерпретации целого значения как последовательности битов приведена в 13.5.7.

Пример: ISHFT (3,1) имеет результат 6.

13.13.50. ISHFTC (I,SHIFT,SIZE)

Необязательный аргумент: SIZE.

Описание: выполняет циклический сдвиг самых правых битов.

Класс: поэлементная функция.

Аргументы:

I	должно быть целым.
SHIFT	должно быть целым. Абсолютное значение SHIFT должно быть меньше или равно SIZE.
SIZE (необязательный)	должно быть целым. Значение SIZE должно быть положительным и не больше, чем BIT_SIZE(I). Если SIZE опущено, подразумевается его значение, равное BIT_SIZE(I).

Тип результата и параметр типа: те же, что у I.

Значение результата: результат имеет значение, получаемое циклическим сдвигом SIZE самых правых битов в I на SHIFT позиций. Если SHIFT положительно, сдвиг влево, если отрицательно - вправо; если SHIFT равно нулю - сдвиг не выполняется. Ни один бит не теряется. Несдвинутые биты не меняются. Модель для интерпретации целого значения как последовательности битов приведена в 13.5.7.

Пример: ISHFTC (3,2,3) имеет результат 5.

13.13.51. KIND (X)

Описание: выдает значение параметра вида для X.

Класс: справочная функция.

Аргумент: X может быть любого встроенного типа.

Тип результата, параметр типа и конфигурация: скаляр стандартного целого типа.

Значение результата: результат имеет значение, равное значению параметра вида для X.

Пример: KIND(0.0) имеет значение параметра вида для стандартного вещественного типа.

13.13.52. LBOUND (ARRAY,DIM)

Необязательный аргумент: DIM.

Описание: выдает все нижние границы массива или требуемую нижнюю границу.

Класс: справочная функция.

Аргументы:

ARRAY	может быть любого типа. Не должен быть скаляром, не должен быть указателем, который не связан, или динамически размещаемым массивом, который не размещен.
DIM (необязательный)	должен быть скаляром целого типа со значением в интервале 1 <= DIM <= *N*, где *N* - размерность ARRAY. Соответствующий фактический аргумент не должен быть необязательным формальным аргументом.

Тип результата, параметр типа и конфигурация: результат имеет стандартный целый тип; это скаляр, если DIM присутствует, а иначе результатом является массив размерности 1 и размера *N*, где *N* - размерность ARRAY.

Значение результата:

Случай 1: для секции массива или для выражения-массива, не являющегося ни полным массивом, ни компонентом структуры, LBOUND(ARRAY,DIM) имеет значение 1, в противном случае это значение равно нижней границе для индекса DIM в

ARRAY (если измерение DIM в ARRAY не имеет нулевого размера), и значение 1, если измерение DIM имеет нулевой размер.

Случай 2: LBOUND(ARRAY) имеет значение, *i*-й компонент которого равен LBOUND (ARRAY,*i*) для *i*=1,2,...,*n*, где *n* -размерность ARRAY.

Примеры: если A объявлено в операторе

```
REAL A (2:3, 7:10)
```

то LBOUND (A) есть [2,7], а LBOUND (A, DIM=2) есть 7.

13.13.53. LEN (STRING)

Описание: выдает длину символьного объекта.

Класс: справочная функция.

Аргумент: STRING должен быть символьного типа. Его значением может быть скаляр или массив.

Тип результата, параметр типа и конфигурация: скаляр стандартного целого типа.

Значение результата: результат имеет значение, равное количеству символов в STRING, если это скаляр, или в элементе STRING, если это значение – массив.

Пример: если C объявлено в операторе

```
CHARACTER (11) C (100)
```

то LEN(C) имеет значение 11.

13.13.54. LEN_TRIM (STRING)

Описание: выдает длину символьного аргумента без учета конечных пробелов.

Класс: поэлементная функция.

Аргумент: STRING должен быть символьного типа.

Тип результата и параметр типа: стандартный целый.

Значение результата: результат имеет значение, равное количеству символов, оставшихся после удаления из STRING всех конечных пробелов. Если аргумент не содержит символов, отличных от пробела, результатом является нуль.

Примеры: LEN_TRIM(' A B ') равно 4, а LEN_TRIM(' ') равно 0.

13.13.55. LGE (STRING_A,STRING_B)

Описание: проверяет на основе кодировки ASCII, является ли одна строка равной или лексически большей, чем другая строка.

Класс: поэлементная функция.

Аргументы:

STRING_A	должен быть стандартного символьного типа.
STRING_B	должен быть стандартного символьного типа.

Тип результата и параметры типа: стандартный логический.

Значение результата: если строки неравной длины, сравнение производится так, как если бы более короткая строка была расширена вправо пробелами до длины более длинной строки. Если какая-либо строка содержит символ, не принадлежащий набору символов ASCII, результат зависит от процессора. Результат - "истина", если строки равны или если STRING_A следует за STRING_B в кодировке ASCII, иначе результат - "ложь". Заметим, что если обе строки нулевой длины, результатом является "истина".

Пример: LGE('ONE','TWO') имеет значение "ложь".

13.13.56. LGT (STRING_A, STRING_B)

Описание: проверяет на основе кодировки ASCII, является ли строка лексически большей, чем другая строка.

Класс: поэлементная функция.

Аргументы:

STRING_A должен быть стандартного символьного типа.

STRING_B должен быть стандартного символьного типа.

Тип результата и параметры типа: стандартный логический.

Значение результата: если строки неравной длины, сравнение производится как если бы более короткая строка была расширена вправо пробелами до длины более длинной строки. Если какая-либо из строк содержит символ, не принадлежащий набору символов ASCII, результат зависит от процессора. Результатом является "истина", если STRING_A следует за STRING_B в кодировке ASCII, иначе результатом является "ложь". Заметим, что если обе строки нулевой длины, результатом является "ложь".

Пример: LGT('ONE','TWO') имеет значение "ложь".

13.13.57. LLE (STRING_A, STRING_B)

Описание: проверяет на основе кодировки ASCII, является ли строка лексически меньшей или равной другой строке.

Класс: поэлементная функция.

Аргументы:

STRING_A должен быть стандартного символьного типа.

STRING_B должен быть стандартного символьного типа.

Тип результата и параметры типа: стандартный логический.

Значение результата: если строки неравной длины, сравнение производится как если бы более короткая строка была расширена вправо пробелами до длины более длинной строки. Если какая-либо из строк содержит символ, не принадлежащий набору символов ASCII, результат зависит от процессора. Результатом является "истина", если строки равны или если STRING_A предшествует STRING_B в кодировке ASCII, иначе результатом является "ложь". Заметим, что если обе строки нулевой длины, результат - "истина".

Пример: LLE ('ONE,'TWO') имеет значение "истина".

13.13.58. LLT (STRING_A, STRING_B)

Описание: проверяет на основе кодировки ASCII, является ли строка лексически меньшей, чем другая строка.

Класс: поэлементная функция.

Аргументы:

STRING_A должен быть стандартного символьного типа.

STRING_B должен быть стандартного символьного типа.

Тип результата и параметры типа: стандартный логический.

Значение результата: если строки неравной длины, сравнение производится как если бы более короткая строка была расширена вправо пробелами до длины более длинной строки. Если какая-либо строка содержит символ, не принадлежащий набору символов ASCII, результат зависит от процессора. Результатом является "истина", если STRING_B следует за STRING_A в кодировке ASCII, иначе результатом является "ложь". Заметим, что если обе строки нулевой длины, результат - "ложь".

Пример: LLT('ONE','TWO') имеет значение "истина".

13.13.59. LOG (X)

Описание: натуральный логарифм.

Класс: поэлементная функция.

Аргумент: X должен быть вещественного или комплексного типа. Если X - вещественный, его значение должно быть больше нуля. Если X - комплексный, его значение не должно быть нулем.

Тип результата и параметр типа: те же, что у X.

Значение результата: результат имеет значение, равное процессорно-зависимой аппроксимации $log_e X$. Результат для комплексного типа - это главное значение с мнимой частью w, находящейся в интервале $-\pi < w <= \pi$. Мнимая часть результата равна π, только когда действительная часть аргумента меньше нуля, а мнимая часть аргумента равна нулю.

Пример: LOG(10.0) имеет значение 2.3025851 (приближенно).

13.13.60. LOG10 (X)

Описание: общий логарифм.

Класс: поэлементная функция.

Аргумент: X должен быть вещественным, значение которого больше нуля.

Тип результата и параметр типа: те же, что у X.

Значение результата: результат имеет значение, равное процессорно-зависимой аппроксимации $log_{10} X$.

Пример: LOG10(10.0) имеет значение 1.0 (приближенно).

13.13.61. LOGICAL (L, KIND)

Необязательный аргумент: KIND.

Описание: изменение параметра вида для логических аргументов.

Класс: поэлементная функция.

Аргументы:

L	должен быть логическим.
KIND (необязательный)	должен быть скалярным целым выражением инициализации.

Тип результата и параметр типа: логический. Если KIND присутствует, параметр вида такой, как специфицировано в KIND, иначе - такой, как для стандартного логического типа.

Значение результата: значение равно L.

Пример: LOGICAL (L .OR. .NOT. L) имеет значение "истина" и стандартный логический тип независимо от параметра вида логической переменной L.

13.13.62. MATMUL (MATRIX_A, MATRIX_B)

Описание: производит умножение числовых или логических матриц.

Класс: функция преобразования.

Аргументы:

MATRIX_A	должен быть числового (целого, вещественного или комплексного) или логического типа. Он должен быть значением-массивом размерности 1 или 2. Его конфигурация должна быть определена.

MATRIX_B — должен быть числового типа, если MATRIX_A числового типа, и логического типа, если MATRIX_A - логический. Его значением должен быть массив размерности 1 или 2. Если размерность MATRIX_A равна 1, размерность MATRIX_B должна быть равна 2. Если размерность MATRIX_B равна 1, размерность MATRIX_A должна быть равна 2. Размер первого (или единственного) измерения MATRIX_B должен быть равным размеру последнего (или единственного) измерения MATRIX_A.

Тип результата, параметр типа и конфигурация: если аргументы числового типа, тип и параметр вида результата определяются типами аргументов в соответствии с 7.1.4.2. Если аргументы логического типа, результат имеет логический тип с параметром вида в соответствии с 7.1.4.2. Конфигурация результата зависит от конфигурации аргументов следующим образом:

Случай 1: если MATRIX_A имеет конфигурацию (*n,m*), а MATRIX_B - конфигурацию (*m,k*), конфигурация результата - (*n,k*);

Случай 2: если MATRIX_A имеет конфигурацию (*m*), а MATRIX_B - (*m,k*), конфигурация результата - (*k*);

Случай 3: если MATRIX_A имеет конфигурацию (*n,m*), а MATRIX_B - (*m*), конфигурация результата - (*n*).

Значение результата:

Случай 1: элемент (*i,j*) результата имеет значение SUM (MATRIX_A (*i,:*) * MATRIX_B (:,*j*)), если аргументы - числового типа, и значение ANY (MATRIX_A (*i,:*) .AND. MATRIX_B (:,*j*)), если аргументы - логического типа.

Случай 2: элемент (*j*) результата имеет значение SUM (MATRIX_A(:) * MATRIX_B (:,*j*)), если аргументы - числового типа, и значение ANY (MATRIX_A (:) .AND. MATRIX_B (:,*j*)), если аргументы логического типа.

Случай 3: элемент (*i*) результата имеет значение SUM (MATRIX_A (*i,:*) * MATRIX_B(:)), если аргументы числового типа, и значение ANY (MATRIX (*i,:*) .AND. MATRIX_B (:)), если аргументы логического типа.

Примеры: пусть A и B - матрицы $\begin{bmatrix} 1 & 2 & 3 \\ 2 & 3 & 4 \end{bmatrix}$ и $\begin{bmatrix} 1 & 2 \\ 2 & 3 \\ 3 & 4 \end{bmatrix}$;

пусть X и Y - векторы [1,2] и [1,2,3].

Случай 1: результат MATMUL (A,B) равен произведению матрицы на матрицу AB со значением $\begin{bmatrix} 14 & 20 \\ 20 & 29 \end{bmatrix}$.

Случай 2: результат MATMUL (X,A) равен произведению вектора на матрицу XA со значением [5 8 11].

Случай 3: результат MATMUL (A,Y) равен произведению матрицы на вектор AY со значением [14 20].

13.13.63. MAX (A1, A2, A3,...)

Необязательные аргументы: A3,...

Описание: максимальное значение.

Класс: поэлементная функция.

Аргументы: аргументы все должны быть одного и того же типа, целого или вещественного, с одинаковыми параметрами вида.

Тип результата и параметр типа: те же, что у аргументов.

Значение результата: значение результата равно наибольшему аргументу.

Пример: MAX(-9.0,7.0,2.0) имеет значение 7.0.

13.13.64. MAXEXPONENT (X)

Описание: выдает максимальную экспоненту в модели, представляющей числа того же типа и с тем же параметром вида, что у аргумента.

Класс: справочная функция.

Аргумент: X должен быть вещественного типа. Его значением может быть скаляр или массив.

Тип результата, параметр типа и конфигурация: скаляр стандартного целого типа.

Значение результата: результат имеет значение ε_{MAX}, как определено в 13.7.1 для модели, представляющей числа того же типа и с тем же параметром вида, что у X.

Пример: MAXEXPONENT(X) имеет значение 127 для вещественного X, модель которого описана в конце 13.7.1.

13.13.65. MAXLOC (ARRAY, MASK)

Необязательный аргумент: MASK.

Описание: определяет расположение первого элемента ARRAY, имеющего максимальное значение среди элементов, идентифицированных с помощью аргумента MASK.

Класс: функция преобразования.

Аргументы:

ARRAY	должен быть целого или вещественного типа. Он не должен быть скаляром.
MASK (необязательный)	должен быть логического типа и должен согласовываться с ARRAY.

Тип результата, параметр типа и конфигурация: результат имеет стандартный целый тип; он является массивом размерности 1, а размер его равен размерности ARRAY.

Значение результата:

Случай 1: если MASK отсутствует, результатом является одномерный массив, значениями элементов которого являются значения индексов того элемента ARRAY, значение которого является максимальным среди значений всех элементов ARRAY. Выдаваемый индекс i находится в интервале от 1 до ε_i где ε_i - протяженность i-го измерения ARRAY. Если более чем один элемент имеет максимальное значение, выдаются индексы первого такого элемента в обычном для массива порядке. Если ARRAY имеет нулевой размер, значение результата зависит от процессора.

Случай 2: если MASK присутствует, результат - это одномерный массив, значениями элементов которого являются значения индексов элемента ARRAY, соответствующего элементу MASK со значением "истина" и имеющему максимальное значение среди всех таких элементов ARRAY. Выдаваемый индекс i находится в интервале от 1 до ε_i где ε_i - протяженность i-го измерения ARRAY. Если более чем один такой элемент имеет максимальное значение, выдаются индексы первого такого элемента в обычном для массива порядке. Если нет таких элементов (то есть, если ARRAY имеет нулевой размер или все элементы MASK имеют значение "ложь"), значение результата процессорно-зависимо.

Элемент результата не определен, если процессор не может представить значение как значение стандартного целого типа.

Примеры:

Случай 1: значение MAXLOC ((/2,6,4,6/)) равно [2].

Случай 2: если A имеет значение $\begin{bmatrix} 0 & -5 & 8 & -3 \\ 3 & 4 & -1 & 2 \\ 1 & 5 & 6 & -4 \end{bmatrix}$.

MAXLOC (A, MASK = A.LT.6) имеет значение [3 , 2]. Заметим, что это верно, даже если A объявлен с нижней границей, отличной от 1.

13.13.66. MAXVAL (ARRAY,DIM,MASK)

Необязательные аргументы: DIM, MASK.

Описание: максимальное значение среди всех элементов ARRAY по измерению DIM, соответствующих элементам MASK со значением "истина".

Класс: функция преобразования.

Аргументы:

ARRAY	должен быть целого или вещественного типа. Он не должен быть скаляром.
DIM (необязательный)	должен быть скаляром целого типа со значением в интервале 1 <= DIM <= N, где N - размерность ARRAY. Соответствующий фактический аргумент не может быть необязательным формальным аргументом.
MASK (необязательный)	должен быть логического типа и должен быть согласован с ARRAY.

Тип результата, параметр типа и конфигурация: результат имеет тот же тип и параметр вида, что и ARRAY; он является скаляром, если DIM отсутствует или ARRAY имеет размерность один; в противном случае он является массивом размерности N-1 и конфигурации $(d_1, d_2, ..., d_{DIM-1}, d_{DIM+1}, ..., d_N)$, где $(d_1, d_2, ..., d_N)$ - конфигурация ARRAY.

Значение результата:

Случай 1: результат MAXVAL (ARRAY) имеет значение, равное максимальному значению из всех элементов ARRAY, либо равно взятому со знаком "-" значению максимального числа, представимого процессором для данных того же типа и с тем же параметром вида, что у ARRAY, если ARRAY имеет нулевой размер.

Случай 2: результат MAXVAL (ARRAY, MASK=MASK) имеет значение, равное максимальному значению среди элементов ARRAY, соответствующих элементам MASK со значением "истина", либо, если нет элементов со значением "истина", равно взятому со знаком "-" значению максимального числа, представимого процессором для данных того же типа и с тем же параметром вида, что у ARRAY.

Случай 3: если ARRAY имеет размерность 1, MAXVAL (ARRAY, DIM [,MASK]) имеет значение, равное значению MAXVAL (ARRAY [,MASK=MASK]). Иначе значение элемента $(s_1, s_2, ..., s_{DIM-1}, s_{DIM+1}, ..., s_N)$ для MAXVAL (ARRAY, DIM [,MASK]) равно MAXVAL (ARRAY$(s_1, s_2, ..., s_{DIM-1}, :, s_{DIM+1}, ..., s_N)$ [,MASK = MASK$(s_1, s_2, ..., s_{DIM-1}, :, s_{DIM+1}, ..., s_N)$]).

Примеры:

Случай 1: значение MAXVAL ((/1,2,3/)) равно 3.

Случай 2: MAXVAL(C, MASK = C .LT. 0.0) находит максимальный из отрицательных элементов C.

Случай 3: если B - массив $\begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix}$, MAXVAL (B, DIM=1) равен [2 , 4 , 6],

а MAXVAL (B, DIM=2) равен [5 , 6].

13.13.67. MERGE (TSOURCE, FSOURCE, MASK)

Описание: выбирает альтернативное значение в соответствии со значением маски.

Класс: поэлементная функция.

Аргументы:

TSOURCE	может быть любого типа.
FSOURCE	должен быть того же типа и с теми же параметрами типа, что и TSOURCE.
MASK	должен быть логического типа.

Тип результата и параметры типа: те же, что у TSOURCE.

Значение результата: результат - это TSOURCE, если MASK = "истина", и FSOURCE в противном случае.

Примеры: если TSOURCE - массив $\begin{bmatrix} 1 & 6 & 5 \\ 2 & 4 & 6 \end{bmatrix}$, FSOURCE - массив $\begin{bmatrix} 0 & 3 & 2 \\ 7 & 4 & 8 \end{bmatrix}$,

MASK- массив $\begin{bmatrix} T & \bullet & T \\ \bullet & \bullet & T \end{bmatrix}$, где T изображает "истина" а $\bullet$ - "ложь",

тогда MERGE (TSOURCE, FSOURCE, MASK) есть $\begin{bmatrix} 1 & 3 & 5 \\ 7 & 4 & 6 \end{bmatrix}$.

Значение MERGE (1.0, 0.0, K>0) есть 1.0 для K = 5 и 0.0 для K = -2.

13.13.68. MIN (A1,A2,A3,...)

Необязательные аргументы: A3,...

Описание: минимальное значение.

Класс: поэлементная функция.

Аргументы: аргументы должны все быть одного и того же типа, целого или вещественного, с одинаковым параметром вида.

Тип результата и параметр вида: те же, что и аргументов.

Значение результата: значение результата равно значению наименьшего из аргументов.

Пример: MIN (-9.0, 7.0, 2.0) имеет значение -9.0.

13.13.69. MINEXPONENT (X)

Описание: выдает минимальную (наибольшую отрицательную) экспоненту в модели, представляющей числа того же типа и с тем же параметром вида, что у аргумента.

Класс: справочная функция.

Аргумент: X должен быть вещественным. Его значением может быть скаляр или массив.

Тип результата, параметр типа и конфигурация: скаляр стандартного целого типа.

Значение результата: результат имеет значение ε_{MIN}, как определено в 13.7.1 для модели, представляющей числа того же типа и с тем же параметром вида, что у X.

Пример: MINEXPONENT(X) имеет значение -126 для вещественного X, модель которого представлена в конце 13.7.1.

13.13.70. MINLOC (ARRAY,MASK)

Необязательный аргумент: MASK.

Описание: определяет расположение первого элемента ARRAY, имеющего минимальное значение среди элементов, идентифицированных с помощью MASK.

Класс: функция преобразования.

Аргументы:

ARRAY — должен быть целого или вещественного типа. Он не должен быть скаляром.

MASK(необязательный) — должен быть логического типа и должен согласовываться с ARRAY.

Тип результата, параметр типа и конфигурация: результат имеет стандартный целый тип; это массив размерности 1 и размера, равного размерности ARRAY.

Значение результата:

Случай 1: если MASK отсутствует, результат - это одномерный массив, значения элементов которого - это значения индексов элемента ARRAY с минимальным значением. Выдаваемый индекс i находится в интервале от 1 до ε_i, где ε_i - протяженность i-го измерения ARRAY. Если более чем один элемент имеет минимальное значение, выдаются индексы первого (в обычном для массива порядке) такого элемента. Если ARRAY имеет нулевой размер, значение результата процессорно-зависимо.

Случай 2: если MASK присутствует, результат - это одномерный массив, значения элементов которого являются значениями индексов того из элементов ARRAY, соответствующих элементу MASK со значением "истина", чье значение минимально. Выдаваемый индекс i находится в интервале от 1 до ε_i, где ε_i - протяженность i-го измерения ARRAY. Если несколько элементов имеют минимальное значение, выдается первый из них (в обычном для массивов порядке). Если ARRAY имеет нулевой размер или все элементы MASK имеют значение "ложь", значение результата процессорно-зависимо.

Элемент результата не определен, если процессор не может представить значение как число стандартного целого типа.

Примеры:

Случай 1: значение MINLOC ((/4,3,6,3/)) равно [2].

Случай 2: если A имеет значение $\begin{bmatrix} 0 & -5 & 8 & -3 \\ 3 & 4 & -1 & 2 \\ 1 & 5 & 6 & -4 \end{bmatrix}$,

то MINLOC (A, MASK = A.GT.-4) равно [1 , 4]. Заметим, что это верно, даже если объявленная нижняя граница A не равна 1.

13.13.71. MINVAL (ARRAY, DIM, MASK)

Необязательные аргументы: DIM, MASK.

Описание: минимальное значение из всех элементов ARRAY по измерению DIM, соответствующих элементам MASK со значением "истина".

Класс: функция преобразования.

Аргументы:

ARRAY — должен быть целого или вещественного типа. Он не должен быть скаляром.

DIM (необязательный) — должен быть скаляром целого типа со значением в интервале 1 <= DIM <= N, где N - размерность ARRAY. Соответствующий фактический аргумент не должен быть необязательным формальным аргументом.

MASK (необязательный) — должен иметь тип логический и должен быть согласован с ARRAY.

Тип результата, параметр типа и конфигурация: результат имеет тот же тип и параметр вида, что ARRAY; он является скаляром, если DIM отсутствует или ARRAY имеет размерность, равную единице; иначе он является массивом размерности N-1 и конфигурации $(d_1, d_2, ..., d_{DIM-1}, d_{DIM+1}, ..., d_N)$, где $(d_1, d_2, ..., d_N)$ - конфигурация ARRAY.

Значение результата:

Случай 1: результат MINVAL (ARRAY) имеет значение, равное минимальному значению среди всех элементов ARRAY, или наибольшему поддержанному процессором положительному числу того же типа и с тем же параметром вида, что у ARRAY, если ARRAY имеет нулевой размер.

Случай 2: результат MINVAL (ARRAY, MASK=MASK) имеет значение, равное минимальному значению среди элементов ARRAY, соответствующих элементам MASK со значением "истина", или, если нет элементов со значением "истина", наибольшему поддержанному процессором положительному числу того же типа и с тем же параметром вида, что у ARRAY.

Случай 3: если ARRAY имеет размерность 1, MINVAL (ARRAY, DIM [,MASK]) имеет значение MINVAL (ARRAY [,MASK=MASK]), иначе значение элемента $(s_1, s_2, ..., s_{DIM-1}, s_{DIM+1}, ..., s_N)$ для MINVAL (ARRAY, DIM [,MASK]) равен MINVAL (ARRAY $(s_1, s_2, ..., s_{DIM-1}, :, s_{DIM+1}, ..., s_N)$ [,MASK=MASK $(s_1, s_2, ..., s_{DIM-1}, :, s_{DIM+1}, ..., s_N)$]).

Примеры:

Случай 1: значение MINVAL ((/1,2,3/)) равно 1.

Случай 2: MINVAL (C, MASK =C.GT.0.0) формирует минимум среди положительных элементов C.

Случай 3: если B - массив $\begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix}$.

MINVAL (B,DIM=1) - [1 3 5], то MINVAL (B, DIM=2) - [1 2].

13.13.72. MOD (A,P)

Описание: функция остатка.

Класс: поэлементная функция.

Аргументы:

A — должен быть целого или вещественного типа.

P — должен быть того же типа с тем же параметром вида, что A.

Тип результата и параметр типа: те же, что у A.

Значение результата: если P /= 0, значение результата равно A - INT (A/P) * P. Если P = 0, результат зависит от процессора.

Примеры: MOD (3.0, 2.0) имеет значение 1.0 (приближенно). MOD (8, 5) имеет значение 3. MOD (-8, 5) имеет значение -3. MOD (8, -5) имеет значение 3. MOD(-8, -5) имеет значение -3.

13.13.73. MODULO (A,P)

Описание: функция "по модулю".

Класс: поэлементная функция.

Аргументы:

A — должен быть целого или вещественного типа.

P — должен быть того же типа с тем же параметром вида, что A.

Тип результата и параметр типа: те же, что у A.

Значение результата:

Случай 1: A - целого типа. Если P /= 0, MODULO (A,P) имеет значение R, такое, что A = Q * P + R, где Q - целое, и соблюдаются неравенства: 0 <= R < P при P>0 и P < R <= 0 при P<0. Если P = 0, результат зависит от процессора.

Случай 2: A - вещественное. Если P /= 0, значение результата есть A - FLOOR (A/P) * P. Если P= 0, результат зависит от процессора.

Примеры: MODULO (8, 5) имеет значение 3. MODULO (-8, 5) имеет значение 2. MODULO (8, -5) имеет значение -2. MODULO (-8, -5) имеет значение -3.

13.13.74. MVBITS (FROM, FROMPOS, LEN, TO, TOPOS)

Описание: копирует последовательность битов из одного объекта данных в другой.

Класс: поэлементная подпрограмма.

Аргументы:

FROM — должно быть целого типа. Это аргумент с атрибутом INTENT(IN).

FROMPOS — должно быть неотрицательным целым. Это аргумент с атрибутом INTENT(IN). FROMPOS + LEN должно быть меньше или равно BIT_SIZE(FROM). Модель для интерпретации целого значения как последовательности битов приведена в 13.5.7.

LEN — должно быть неотрицательным целым. Это аргумент с атрибутом INTENT(IN).

TO — должно быть целой переменной с тем же параметром вида, что у FROM, и может вообще совпадать с FROM. Это аргумент с атрибутом INTENT (INOUT). Последовательность битов длиной LEN из FROM, начиная с позиции FROMPOS, копируется в TO, начиная с позиции TOPOS. Остальные биты в TO не изменяются. На выходе LEN битов в TO, начиная с позиции TOPOS, равны значению LEN битов FROM при входе, начиная с позиции FROMPOS. Модель для интерпретации целого значения как последовательности битов приведена в 13.5.7.

TOPOS — должно быть неотрицательным целым. Это аргумент с атрибутом INTENT (IN). TOPOS+LEN должно быть меньше или равно BIT_SIZE (TO).

Пример: если TO имеет начальное значение 6, то после выполнения оператора CALL MVBITS (7,2,2,TO,0) его значение равно 5.

13.13.75. NEAREST (X, S)

Описание: выдает ближайшее представимое в машине число в данном направлении.

Класс: поэлементная функция.

Аргументы:

X — должен быть вещественным.

S — должен быть вещественным и не равным нулю.

Тип результата и параметр типа: те же, что у X.

Значение результата: результатом является значение, равное представимому в машине числу, отличному от X и ближайшему к нему в направлении к бесконечности с тем же знаком, что у S.

Пример: NEAREST (3.0, 2.0) имеет значение $3+2^{-22}$ на машине, имеющей ту же модель, что описана в конце 13.7.1.

13.13.76. NINT (A, KIND)

Необязательный аргумент: KIND.

Описание: ближайшее целое.

Класс: поэлементная функция.

Аргументы:

A	должен быть вещественного типа.
KIND (необязательный)	должен быть скалярным целым выражением инициализации.

Тип результата и параметр типа: целый. Параметр вида такой, как специфицировано в KIND (если он присутствует), иначе берется как для данных стандартного целого типа.

Значение результата: если A>0, NINT (A) имеет значение INT (A + 0.5); если A <= 0, NINT (A) имеет значение INT (A-0.5). Если процессор не может представить результат заданного целого типа, результат не определен.

Пример: NINT(2.783) имеет значение 3.

13.13.77. NOT (I)

Описание: выполняет операцию логического дополнения.

Класс: поэлементная функция.

Аргумент: I должно быть целого типа.

Тип результата и параметр типа: те же, что у I.

Значение результата: результат имеет значение, полученное выполнением операции поразрядного дополнения I в соответствии со следующей таблицей истинности:

I	NOT(I)
1	0
0	1

Модель для интерпретации целого значения как последовательности битов приведена в 13.5.7.

Пример: если I представляется строкой битов 01010101, NOT(I) имеет двоичное значение 10101010.

13.13.78. PACK (ARRAY, MASK, VECTOR)

Необязательный аргумент: VECTOR.

Описание: упаковка массива в одномерный массив под управлением маски.

Класс: функция преобразования.

Аргументы:

ARRAY	может быть любого типа. Он не должен быть скаляром.
MASK	должен быть логическим и согласованным с ARRAY.

VECTOR (необязательный) должен быть одномерным массивом того же типа и с теми же параметрами типа, что ARRAY. Он должен иметь, по меньшей мере, столько элементов, сколько в MASK элементов со значением "истина", а если MASK - скаляр со значением "истина", VECTOR должен иметь, по меньшей мере, столько же элементов, сколько их в ARRAY.

Тип результата, параметр типа и конфигурация: результат - это одномерный массив того же типа с теми же параметрами типа, что и ARRAY. Если аргумент VECTOR присутствует, размер результата равен размеру VECTOR; иначе его размер равен числу *t* элементов со значением "истина" в MASK, если MASK не является скаляром со значением "истина". В последнем случае размер результата совпадает с размером ARRAY.

Значение результата: элемент *i* результата - это элемент ARRAY, который соответствует *i*-му из элементов MASK со значением "истина" при выборе элементов в индексном порядке для *i*=1, 2,...,*t*. Если VECTOR присутствует и он имеет размер *n*>*t*, элемент *i* результата имеет значение VECTOR(*i*) для *i*=*t*+1,...,*n*.

Примеры: ненулевые элементы массива M со значением $\begin{bmatrix} 0 & 0 & 0 \\ 9 & 0 & 0 \\ 0 & 0 & 7 \end{bmatrix}$ могут быть "собраны" функцией PACK. Результат PACK (M, MASK = M.NE.0) есть [9,7], а результат PACK(M, M.NE.0, VECTOR=(/2,4,6,8,10,12/)) равен [9,7,6,8,10,12].

13.13.79. PRECISION (X)

Описание: выдает десятичную точность в модели представления чисел того же типа и с тем же параметром типа, что у аргумента.

Класс: справочная функция.

Аргумент: X должно быть вещественного или комплексного типа, его значение может быть скаляром или массивом.

Тип результата, параметр типа и конфигурация: скаляр стандартного целого типа.

Значение результата: результат имеет значение INT ((*p*-1) * LOG10 (*b*)) + *k*, где *b* и *p* являются такими, как определено в разделе 13.7.1 для модели представления вещественных чисел с тем же параметром вида, что у X, а *k* = 1, если *b* - целая степень 10, или 0 - в противном случае.

Пример: PRECISION (X) имеет значение INT (23 * LOG10 (2.)) = INT (6.92...) = 6 для вещественного X, модель которого приведена в конце раздела 13.7.1.

13.13.80. PRESENT (X)

Описание: определяет, присутствует ли необязательный аргумент.

Класс: справочная функция.

Аргумент: должен быть необязательным аргументом той процедуры, в которой появляется ссылка на функцию PRESENT.

Тип результата и параметры типа: скаляр стандартного логического типа.

Значение результата: результат имеет значение "истина", если A присутствует (12.5.2.8), иначе - "ложь".

13.13.81. PRODUCT (ARRAY, DIM, MASK)

Необязательные аргументы: DIM, MASK.

Описание: произведение всех элементов ARRAY по измерению DIM, соответствующих элементам MASK со значением "истина".

Класс: функция преобразования.

Аргументы:

ARRAY — должен иметь тип целый, вещественный или комплексный. Он не должен быть скаляром.

DIM (необязательный) — должен быть скаляром целого типа со значением в интервале 1 <= DIM <= n, где n - размерность ARRAY. Соответствующий фактический аргумент не может быть необязательным формальным аргументом.

MASK (необязательный) — должен быть логического типа и быть согласованным с ARRAY.

Тип результата, параметр типа и конфигурация: результат имеет тот же тип и параметр типа, что ARRAY; это скаляр, если DIM отсутствует или если ARRAY имеет размерность 1; иначе это массив размерности n-1 и конфигурации $(d_1, d_2, ..., d_{DIM-1}, d_{DIM+1}, ..., d_n)$, где $(d_1, d_2, ..., d_n)$ - конфигурация ARRAY.

Значение результата:

Случай 1: результат PRODUCT (ARRAY) имеет значение, равное процессорно-зависимой аппроксимации произведения всех элементов ARRAY или равен единице, если ARRAY имеет нулевой размер.

Случай 2: результат PRODUCT (ARRAY, MASK=MASK) имеет значение, равное процессорно-зависимой аппроксимации произведения тех элементов ARRAY, которые соответствуют элементам MASK со значением "истина", или равен единице, если таких элементов нет.

Случай 3: если ARRAY имеет размерность 1, PRODUCT (ARRAY, DIM [,MASK]) имеет то же значение, что PRODUCT (ARRAY [,MASK = MASK]). Иначе значение элемента $(s_1, s_2, ..., s_{DIM-1}, s_{DIM+1}, ..., s_n)$ массива PRODUCT (ARRAY, DIM [,MASK]) равно PRODUCT (ARRAY $(s_1, s_2, ..., s_{DIM-1}, :, s_{DIM+1}, ..., s_n)$ [,MASK=MASK$(s_1, s_2, ..., s_{DIM-1}, :, s_{DIM+1}, ..., s_n)$]).

Примеры:

Случай 1: значение PRODUCT ((/1,2,3/)) равно 6.

Случай 2: PRODUCT (C, MASK = C .GT. 0.0) формирует произведение положительных элементов C.

Случай 3: если B - массив $\begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix}$, PRODUCT (B, DIM=1) есть [2,12,30],

PRODUCT (B, DIM=2) есть [15,48].

13.13.82. RADIX (X)

Описание: выдает основание модели, представляющей числа того же типа и с тем же параметром вида, что у аргумента.

Класс: справочная функция.

Аргумент: X должен быть целого или вещественного типа. Его значением может быть скаляр или массив.

Тип результата, параметр типа и конфигурация: скаляр стандартного целого типа.

Значение результата: результат имеет значение r, если X имеет целый тип, и значение b, если X имеет вещественный тип, где r и b определены, как в 13.7.1 для модели, представляющей числа того же типа и с тем же параметром типа, что и X.

Пример: RADIX(X) имеет значение 2 для вещественного X, модель которого представлена в конце 13.7.1.

13.13.83. RANDOM_NUMBER (HARVEST)

Описание: выдает одно псевдослучайное число или массив псевдослучайных чисел из равномерного распределения в диапазоне $0 \le x < 1$.

Класс: подпрограмма.

Аргумент: HARVEST *должен быть вещественного типа. Это аргумент с атрибутом* INTENT (OUT). Его значением может быть скаляр или массив. В него заносятся псевдослучайные числа из равномерного распределения в интервале $0 \le x < 1$.

Примеры:

```
REAL X,Y (10,10)
      ! Присвоить X псевдослучайное число
CALL RANDOM_NUMBER (HARVEST = X)
CALL RANDOM_NUMBER (Y)
      ! X и Y содержат равномерно распределенные случайные числа
```

13.13.84. RANDOM_SEED (SIZE, PUT, GET)

Необязательные аргументы: SIZE, PUT, GET.

Описание: переустанавливает или запрашивает генератор псевдослучайных чисел, используемый в RANDOM_NUMBER.

Класс: подпрограмма.

Аргументы:

SIZE (необязательный) — должен быть скаляром стандартного целого типа. Это аргумент с атрибутом INTENT(OUT). Его значение устанавливается равным N (количеству целых, которые процессор использует для получения массива начальных значений).

PUT (необязательный) — должен быть одномерным массивом стандартного целого типа с размером $\ge$ N. Это аргумент с атрибутом INTENT (IN). Он используется процессором для установки начальных значений для генератора.

GET (необязательный) — должен быть одномерным массивом стандартного целого типа с размером $\ge$ N. Это аргумент с атрибутом INTENT (OUT). Он устанавливается процессором равным текущему значению массива начальных значений для генератора.

Если аргументов нет, процессор устанавливает начальные значения для генератора, зависящие от процессора.

Примеры:

```
CALL RANDOM_SEED                          ! Инициализация процессора
CALL RANDOM_SEED (SIZE=K)                 ! Устанавливает K = N
CALL RANDOM_SEED (PUT=SEED(1:K))          ! Устанавливает начальные зна-
                                          ! чения, заданные пользователем
CALL RANDOM_SEED (GET=OLD(1:K))           ! Считывает текущие начальные
                                          ! значения
```

13.13.85. RANGE (X)

Описание: выдает десятичный диапазон показателя экспоненты в модели, представляющей целые и вещественные числа того же типа и с тем же параметром вида, что у аргумента.

Класс: справочная функция.

Аргумент: X должно быть целого, вещественного или комплексного типа, его значение может быть скаляром или массивом.

Тип результата, параметр типа и конфигурация: скаляр стандартного целого типа.

Значение результата:

Случай 1: для целого аргумента результат имеет значение INT (LOG10 (*HUGE*)), где *HUGE* - наибольшее положительное целое в модели, представляющей целые числа с тем же параметром вида, что у X (13.7.1).

Случай 2: для вещественного или комплексного аргумента результат имеет значение INT (MIN (LOG10 (*HUGE*),-LOG10 (*TINY*))), где *HUGE* и *TINY* являются наибольшим и наименьшим положительными числами в модельном представлении вещественных чисел с тем же параметром вида, что у X (см. 13.7.1).

Пример: RANGE(X) имеет значение 38 для вещественного числа X, модель которого приведена в конце раздела 13.7.1, т.к. в этом случае $huge = (1 - 2^{-24}) \cdot 2^{127}$ и $tiny = 2^{-127}$.

13.13.86. REAL (A, KIND)

Необязательный аргумент: KIND.

Описание: преобразование к вещественному типу.

Класс: поэлементная функция.

Аргументы:

A	должен быть целого, вещественного или комплексного типа.
KIND (необязательный)	должно быть скалярным целым выражением инициализации.

Тип результата и параметр типа: вещественный.

Случай 1: если A - целого или вещественного типа и KIND присутствует, параметр вида задается с помощью KIND. Если A - целое или вещественное и KIND отсутствует, параметром вида является процессорно-зависимый параметр вида для стандартного вещественного типа.

Случай 2: если A - комплексного типа и KIND присутствует, значение параметра вида задается с помощью KIND, если же KIND отсутствует, параметр вида такой же, как у A.

Значение результата:

Случай 1: если A имеет тип целый или вещественный, результат равен процессорно-зависимой аппроксимации A.

Случай 2: если A имеет комплексный тип, результат равен процессорно-зависимой аппроксимации действительной части A.

Примеры: REAL(-3) имеет значение -3.0. REAL (Z) имеет тот же параметр вида и то же значение, что действительная часть комплексной переменной Z.

13.13.87. REPEAT (STRING, NCOPIES)

Описание: конкатенация нескольких копий строки.

Класс: функция преобразования.

Аргументы:

STRING	должен быть скаляром символьного типа.
NCOPIES	должен быть скаляром целого типа. Его значение не должно быть отрицательным.

Тип результата, параметр типа и конфигурация: символьный с длиной, равной длине STRING, умноженной на значение NCOPIES. Параметр вида такой же, как у STRING.

Значение результата: значением результата является конкатенация NCOPIES копий STRING.

Пример: REPEAT('H',2) имеет значение 'HH'. REPEAT ('XYZ',0) -имеет значением строку нулевой длины.

13.13.88. RESHAPE (SOURCE, SHAPE, PAD, ORDER)

Необязательные аргументы: PAD, ORDER.

Описание: конструирование массива заданной конфигурации из элементов данного массива.

Класс: функция преобразования.

Аргументы:

SOURCE	может быть любого типа. Его значением должен быть массив. Если PAD отсутствует или нулевого размера, размер SOURCE должен быть больше или равен PRODUCT (SHAPE). Размер результата есть произведение значений элементов SHAPE.
SHAPE	должен быть целым, одномерным и константного размера. Размер должен быть положительным и меньше 8. Он не должен иметь отрицательных элементов.
PAD (необязательный)	должен быть того же типа и с теми же параметрами, что SOURCE. Он должен иметь значение-массив.
ORDER(необязательный)	должен быть целым той же конфигурации, что и SHAPE, а его значение должно быть перестановкой элементов (1, 2,...,*N*), где *N* - размер SHAPE. Если он опущен, считается, что он присутствует со значением (1,2,...,*N*).

Тип результата, параметр типа и конфигурация: результат - это массив конфигурации SHAPE (т.е. SHAPE (RESHAPE (SOURCE, SHAPE, PAD, ORDER)) равно SHAPE), тип и параметры типа для которого те же, что у SOURCE.

Значение результата: элементы результата, взятого в переставленном порядке индексов ORDER(1),...,ORDER(*N*), такие же, что у SOURCE при нормальном порядке, дополненные, если необходимо, элементами PAD в принятом для массива порядке, за которыми, если необходимо, следуют дополнительные копии PAD в обычном порядке.

Примеры: RESHAPE ((/1,2,3,4,5,6/), (/2,3/)) имеет значение - $\begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix}$,

а RESHAPE((/1,2,3,4,5,6/), (/2,4/), (/0,0/), (/2,1/)) имеет значение $\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 0 & 0 \end{bmatrix}$.

13.13.89. RRSPACING (X)

Описание: выдает обратную величину от относительной окрестности модельных чисел около значения аргумента.

Класс: поэлементная функция.

Аргумент: X должен быть вещественным.

Тип результата и параметр типа: те же, что у X.

Значение результата: результат имеет значение $X*b^{-\varepsilon}*b^{p}$, где b, ε и p такие, как определено в 13.7.1 для модельного представления X.

Пример: RRSPACING (-3.0) имеет значение $0.75*2^{24}$ для вещественных чисел, модель которого представлена в конце 13.7.1.

13.13.90. SCALE (X, I)

Описание: выдает $X*b^I$, где b - основание в модельном представлении X.

Класс: поэлементная функция.

Аргументы:

X должен быть вещественным.

I должен быть целым.

Тип результата и параметр типа: те же, что у X.

Значение результата: результат имеет значение $X*b^I$, где b определено в 13.7.1 для модельных чисел, представляющих значения X, при условии, что этот результат находится внутри диапазона, иначе результат зависит от процессора.

Пример: SCALE (3.0, 2) имеет значение 12.0 для вещественных чисел, модель которых представлена в конце 13.7.1.

13.13.91. SCAN (STRING, SET, BACK)

Необязательный аргумент: BACK.

Описание: просмотр строки для нахождения какого-либо символа в некотором наборе символов.

Класс: поэлементная функция.

Аргументы:

STRING должен быть символьного типа.

SET должен быть символьного типа.

BACK (необязательный) должен быть логического типа.

Тип результата и параметр типа: стандартный целый тип.

Значение результата:

Случай 1: если BACK отсутствует либо присутствует со значением "ложь" и если STRING содержит хотя бы один символ из SET, значением результата является позиция самого левого символа STRING, который находится и в SET.

Случай 2: если BACK присутствует со значением "истина" и если STRING содержит хотя бы один символ, найденный в SET, значением результата является позиция самого правого из символов STRING, принадлежащих SET.

Случай 3: результатом является нуль, если STRING не содержит никаких символов, которые имеются в SET, или если длина либо STRING, либо SET равна нулю.

Примеры:

Случай 1: SCAN('FORTRAN','TR') имеет значение 3.

Случай 2: SCAN('FORTRAN','TR',BACK=.TRUE.) имеет значение 5.

Случай 3: SCAN('FORTRAN','BCD') имеет значение 0.

13.13.92. SELECTED_INT_KIND (R)

Описание: выдает значение параметра вида для целого типа данных, который представляет все целые значения n в интервале
$-10^R < n < 10^R$.

Класс: функция преобразования.

Аргумент: R должно быть скаляром целого типа.

Тип результата, параметр типа и конфигурация: скаляр стандартного целого типа.

Значение результата: результат имеет значение, равное значению параметра вида для целого типа данных, представляющего все значения n в диапазоне $-10^R < n < 10^R$; если же такой параметр вида в процессоре отсутствует, результат равен -1. Если этому критерию отвечает более чем один параметр вида, выдается значение того из них, у которого наименьшее значение диапазона десятичного порядка, если же и таких значений несколько, выдается наименьшее из них.

Пример: SELECTED_INT_KIND(6) имеет значение KIND(0) на машине, которая поддерживает способ представления данных стандартного целого типа с $R=2$ и $Q=31$.

13.13.93. SELECTED_REAL_KIND (P, R)

Необязательные аргументы: P, R.

Описание: выдает значение параметра вида для вещественного типа данных с десятичной точностью, как минимум, P цифр и диапазоном десятичного порядка, как минимум, R.

Класс: функция преобразования.

Аргументы: должен присутствовать хотя бы один аргумент.

P (необязательный) должен быть скаляром целого типа.

R (необязательный) должен быть скаляром целого типа.

Тип результата, параметр типа и конфигурация: скаляр стандартного целого типа.

Значение результата: результат имеет значение, равное значению параметра вида для вещественного типа данных с десятичной точностью, выдаваемой функцией PRECISION и равной, как минимум, P цифрам и с диапазоном десятичного порядка, выдаваемым функцией RANGE и равным, как минимум, R; если такого параметра вида нет в процессоре, результат равен -1, когда недоступна точность, -2, когда недоступен диапазон порядка, и -3, когда недоступно ни то, ни другое. Если более чем одно значение параметра вида удовлетворяет заданным критериям, выдается значение того из них, который имеет наименьшую десятичную точность, а если и таких значений несколько, выдается наименьшее из них.

Пример: SELECTED_REAL_KIND (6, 70) имеет значение KIND(0.0) на машине, которая поддерживает способ аппроксимации данных стандартного вещественного типа с $b=16$, $p=6$, $\varepsilon_{MIN}=-64$ и $\varepsilon_{MAX}=63$.

13.13.94. SET_EXPONENT (X, I)

Описание: выдает модельное число, дробная часть которого является дробной частью модельного представления X и экспонентная часть которого равна I.

Класс: поэлементная функция.

Аргументы:

X должен быть вещественного типа.

I должен быть целого типа.

Тип результата и параметр типа: те же, что у X.

Значение результата: результат имеет значение $X*b^{I-\varepsilon}$, где b и ε определены в 13.7.1 для модельного представления X, при условии, что результат находится в диапазоне, в противном случае результат зависит от процессора. Если X имеет нулевое значение, результат имеет значение нуль.

Пример: SET_EXPONENT (3.0, 1) имеет значение 1.5 для вещественных чисел, модель которых описана в конце 13.7.1.

13.13.95. SHAPE (SOURCE)

Описание: выдает конфигурацию массива или скаляра.

Класс: справочная функция.

Аргумент: SOURCE может быть любого типа. Его значением может быть скаляр или массив. Он не должен быть неразмещенным динамически размещаемым массивом или не связанным указателем. Он не должен быть массивом неопределенного размера.

Тип результата, параметр типа и конфигурация: результат является массивом стандартного целого типа и размерности один, размер которого равен размерности SOURCE.

Значение результата: значение результата - это конфигурация SOURCE.

Примеры: значение SHAPE (A (2:5, -1:1)) есть [4,3]. Значением SHAPE(3) является одномерный массив нулевого размера.

13.13.96. SIGN (A, B)

Описание: абсолютное значение A со знаком B.

Класс: поэлементная функция.

Аргументы:

A — должен быть целого или вещественного типа.

B — должен быть того же типа и с тем же параметром вида, что и A.

Тип результата и параметр типа: те же, что у A.

Значение результата: значение результата равно |A|, если B >= 0, и -|A|, если B < 0.

Пример: SIGN (-3.0, 2.0) имеет значение 3.0.

13.13.97. SIN (X)

Описание: функция синус.

Класс: поэлементная функция.

Аргумент: X должен быть вещественного или комплексного типа.

Тип результата и параметр типа: те же, что у X.

Значение результата: результат имеет значение, равное процессорно-зависимой аппроксимации sin(X). Если X имеет тип вещественный, он рассматривается как значение в радианах. Если X имеет комплексный тип, его действительная часть рассматривается как значение в радианах.

Пример: SIN(1.0) имеет значение 0.84147098 (приближенно).

13.13.98. SINH (X)

Описание: гиперболический синус.

Класс: поэлементная функция.

Аргумент: X должен быть вещественного типа.

Тип результата и параметр типа: те же, что у X.

Значение результата: результат имеет значение, равное процессорно-зависимой аппроксимации sinh(X).

Пример: SINH(1.0) имеет значение 1.1752012 (приближенно).

13.13.99. SIZE (ARRAY, DIM)

Необязательный аргумент: DIM.

Описание: выдает протяженность массива по заданному измерению или общее число элементов массива.

Класс: справочная функция.

Аргументы:

ARRAY — может быть любого типа. Он не может быть скаляром, неразмещенным динамически размещаемым массивом, не связанным указателем. Если ARRAY - массив неопределенного размера, DIM должен присутствовать со значением, меньшим размерности ARRAY.

DIM (необязательный) — должен быть скаляром целого типа со значением в интервале 1 <= DIM <= *N*, где *N* - размерность ARRAY.

Тип результата, параметр типа и конфигурация: стандартный целый скаляр.

Значение результата: результат имеет значение, равное протяженности измерения DIM в массиве ARRAY или, если DIM отсутствует, результат равен общему числу элементов ARRAY.

Примеры: значение SIZE (A (2:5, -1:1), DIM=2) есть 3. Значение SIZE (A (2:5, -1:1)) равно 12.

13.13.100. SPACING (X)

Описание: выдает значение абсолютной окрестности модельного числа от значения аргумента.

Класс: поэлементная функция.

Аргумент: X должен быть вещественного типа.

Тип результата и параметр типа: те же, что у X.

Значение результата: результат имеет значение $b^{\varepsilon-p}$, где b, ε и p определены в 13.7.1 для модельного представления X, при условии, что результат находится внутри интервала. В противном случае результат совпадает с TINY (X).

Пример: SPACING (3.0) имеет значение 2^{-22} для вещественных, модель которых описана в конце 13.7.1.

13.13.101. SPREAD (SOURCE, DIM, NCOPIES)

Описание: дублирует массив добавлением измерения. Прибавляет несколько копий SOURCE по заданному измерению (как в случае формирования книги из копий одной страницы) и, таким образом, формирует массив с размерностью на единицу большей.

Класс: функция преобразования.

Аргументы:

SOURCE — может быть любого типа. Его значением может быть скаляр или массив. Размерность SOURCE должна быть меньше 7.

DIM — должен быть скаляром целого типа со значением в диапазоне 1 <= DIM <= *N*+1, где *N* - размерность SOURCE.

NCOPIES — должен быть скаляром целого типа.

Тип результата, параметр типа и конфигурация: результатом является массив того же типа с теми же параметрами типа, что и SOURCE, и размерности *N*+1, где *N* - размерность SOURCE.

Случай 1: если SOURCE - скаляр, конфигурация результата - (MAX (NCOPIES,0)).

Случай 2: если SOURCE имеет значение-массив и его конфигурация - $(d_1, d_2, \ldots, d_N)$, конфигурация результата есть $(d_1, d_2, \ldots, d_{DIM-1}, MAX (NCOPIES, 0), d_{DIM}, \ldots, d_N)$.

Значение результата:

Случай 1: если SOURCE - скаляр, каждый элемент результата имеет значение, равное SOURCE.

Случай 2: если SOURCE имеет значение-массив, элемент результата с индексами $(R_1, R_2, ..., R_{N+1})$ имеет значение SOURCE $(R_1, R_2, ..., R_{DIM-1}, R_{DIM+1}, ..., R_{N+1})$.

Пример: если A - массив [2, 3, 4], SPREAD (A,DIM=1, NCOPIES=NC)- это массив $\begin{bmatrix} 2 & 3 & 4 \\ 2 & 3 & 4 \\ 2 & 3 & 4 \end{bmatrix}$,

если NC имеет значение 3, либо массив нулевого размера, если NC имеет значение 0.

13.13.102. SQRT (X)

Описание: корень квадратный.

Класс: поэлементная функция.

Аргумент: X должен быть вещественного или комплексного типа. Если X не комплексный, его значение должно быть больше или равно нулю.

Тип результата и параметр типа: те же, что у X.

Значение результата: результат имеет значение, равное процессорно-зависимой аппроксимации квадратного корня из X. Результат для комплексного типа - главное значение с действительной частью, большей или равной нулю. Когда действительная часть результата равна нулю, мнимая часть больше или равна нулю.

Пример: SQRT(4.0) имеет значение 2.0 (приближенно).

13.13.103. SUM (ARRAY, DIM, MASK)

Необязательные аргументы: DIM, MASK.

Описание: сумма всех элементов ARRAY по измерению DIM, соответствующих элементам MASK со значением "истина".

Класс: функция преобразования.

Аргументы:

ARRAY	должен быть целого, вещественного или комплексного типа. Он не должен быть скаляром.
DIM (необязательный)	должен быть скаляром целого типа со значением в диапазоне 1 <= DIM <= N, где N - размерность ARRAY. Соответствующий фактический аргумент не должен быть необязательным формальным аргументом.
MASK(необязательный)	должен быть логического типа и должен быть согласован с ARRAY.

Тип результата, параметр типа и конфигурация: результат имеет тот же тип и тот же параметр вида, что ARRAY; он является скаляром, если DIM отсутствует или ARRAY имеет размерность, равную единице; иначе он является массивом размерности N-1 и конфигурации $(d_1, d_2, ..., d_{DIM-1}, d_{DIM+1}, ..., d_N)$, где $(d_1, d_2, ..., d_N)$ - конфигурация ARRAY.

Значение результата:

Случай 1: результат SUM(ARRAY) имеет значение, равное процессорно-зависимой аппроксимации суммы всех элементов ARRAY или равен нулю, если ARRAY имеет нулевой размер.

Случай 2: результат SUM (ARRAY, MASK=MASK) имеет значение, равное процессорно-зависимой аппроксимации суммы элементов ARRAY, соответствующих элементам MASK со значением "истина", или равен нулю, если таковых нет.

Случай 3: если ARRAY имеет размерность 1, SUM (ARRAY, DIM [,MASK]) имеет значение, равное значению SUM (ARRAY [,MASK = MASK]). Иначе значение элемента (s_1, s_2,..., s_{DIM-1}, s_{DIM+1}, ..., s_N) массива SUM (ARRAY, DIM [,MASK]) равно SUM (ARRAY(s_1, s_2, ..., s_{DIM-1}, :, s_{DIM+1}, ...,s_N) [,MASK = MASK (s_1, s_2, ..., s_{DIM-1}, :, s_{DIM+1}, ..., s_N)]).

Примеры:

Случай 1: значение SUM ((/1, 2, 3/)) равно 6.

Случай 2: SUM (C, MASK = C .GT. 0.0) формирует арифметическую сумму положительных элементов C.

Случай 3: если B- массив $\begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix}$,

SUM (B, DIM = 1) есть [3,7,11], а SUM (B, DIM=2) - это [9, 12].

13.13.104. SYSTEM_CLOCK (COUNT, COUNT_RATE, COUNT_MAX)

Необязательные аргументы: COUNT, COUNT_RATE, COUNT_MAX.

Описание: выдает дату из часов реального времени в виде целого числа.

Класс: подпрограмма.

Аргументы:

COUNT (необязательный) должен быть скаляром стандартного целого типа. Это аргумент с атрибутом INTENT (OUT). Он устанавливается равным процессорно-зависимому значению, основанному на текущем значении процессорных часов, или равен значению -HUGE(0), если часов нет. Процессорно-зависимое значение увеличивается на единицу для каждого такта часов, пока не достигнет значения COUNT_MAX, и устанавливается в нуль для следующего такта. Оно лежит в интервале от 0 до COUNT_MAX, если часы есть.

COUNT_RATE (необязательный)

должен быть скаляром стандартного целого типа. Это аргумент с атрибутом INTENT (OUT). Он устанавливается равным количеству тактов процессорного времени в секунду, или нулю, если часов нет.

COUNT_MAX (необязательный)

должен быть скаляром стандартного целого типа. Он имеет атрибут INTENT(OUT). Он устанавливается равным максимальному значению, которого может достигнуть COUNT, или нулю, если нет часов.

Пример: если процессорные часы - это 24-часовые часы, которые регистрируют время с секундными интервалами, в момент времени 11.30 ссылка

```
CALL SYSTEM_CLOCK (COUNT=C, COUNT_RATE = R, COUNT_MAX=M)
```

установит C=11*3600+30*60 = 41400, R=1 и M = 24*3600-1 = 86399.

13.13.105. TAN (X)

Описание: функция тангенса.

Класс: поэлементная функция.

Аргумент: X должен быть вещественным.

Тип результата и параметр типа: те же, что у X.

Значение результата: результат имеет значение, равное процессорно-зависимой аппроксимации TAN(X), где X рассматривается как значение в радианах.

Пример: TAN(1.0) имеет значение 1.5574077 (приближенно).

13.13.106. TANH (X)

Описание: функция гиперболического тангенса.

Класс: поэлементная функция.

Аргумент: X должен быть вещественным.

Тип результата и параметр типа: те же, что у X.

Значение результата: результат имеет значение, равное процессорно-зависимой аппроксимации TANH(X).

Пример: TANH(1.0) имеет значение 0.76159416 (приближенно).

13.13.107. TINY (X)

Описание: выдает наименьшее положительное число в модели, представляющей числа того же типа и с тем же параметром вида, что у аргумента.

Класс: справочная функция.

Аргумент: X должен быть вещественным. Его значением может быть скаляр или массив.

Тип результата, параметр типа и конфигурация: скаляр того же типа и с тем же параметром вида, что у X.

Значение результата: результат имеет значение $b**\epsilon^{MIN-1}$, где b и ϵ определены в 13.7.1. для модели, представляющей числа того же типа и с тем же параметром вида, что и X.

Пример: TINY(X) имеет значение 2^{-127} для вещественного X, модель для которого приведена в конце 13.7.1.

13.13.108. TRANSFER (SOURCE, MOLD, SIZE)

Необязательный аргумент: SIZE.

Описание: выдает результат с физическим представлением, идентичным физическому представлению SOURCE, но интерпретируемым с типом и параметрами типа MOLD.

Класс: функция преобразования.

Аргументы:

SOURCE	может иметь значением массив или скаляр любого типа.
MOLD	может иметь значением массив или скаляр любого типа.
SIZE (необязательный)	должен быть скаляром целого типа. Соответствующий фактический аргумент не должен быть необязательным формальным аргументом.

Тип результата, параметр типа и конфигурация: тип и параметры типа результата совпадают с MOLD.

Случай 1: если MOLD - скаляр и SIZE отсутствует, результат является скаляром.

Случай 2: если значением MOLD является массив и SIZE отсутствует, значением результата является одномерный массив. Его размер - минимально возможный для того, чтобы его физическое представление было не короче представления SOURCE.

Случай 3: если присутствует SIZE, значением результата является одномерный массив размера SIZE.

Значение результата: если физическое представление результата имеет ту же длину, что и SOURCE, физическим представлением результата является представление SOURCE. Если физическое представление результата длиннее, чем для SOURCE, физическое представление начальной части результата совпадает с SOURCE, а остальная часть не определена. Если физическое представление результата короче, чем у SOURCE, физическим представлением результата является начальная часть SOURCE. Если D и E - скалярные переменные и физическое представление D той же или большей длины, чем у E, значением

TRANSFER (TRANSFER (E,D), E) должно быть значение E. Если D -массив, а E - одномерный массив, значением TRANSFER (TRANSFER (E,D), E, SIZE(E)) должно быть значение E.

Примеры:

Случай 1: TRANSFER (1082130432, 0.0) имеет значение 4.0 в процессоре, который представляет значения 4.0 и 1082130432 как строку двоичных цифр 0100 0000 1000 0000 0000 0000 0000 0000.

Случай 2: TRANSFER ((/1.1, 2.2, 3.3/), (/(0.0, 0.0)/)) - это комплексный одномерный массив длины 2, первый элемент которого имеет значение (1.1,2.2), а второй элемент имеет действительную часть со значением 3.3. Мнимая часть второго элемента не определена.

Случай 3: TRANSFER((/1.1, 2.2, 3.3/), (/(0.0, 0.0)/), 1) имеет значение (1.1 + 2.2*i*).

13.13.109. TRANSPOSE (MATRIX)

Описание: транспонирование массива размерности 2.

Класс: функция преобразования.

Аргумент: MATRIX может быть любого типа и должен иметь размерность два. Его конфигурация должна быть определена.

Тип результата, параметр типа и конфигурация: результат - это массив того типа и с теми же параметрами типа, что у MATRIX, размерности два и конфигурации (*n*,*m*), где (*m*,*n*) - конфигурация MATRIX.

Значение результата: элемент (*i*,*j*) результата имеет значение MATRIX(*i*,*j*), *i*=1,2,...,*n*; *j*=1,2,...,*m*.

Пример: если A - массив $\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$, то TRANSPOSE (A) равно $\begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{bmatrix}$.

13.13.110. TRIM (STRING)

Описание: выдает результат с удаленными конечными пробелами.

Класс: функция преобразования.

Аргумент: STRING должен быть скаляром символьного типа.

Тип результата и параметры типа: символьный с тем же параметром вида, что у STRING, а длина меньше длины STRING на число конечных пробелов STRING.

Значение результата: значением результата является значение STRING без конечных пробелов. Если STRING не содержит символов, отличных от пробела, результат имеет нулевую длину.

Пример: TRIM(' A B ') имеет значение ' A B'.

13.13.111. UBOUND (ARRAY, DIM)

Необязательный аргумент: DIM.

Описание: выдает все верхние границы массива или указанную верхнюю границу.

Класс: справочная функция.

Аргументы:

ARRAY — может быть любого типа. Он не должен быть скаляром, неразмещенным динамически размещаемым массивом или не связанным указателем. Если ARRAY - массив неопределенного размера, DIM должен присутствовать со значением, меньшим размерности ARRAY.

DIM (необязательный) — должен быть скаляром целого типа со значением в интервале 1 <= DIM <= *N*, где *N* - размерность ARRAY. Соответствующий фактический аргумент не должен быть необязательным формальным аргументом.

Тип результата, параметр типа и конфигурация: результат имеет стандартный целый тип; это скаляр, если присутствует DIM, а иначе - одномерный массив размера *N*, где *N* - размерность ARRAY.

Значение результата:

Случай 1: для секции массива или выражения-массива, не являющегося ни полным массивом, ни компонентом структуры, UBOUND (ARRAY, DIM) имеет значение, равное числу элементов в данном измерении, в противном случае это значение равно верхней границе индекса DIM в ARRAY, если измерение DIM не имеет нулевого размера, и значение 0, если измерение DIM имеет размер 0.

Случай 2: UBOUND (ARRAY) имеет значение, *i*-й компонент которого равен UBOUND (ARRAY,*i*) для *i*=1,2,...,*N*, где *N* - размерность ARRAY.

Примеры: если A объявлен в операторе

```
REAL A (2:3, 7:10)
```

то UBOUND (A) есть [3,10], а UBOUND (A, DIM=2) есть 10.

13.13.112. UNPACK (VECTOR, MASK, FIELD)

Описание: распаковка одномерного массива в массив под управлением маски.

Класс: функция преобразования.

Аргументы:

VECTOR — может быть любого типа. Он должен быть размерности один. Его размер должен быть, по крайней мере, *t*, где *t* - количество элементов в MASK со значением "истина".

MASK — должен иметь значением логический массив.

FIELD — должен быть того же типа и с теми же параметрами типа, что VECTOR, и должен согласовываться с MASK.

Тип результата, параметр типа и конфигурация: результатом является массив того же типа и с теми же параметрами типа, что у VECTOR, и той же конфигурации, что MASK.

Значение результата: элемент результата, который соответствует *i*-му элементу MASK со значением "истина" (в индексном порядке), имеет значение VECTOR(*i*) для *i*=1,2,...,*t*, где *t* - количество элементов со значением "истина" в MASK. Каждый из остальных элементов имеет значение, равное FIELD, если FIELD - скаляр, или соответствующего элемента FIELD, если FIELD - массив.

Примеры: конкретные значения можно "рассеять" по конкретным позициям в массиве с использованием UNPACK. Если M - массив

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \text{ V - массив [1,2,3], а Q - логическая маска } \begin{bmatrix} \bullet & T & \bullet \\ T & \bullet & \bullet \\ \bullet & \bullet & T \end{bmatrix},$$

где T представляет значение "истина", а • - "ложь",

то результат UNPACK (V, MASK = Q, FIELD = M) имеет значение $\begin{bmatrix} 1 & 2 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 3 \end{bmatrix}$.

а резуль-тат UNPACK (V, MASK = Q, FIELD = 0) имеет значение $\begin{bmatrix} 0 & 2 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 3 \end{bmatrix}$.

13.13.113. VERIFY (STRING, SET, BACK)

Необязательный аргумент: BACK.

Описание: проверяет, все ли символы в строке входят во множество символов, идентифицируя при этом позицию первого из символов строки, не попадающего в данное множество.

Класс: поэлементная функция.

Аргументы:

STRING	должен быть символьного типа.
SET	должен быть символьным с тем же параметром вида, что у STRING.
BACK (необязательный)	должен быть логического типа.

Тип результата и параметр типа: стандартный целый.

Значение результата:

Случай 1: если BACK отсутствует или присутствует со значением "ложь" и если STRING содержит хотя бы один символ не из SET, значением результата является позиция самого левого из символов STRING, не входящих в SET.

Случай 2: если BACK присутствует со значением "истина" и если STRING содержит хотя бы один символ не из SET, значением результата является позиция самого правого из символов STRING, не входящих в SET.

Случай 3: значение результата равно нулю, если каждый символ в STRING входит в SET или если STRING имеет нулевую длину.

Примеры:

Случай 1: VERIFY ('ABBA','A') имеет значение 2.

Случай 2: VERIFY ('ABBA','A',BACK=.TRUE.) имеет значение 3.

Случай 3: VERIFY ('ABBA','AB') имеет значение 0.

Глава 14. ОБЛАСТЬ ДЕЙСТВИЯ, СВЯЗЬ И ОПРЕДЕЛЕННОСТЬ

Объекты идентифицируются лексическими элементами в рамках некоторой **области действия**, такой, как выполняемая программа, контекстная область, отдельный оператор или часть оператора. Если областью действия является выполняемая программа, объект называется **глобальным объектом**. Если областью действия является контекстная область (2.2), объект называется **локальным объектом**. Если областью действия является отдельный оператор или часть оператора, объект называется **операторным объектом**.

Объект можно идентифицировать при помощи:

1) имени (14.1);
2) метки (14.2);
3) номера внешнего устройства ввода/вывода (14.3);
4) символа операции (14.4);
5) символа присваивания (14.5).

Посредством связывания можно ссылаться на объект через один и тот же идентификатор или через разные идентификаторы в разных контекстных областях, либо через разные идентификаторы в одной контекстной области.

14.1. Область действия имен

Именованные объекты бывают глобальными, локальными и операторными.

14.1.1. Глобальные объекты

Программные единицы, общие блоки и внешние процедуры являются в выполняемой программе глобальными объектами. Имя, идентифицирующее глобальный объект, нельзя использовать для идентификации другого глобального объекта в той же выполняемой программе.

14.1.2. Локальные объекты

Внутри контекстной области локальными для нее являются объекты следующих классов:

1) именованные переменные, которые не являются операторными объектами (14.1.3), именованные константы, именованные конструкции, операторные функции, внутренние процедуры, модульные процедуры, формальные процедуры, встроенные процедуры, обобщенные идентификаторы, производные типы и группы NAMELIST;
2) компоненты типов, по отдельному классу на каждый компонент;
3) ключевые слова аргументов, по отдельному классу на каждую процедуру с явным интерфейсом.

Имя, идентифицирующее в контекстной области глобальный объект, кроме имени общего блока (14.1.2.1) или имени внешней функции (14.1.2.2), нельзя использовать для идентификации в этой контекстной области локальных объектов класса (1).

Имя, идентифицирующее в контекстной области локальный объект одного класса, нельзя использовать для идентификации другого локального объекта того же класса, кроме случая обобщенных имен (12.3.2.1). Имя, идентифицирующее локальный объект одного класса, может быть использовано для идентификации локального объекта другого класса.

Заметим, что встроенная процедура недоступна в контекстой области, содержащей одноименный локальный объект того же класса. Например, во фрагменте программы

```
SUBROUTINE SUB
    ...
    A = SIN(K)
    ...
CONTAINS
    FUNCTION SIN(X)
        ...
    END FUNCTION SIN
END SUBROUTINE SUB
```

любая ссылка на функцию SIN в подпрограмме SUB указывает на внутреннюю фунцию SIN, а не на одноименную встроенную функцию.

Имя локального объекта идентифицирует данный объект в данной контекстной области и может быть использовано для идентификации любого локального объекта или глобального объекта в другой контекстной области.

14.1.2.1. Общие блоки

Имя общего блока в контекстной области может быть именем любого локального объекта, кроме именованной константы, встроенной процедуры или локальной переменной, которая является также внешней функцией в процедуре-функции. Если имя используется как для общего блока, так и для локального объекта, вхождение этого имени в любой контекст, кроме имени общего блока в операторах COMMON или SAVE, идентифицирует только локальный объект. Заметим, что имя встроенной функции может быть именем общего блока в контекстной области, которая не ссылается на эту встроенную функцию.

14.1.2.2. Результаты функций

Для каждого оператора FUNCTION или ENTRY в процедуре-функции есть результирующая переменная. Если нет спецификации RESULT, результирующая переменная имеет то же имя, что сама функция, иначе результирующая переменная имеет имя, заданное в спецификации RESULT.

14.1.2.3. Однозначные обобщенные ссылки на процедуры

Данный подраздел содержит правила, которым должна удовлетворять каждая пара специфических процедур, которые либо имеют одинаковое обобщенное имя и одинаковый обобщенный знак операции, либо обе определяют присваивание. Они обеспечивают однозначность обобщенной ссылки на процедуру. Когда расширяется встроенная процедура, знак операции или присваивание, правила применяются так, как если бы встроенный объект был набором специфических процедур, по одному для каждой допустимой комбинации типа, параметра вида и размерности для каждого аргумента или операнда. Когда обобщенная процедура берется из модуля, правила применяются ко всем специфическим версиям, даже если некоторые из них не доступны по своим специфическим именам.

Если две процедуры в контекстной области определяют одну и ту же обобщенную операцию с одинаковым числом операндов либо обе определяют присваивание, одна из них должна иметь формальный аргумент, отличающийся типом, параметрами типа или размерностью от формального аргумента другой процедуры, находящегося в той же позиции в списке аргументов.

Две процедуры, имеющие в контекстной области одинаковое обобщенное имя, должны обе быть либо подпрограммами, либо функциями и хотя бы одна из них должна иметь обязательный формальный аргумент, который

1) соответствует по позиции в списке аргументов формальному аргументу, либо отсутствующему в другой процедуре, либо присутствующему с другим типом или с другим параметром вида, либо с другой размерностью;

2) соответствует по ключевому слову аргумента формальному аргументу либо отсутствующему в другой процедуре, либо присутствующему с другим типом, с другим параметром вида, либо с другой размерностью.

Например, процедуры с телами интерфейса, заданными блоком интерфейса:

```
INTERFACE A
    SUBROUTINE AR (X)
        REAL X
    END SUBROUTINE AR

    SUBROUTINE AI (J)
        INTEGER J
    END SUBROUTINE AI
END INTERFACE
```

удовлетворяют правилам 1 и 2. Однако, если бы J было объявлено как REAL, правило 1 не выполнялось бы, тогда как правило 2 выполнялось бы по-прежнему. В этом случае ссылка на A в операторе

```
CALL A(0.0)
```

была бы неоднозначной.

14.1.2.4. Разрешение ссылок на процедуры

Правила для интерпретации ссылки на процедуру зависят от того, можно ли по доступным описаниям и спецификациям определить, является ли имя процедуры, указанное в ссылке, обобщенным или специфическим в той контекстной области, в которой находится ссылка, либо этого определить нельзя.

1. Имя процедуры определяется в контекстной области как обобщенное, если
   а) данная контекстная область содержит блок интерфейса с таким именем;
   б) данная контекстная область содержит спецификацию атрибута INTRINSIC для этого имени, и это имя является именем обобщенной встроенной процедуры;
   в) если контекстная область содержит оператор USE, который делает доступным это имя процедуры, а соответствующее имя в модуле определено как обобщенное;
   г) если данная контекстная область не содержит объявлений этого имени, но находится внутри объемлющей контекстной области, в которой это имя определяется как обобщенное.
2. Имя процедуры определяется как только специфическое в данной контекстной области, если оно определяется как специфическое и не определяется как обобщенное в этой контекстной области. Имя определяется как специфическое, если:
   а) данная контекстная область содержит тело интерфейса с этим именем;
   б) данная контекстная область содержит модульную или внутреннюю процедуру либо операторную функцию с этим именем;
   в) данная контекстная область содержит спецификацию атрибута INTRINSIC для этого имени, и это имя является именем специфической встроенной процедуры;
   г) данная контекстная область содержит спецификацию атрибута EXTERNAL для этого имени;
   д) если данная контекстная область содержит оператор USE, который делает доступным это имя, а соответствующее имя в модуле определяется как специфическое;

e) если данная контекстная область не содержит объявлений этого имени, но находится внутри объемлющей контекстной области, в которой это имя определяется как специфическое.

3. Процедура не определяется в данной контекстной области, если ее нельзя определить ни как обобщенную, ни как специфическую.

14.1.2.4.1. Разрешение процедурных ссылок для имен, определенных как обобщенные

1. Если ссылка согласуется с одним из специфических интерфейсов блока интерфейса, который имеет это имя и либо сам расположен в контекстной области, содержащей ссылку, либо в контекстной области находится оператор USE, делающий интерфейс доступным, тогда эта ссылка является ссылкой на специфическую процедуру в блоке интерфейса, обеспечивающем данный интерфейс. Заметим, что правила в 14.1.2.3 подразумевают, что может быть не более одной такой специфической процедуры.
2. Если не выполняется правило (1), если контекстная область содержит либо спецификацию атрибута INTRINSIC для этого имени, либо оператор USE, который делает его доступным из модуля, где соответствующее имя описано с атрибутом INTRINSIC, и если ссылка не противоречит интерфейсу этой встроенной процедуры, данная ссылка является ссылкой на эту встроенную процедуру. Заметим, что в случае оператора USE, благодаря возможности переименования, имя в ссылке может отличаться от имени встроенной процедуры.
3. Если не выполняются правила (1) и (2), если контекстная область находится в объемлющей контекстной области, а в ней это имя определяется как обобщенное, и если нет противоречия между данной и объемлющей контекстной областями в том смысле, является ли имя именем функции или именем подпрограммы, ссылка разрешается применением правил данной главы к объемлющей контекстной области.
4. Если не выполняются правила (1), (2) и (3), имя процедуры должно быть именем обобщенной встроенной процедуры, ссылка должна согласовываться с интерфейсом этой встроенной процедуры и тогда эта ссылка является ссылкой на эту встроенную процедуру.

14.1.2.4.2. Разрешение процедурных ссылок для имен, определенных только как специфические

1. Если контекстная область содержит тело интерфейса или спецификацию атрибута EXTERNAL для этого имени, если контекстная область является процедурой и если имя является именем формального аргумента этой процедуры, формальный аргумент является формальной процедурой, а ссылка - ссылкой на формальную процедуру; т.е. при выполнении этой ссылки вызывается процедура, определяемая с помощью фактического аргумента, соответствующего данному формальному аргументу.
2. Если контекстная область содержит тело интерфейса или спецификацию атрибута EXTERNAL для этого имени и правило (1) не применимо, ссылка является ссылкой на внешнюю процедуру с этим именем.
3. Если контекстная область содержит модульную процедуру, внутреннюю процедуру или операторную функцию с этим именем, ссылка является ссылкой на определяемую ими процедуру.
4. Если контекстная область содержит спецификацию атрибута INTRINSIC для этого имени, ссылка является ссылкой на встроенную процедуру с этим именем.

5. Если контекстная область содержит оператор USE, делающий доступной по этому имени процедуру, ссылка является ссылкой на эту процедуру. Заметим, что благодаря возможности переименования в операторе USE, имя в ссылке может отличаться от объявленного имени процедуры.
6. Если не выполняется ни одно из перечисленных выше правил, контекстная область должна находиться в объемлющей контекстной области, и ссылка разрешается применением правил из этой главы к объемлющей контекстной области.

14.1.2.4.3. Разрешение процедурных ссылок для имен, которые не определены

1. Если контекстная область является процедурой и если имя является именем формального аргумента этой процедуры, то формальный аргумент является формальной процедурой, а ссылка - ссылкой на формальную процедуру; т.е. при выполнении этой ссылки вызывается процедура, определяемая фактическим аргументом, соответствующим данному формальному аргументу.
2. Если правило (1) не выполняется, если имя является именем встроенной процедуры и если нет противоречия между ссылкой и статусом встроенной процедуры (является ли она функцией или подпрограммой), то ссылка является ссылкой на встроенную процедуру.
3. Если не выполняются правила (1) и (2), ссылка является ссылкой на внешнюю процедуру с этим именем.

14.1.2.5. Компоненты

Имя компонента имеет ту же область действия, что и тип, компонентом которого он является. Имя может появиться только в обозначении компонента структуры данного типа. Если тип доступен через связь по использованию или по объемлющему контексту (14.6.1.2) в другой контекстной области и описание типа не включает оператор PRIVATE (4.4.1), имя компонента доступно для имен компонентов структур этого типа в этой контекстной области.

14.1.2.6. Ключевые слова аргументов

Имя формального аргумента во внутренней процедуре, модульной процедуре или в блоке интерфейса процедуры имеет область действия, как ключевое слово аргумента в своей объемлющей контекстной области. В качестве ключевого слова аргумента оно может появиться только в ссылке на процедуру, формальным аргументом которой оно является. Если процедура или блок интерфейса процедуры доступны в другой контекстной области через связь по использованию или по объемлющему контексту (14.6.1.2), ключевое слово аргумента доступно для ссылок на эту процедуру в той контекстной области.

14.1.3. Операторные объекты

Имя переменной, входящей в качестве формального аргумента в описание операторной функции, имеет областью действия тот оператор, в котором находится. Переменная имеет тот тип, который она имела бы как переменная в контекстной области, содержащей эту операторную функцию.

Областью действия имени переменной, которая входит в качестве управляющей переменной неявного цикла в операторе DATA, является список неявного цикла. Она имеет те тип и параметры типа, которые имела бы, если бы была переменной в контекстной области, включающей оператор DATA или конструктор массива, и это должен быть тип целый.

Имя операторного объекта может быть также именем глобального объекта или локального объекта в этой же контекстной области; в этом случае оно интерпретируется внутри оператора как имя операторной переменной.

14.2. Область действия меток

Метка является локальным объектом. В одной контекстной области не может быть двух операторов с одинаковыми метками.

14.3. Область действия внешних устройств ввода/вывода

Внешнее устройство ввода/вывода является глобальным объектом.

14.4. Область действия знаков операций

Встроенные знаки операций - это глобальные языковые объекты. Знак определяемой операции - это локальный языковый объект. Внутри контекстной области знак операции может идентифицировать дополнительные операции, как это определено правилами для обобщенных знаков операций (12.3.2.1).

14.5. Область действия символа присваивания

Символ присваивания является глобальным языковым объектом. В контекстной области символ присваивания может идентифицировать дополнительные операции присваивания, как это определено правилами для обобщенного присваивания (12.3.2.1).

14.6. Связывание

Два объекта могут быть связаны посредством связи по имени, связи по указателю или связи по памяти. Когда объекты связываются, каждая часть одного связывается с соответствующей частью другого.

14.6.1. Связь по имени

Есть три вида **связи по имени**: связь по аргументу, связь по использованию или связь по объемлющему контексту. Все они обеспе- -вают механизмы, с помощью которых объекты, известные в одной контекстной области, могут быть доступны в другой контекстной области.

14.6.1.1. Связь по аргументу

Правила, управляющие связью по аргументу, приведены в главе 12. Как сказано в разделе 12.4, выполнение ссылки на процедуру устанавливает связь между фактическим аргументом и соответствующим ему формальным аргументом. Связь по аргументу может быть связью последовательностей (12.4.1.4).

Имя формального аргумента может отличаться от имени (если оно есть) связанного с ним фактического аргумента. (Заметим, что фактический аргумент может быть безымянным объектом данных, например, выражением, которое не является просто переменной или константой). Имя формального аргумента - это имя, под которым связанный с ним фактический аргумент известен в вызываемой процедуре и по которому к нему можно обращаться.

По окончании выполнения ссылки на процедуру все связи по аргументу, установленные этой ссылкой, прекращаются. При последующем вызове этой процедуры формальный аргумент может быть связан с совсем другим фактическим аргументом.

14.6.1.2. Связь по использованию и по объемлющему контексту

Связь по использованию - это связь имен, находящихся в разных контекстных областях и специфицированных в операторе USE. Правила такого связывания приведены в 11.3.2. Они позволяют обращаться к переименованным объектам.

Правила связи по объемлющему контексту приведены в 12.1.2.2.1.

Эти два вида связи позволяют обращаться в одной контекстной области к объектам, определенным в другой контекстной области, и действуют на протяжении всего исполнения выполняемой программы.

14.6.2. Связь по указателю

Связь между указателем и адресатом позволяет ссылаться на адресата путем ссылки на указатель. В разные моменты исполнения программы указатель может быть не определен, связан с разными адресатами или не связан вообще. Начальный статус связанности указателя - не определен. Если указатель связан с адресатом, его статус определенности определен или не определен в зависимости от статуса определенности адресата.

14.6.2.1. Статус связанности указателя

Статус связанности указателя определяется следующим образом.

1) **Связанный:** указатель становится связанным, в следующих случаях:

   а) указатель размещается (6.3.1) в результате успешного выполнения оператора ALLOCATE, ссылающегося на указатель;

   б) производится присваивание указателю адресата (7.5.2), который связан или описан с атрибутом TARGET и, если он динамически размещаемый, размещен в данный момент.

2) **Не связанный:** указатель становится не связанным, в случаях:

   а) выполняется оператор NULLIFY для этого указателя (6.3.2);

   б) указатель удаляется из памяти (6.3.3) или

   в) указателю присваивается не связанный указатель (7.5.2).

3) **Неопределенный:** статус связанности для указателя не определен в следующих случаях:

   а) начально (т.е. когда указатель еще ни разу не был связан и связь ни разу не была прервана);

   б) если адресат никогда еще не был размещен;

   в) если его адресат удален из памяти не с помощью указателя;

   г) если выполнение оператора END или RETURN делает его адресата неопределенным (6.3.3.2, п.4 в 14.7.6).

14.6.2.2. Статус определенности указателя

Статус определенности у указателя тот же, что у его адресата. Если указатель связан с определимым адресатом, его можно определять или лишать определенности в соответствии с правилами для переменных (14.7).

14.6.2.3. Взаимосвязь между статусом связанности и статусом определенности

Если статус связанности указателя не связан или не определен, на указатель нельзя ссылаться, его нельзя удалять из памяти. Каков бы ни был статус связанности, указатель всегда можно разместить или присвоить указателю, всегда можно разорвать его связь с адресатом. Если связь разорвана оператором NULLIFY, указатель не связан. При

размещении указателя он становится связанным, но не определенным. При присваивании указателю его статус связанности и статус определенности определяются его адресатом.

14.6.3. Связь по памяти

Для описания отношений, существующих между переменными, общими блоками и результирующими переменными функции, используются последовательности памяти. **Связь по памяти** - это связь двух или более объектов данных, возникающая, когда две или более последовательности памяти занимают одни и те же единицы памяти.

14.6.3.1. Последовательность памяти

Последовательность памяти - это последовательность единиц памяти. **Размер последовательности памяти** - это число единиц памяти в ней. **Единица памяти** - это либо символьная единица памяти, либо числовая единица памяти, либо неспецифицированная единица памяти.

В контексте связи по памяти имеет место следующее:

1) скалярный объект, не являющийся указателем и имеющий стандартный целый, вещественный или логический тип, занимает одну **числовую единицу памяти**;
2) скалярный объект, не являющийся указателем и имеющий вещественный двойной точности или стандартный комплексный тип, занимает две числовые единицы памяти;
3) скалярный объект, не являющийся указателем и имеющий стандартный символьный тип с длиной, равной единице, занимает одну **символьную единицу памяти**;
4) скалярный объект, не являющийся указателем и имеющий стандартный символьный тип и длины *len* занимает *len* последовательных символьных единиц памяти;
5) скалярный объект, не являющийся указателем и имеющий нестандартный целый, вещественный, логический, комплексный, символьный любой длины или непоследовательный тип, занимает одну **неспецифицированную единицу памяти**, разную в каждом из случаев;
6) массив (не являющийся указателем) встроенного или последовательного производного типа занимает последовательность непрерывных единиц памяти, по одной на каждый элемент массива, в обычном для массива порядке (6.2.2.2);
7) скалярный объект последовательного типа, не являющийся указателем, занимает последовательность единиц памяти, соответствующую последовательности образующих его компонентов;
8) указатель занимает одну неспецифицированную единицу памяти, отличную от аналогичных единиц памяти для всех объектов, не являющихся указателями, и для любой комбинации типа, параметров типа и размерности единицы памяти для указателя различны.

Последовательность последовательностей памяти образует последовательность памяти. Порядок единиц памяти в такой составной последовательности образуют отдельные единицы памяти составляющих последовательностей, игнорируя те из них, которые имеют нулевой размер.

Каждый общий блок имеет последовательность памяти (5.5.2.1).

14.6.3.2. Связь последовательностей памяти

Две последовательности памяти s_1 и s_2 **связаны по памяти**, если i-я единица памяти s_1 является j-й единицей памяти в s_2. Отсюда следует что $(i+k)$-я единица памяти s_1

является в то же время $(j+k)$-й единицей памяти s_2 для каждого целого k, такого, что $1 \leq j+k \leq$ (*размер* s_1) и $1 \leq j+k \leq$ (*размер* s_2).

Связь по памяти определена также между двумя последовательностями памяти нулевого размера и между последовательностью памяти нулевого размера и другой единицей памяти. Если последовательность памяти нулевого размера входит в последовательность последовательностей памяти и не является там последней, она связана по памяти с последующей частью. Если последующая часть - тоже последовательность памяти нулевого размера, они связаны по памяти. Если последовательность является последовательностью памяти ненулевого размера, последовательность нулевого размера связана по памяти с его первой единицей памяти. Две единицы памяти, связанные с одной последовательностью памяти нулевого размера, совпадают между собой.

14.6.3.3. Связь скалярных объектов данных

Два скалярных объекта данных связаны, если связаны их последовательности памяти. Два объекта **связаны полностью**, если они имеют одну и ту же последовательность памяти. Два объекта **связаны частично**, если они связаны, но не полностью.

Статус определенности и значение объекта данных влияют на статус определенности и значение любого связанного с ним объекта. Операторы EQUIVALENCE, COMMON или ENTRY могут вызвать связь по памяти последовательностей памяти.

Оператор EQUIVALENCE вызывает связь объектов по памяти только внутри одной контекстной области, кроме случая, когда один из эквивалентных объектов входит также и в общий блок (5.5.1.1 и 5.5.2.1).

Операторы COMMON вызывают связь объектов из одной контекстной области с объектами другой контекстной области.

Допускается, чтобы именованный общий блок содержал последовательность разных единиц памяти при условии, что каждая контекстная область, использующая общий блок, специфицировала бы идентичную последовательность единиц памяти. Это правило распространяется и на неименованные общие блоки. Если размеры двух неименованных общих блоков отличаются, последовательность единиц памяти более короткого блока должна быть идентична начальной последовательности единиц памяти более длинного блока.

Оператор ENTRY, расположенный в программной единице - функции, тоже может вызвать связь по памяти результирующих переменных.

Частичная связь может существовать только между следующими объектами:

1) объектом стандартного символьного или символьного последовательного типа и объектом стандартного символьного или символьного последовательного типа;

2) объектом стандартного комплексного, вещественного двойной точности или числового последовательного типа и объектом стандартного целого, стандартного вещественного, стандартного логического, стандартного комплексного, вещественного двойной точности или числового последовательного типа.

Для несимвольных объектов частичная связь может возникнуть только при использовании операторов COMMON, EQUIVALENCE или ENTRY. Для символьных объектов частичная связь может возникнуть при связи по аргументу и при использовании операторов COMMON, EQUIVALENCE или ENTRY.

В примере

```
REAL A(4),B
COMPLEX C(2)
DOUBLE PRECISION D
EQUIVALENCE (C(2), A(2),B), (A,D)
```

третья единица памяти C, вторая единица памяти A, единица памяти B и вторая единица памяти D описаны как совпадающие. Последовательность памяти может быть проиллюстрирована так:

```
единица памяти     | 1   | 2   | 3   | 4   | 5   |
                   |     |     |     |     |     |
                   |-- C(1) -|- C(2) --|     |
                   |     |A(1)|A(2)|A(3)|A(4)|
                   |     |     |- B-|     |     |
                   |     |--- D ---|     |     |
```

A(2) и B связаны полностью. Частично связаны: A(1) и C(1), A(2) и C(2), A(3) и C(2), B и C(2), A(1) и D, A(2) и D, B и D, C(1) и D, C(2) и D. Заметим, что хотя C(1) и C(2) каждый связан по памяти с D, друг с другом они по памяти не связаны.

Частичная связь между символьными объектами появляется, когда некоторые, но не все, единицы памяти объектов совпадают.

В примере:

```
CHARACTER A*4, B*4, C*3
EQUIVALENCE (A(2:3), B,C)
```

A, B и C частично связаны.

14.7. Определенность и неопределенность переменных

Переменная может быть определена и может быть не определена, и статус ее определенности может меняться во время исполнения выполняемой программы. Действие, которое приводит к тому, что переменная становится неопределенной, не подразумевает, что переменная была предварительно определена. Действие, определяющее переменную, не подразумевает, что перед этим переменная была не определена.

14.7.1. Определение объектов и подобъектов

Массивы, включая секции, и переменные производного, стандартного символьного, комплексного или вещественного двойной точности типа являются объектами, состоящими из нуля или более подобъектов. Могут быть установлены связи между переменными и подобъектами и между подобъектами разных переменных. Эти подобъекты могут стать определенными или неопределенными.

1. Объект определен тогда и только тогда, когда определены все его подобъекты.
2. Если объект не определен, хотя бы один (не обязательно все) из его подобъектов не определен.

14.7.2. Переменные, которые всегда определены

Массивы нулевого размера и строки нулевой длины определены всегда.

14.7.3. Переменные, которые определены начально

Следующие переменные начально определены:

1) переменные, получившие начальные значения в операторах DATA;
2) переменные, получившие начальные значения в операторах объявления типа;
3) переменные, которые всегда определены.

14.7.4. Переменные, не определенные начально

Все остальные переменные не определены начально.

14.7.5. События, которые приводят к определению объектов

Переменные становятся определенными в следующих случаях.

1. Исполнение встроенного оператора присваивания (но не присваивания массиву по маске) вызывает определение переменной, стоящей перед знаком равенства. Выполнение определяемого оператора присваивания может вызвать определение всей переменной, стоящей перед знаком равенства, или ее части.
2. Выполнение присваивания массиву по маске может вызвать определение некоторых или всех элементов массива в операторе присваивания (7.5.3).
3. Когда выполняется оператор ввода, каждая переменная, которой присваивается значение из файла ввода, становится определенной во время передачи ей этого данного (см.п.5 в 14.7.6). Выполнение оператора WRITE, спецификатор устройства в котором идентифицирует внутренний файл, вызывает определение каждой записываемой записи.
4. Выполнение оператора DO вызывает определение переменной цикла (если она есть).
5. Начало исполнения действия, заданного списком неявного цикла в операторе ввода/вывода, вызывает определение переменной неявного цикла.
6. Выполнение оператора ASSIGN вызывает определение переменной в этом операторе со значением метки оператора.
7. Ссылка на процедуру вызывает определение формального аргумента целиком, если целиком определен соответствующий фактический аргумент (но значение - не метка оператора).

   Ссылка на процедуру вызывает определение подобъекта формального аргумента, если определен соответствующий подобъект фактического аргумента.
8. Выполнение оператора ввода/вывода, содержащего спецификатор IOSTAT= , вызывает определение специфицированной целой переменной.
9. Выполнение оператора ввода/вывода, содержащего спецификатор SIZE= , вызывает определение специфицированной целой переменной.
10. Выполнение оператора INQUIRE вызывает определение любой переменной, которой присваивается значение во время исполнения этого оператора, если не возникнет ситуация ошибки.
11. Когда символьная единица памяти становится определенной, все связанные с ней символьные единицы памяти становятся определенными.

    Когда становится определенной числовая единица памяти, становятся определенными все связанные с ней числовые единицы памяти, кроме переменных, связанных с переменной из оператора ASSIGN, которые становятся неопределенными при выполнении оператора ASSIGN. Когда становится определенным вещественный двойной точности объект, становятся определенными все полностью связанные с ним объекты того же типа.

    Когда становится определенной неспецифицированная единица памяти, все связанные с ней неспецифицированные единицы памяти тоже становятся определенными.
12. Когда становится определенным объект стандартного комплексного типа, все частично связанные с ним объекты стандартного вещественного типа тоже становятся определенными.

13. Когда определяются обе части объекта стандартного комплексного типа в результате определения частично связанных с ним объектов стандартного вещественного или стандартного комплексного типа, становится определенным весь объект стандартного комплексного типа.
14. Когда все компоненты числовой последовательной структуры или символьной последовательной структуры становятся определенными в результате определения частично связанных с ней объектов, становится определенной вся структура.
15. Выполнение операторов ALLOCATE и DEALLOCATE со спецификатором STAT= вызывает определение специфицированной переменной.
16. Размещение массива нулевого размера делает его определенным.
17. Вызов процедуры приводит к определению всех динамических объектов нулевого размера в этой процедуре.
18. Выполнение присваивания указателю, связывающее указатель с адресатом, делает указатель определенным.

14.7.6. События, приводящие к неопределенности объектов

Переменные становятся неопределенными в следующих случаях.

1. Когда переменная данного типа становится определенной, все полностью связанные с ней переменные другого типа становятся неопределенными. Однако, когда переменная стандартного вещественного типа частично связана с переменной стандартного комплексного типа, комплексная переменная не становится неопределенной в результате определения вещественной переменной, а вещественная переменная не становится неопределенной, когда определена комплексная переменная. Когда переменная стандартного комплексного типа частично связана с другой переменной стандартного комплексного типа, определение одной переменной не вызывает неопределенности другой.
2. Выполнение оператора ASSIGN вызывает неопределенность переменной в этом операторе как переменной целого типа. Переменные, связанные с этой переменной, тоже становятся неопределенными.
3. Если вычисление функции может вызвать определение аргумента этой функции, либо переменной из общего блока или из модуля, и если ссылка на функцию появляется в выражении, в котором для вычисления его значения не нужно значение данной функции, то аргумент или переменная из общего блока или из модуля становятся неопределенными при вычислении значения выражения.
4. Выполнение оператора RETURN или END внутри процедуры вызывает неопределенность всех переменных, локальных в ее контекстной области или, в случае рекурсивного вызова, локальных в текущем экземпляре ее контекстной области, кроме следующих переменных:
   а) переменных с атрибутом SAVE;
   б) переменных из неименованного общего блока;
   в) переменных из именованного общего блока, который имеется в данной процедуре и хотя бы в одной другой контекстной области, которая либо прямо, либо косвенно ссылается на эту процедуру;
   г) переменных, доступных из объемлющей контекстной области;
   д) переменных, доступных из модуля, на которые прямо или косвенно ссылается хотя бы одна из контекстных областей, прямо или косвенно ссылающихся на данную процедуру;

е) переменных из именованных общих блоков, которые были начально определены (14.7.3) и впоследствии не переопределялись или не теряли определенности.

5. Когда во время исполнения оператора ввода возникает либо ситуация ошибки, либо ситуация конца файла, все переменные, заданные списком ввода или *группой-NAMELIST*, становятся неопределенными.

6. Когда во время выполнения оператора ввода/вывода возникает ситуация ошибки, конца файла или конца записи, некоторые или все переменные неявных циклов могут стать неопределенными (9.4.3).

7. Выполнение оператора определяемого присваивания может оставить неопределенной переменную, стоящую перед знаком равенства, или ее часть.

8. Выполнение оператора ввода прямого доступа, который специфицирует запись, не записанную предварительно, вызывает неопределенность всех переменных, перечисленных в списке ввода этого оператора.

9. Выполнение оператора INQUIRE может вызвать неопределенность переменных в спецификаторах NAME=, REAL= и NEXTREC= (9.6).

10. Когда становится неопределенной символьная единица памяти, все связанные с ней символьные единицы памяти тоже становятся неопределенными.

    Когда становится неопределенной числовая единица памяти, все связанные с ней числовые единицы памяти тоже становятся неопределенными, кроме случаев, когда неопределенность есть результат определения связанной числовой единицы памяти с другим типом (см. 1) выше).

    Когда становится неопределенным объект вещественный двойной точности, все полностью связанные с ним объекты того же типа тоже становятся неопределенными.

    Когда становится неопределенной неспецифицированная единица памяти, все связанные с ней неспецифицированные единицы памяти тоже становятся неопределенными.

11. Ссылка на процедуру приводит к неопределенности части формального аргумента, если соответствующая часть фактического аргумента определена со значением метки оператора.

12. Когда динамически размещаемый массив становится неразмещенным, он становится и неопределенным. Успешное выполнение оператора ALLOCATE делает его определенным.

13. Возникновение ошибки при выполнении оператора INQUIRE вызывает неопределенность всех переменных в спецификаторах запроса, кроме переменной в спецификаторе IOSTAT=, если он есть.

14. Когда процедура вызвана:

    а) необязательный формальный аргумент, не связанный с фактическим, не определен;

    б) не определен формальный аргумент с атрибутом INTENT(OUT);

    в) фактический аргумент, связанный с формальным аргументом с атрибутом INTENT(OUT), становится неопределенным;

    г) не определен подобъект формального аргумента, если соответствующий подобъект фактического аргумента не определен;

    д) результирующая переменная функции не определена.

15. Когда значение статуса связанности указателя - "не определен" или "не связан" (6.3), указатель становится неопределенным.

14.8. Статус размещенности

Статус размещенности динамически размещаемого массива может во время исполнения выполняемой программы принимать следующие значения.

1. "Не размещенный в данный момент", что означает, что массив или никогда не был размещен или последней операцией над ним было выполнение оператора DEALLOCATE.
2. "Размещен в данный момент", что означает, что массиву была отведена память с помощью оператора ALLOCATE и впоследствии этот массив не становился неразмещенным.
3. "Не определен", что означает, что массив не имеет атрибута SAVE и был размещен, а выполнение оператора RETURN или END привело к тому, что не осталось контекстных областей, имеющих к нему доступ.

Если статус размещенности - "размещен в данный момент", на массив можно ссылаться и можно его определять. Если статус размещенности - "не размещен в данный момент", на массив нельзя ссылаться, его нельзя определять. Если же статус - "не определен", на массив нельзя ссылаться, его нельзя определять, размещать, делать неразмещенным.

Приложение А
(информационное)

Словарь технических терминов

Приводится список основных технических терминов, используемых в Международном стандарте, и их описания. Сразу же после перевода термина в скобках указывается номер раздела, где термин определяется или объясняется. Если объяснение использует термин, который тоже определяется в данном словаре, первое вхождение термина в это объяснение выделяется курсивом.

ACTION STATEMENT (оператор действия) (2.1): отдельный *оператор* (*statement*), специфицирующий действие (R216).

ACTUAL ARGUMENT (фактический аргумент) (12.4.1): *выражение* (*expression*), *переменная* (*variable*), *процедура* (*procedure*) или спецификатор альтернативного возврата, заданный в *ссылке* (*reference*) на процедуру.

ALLOCATABLE ARRAY (динамически размещаемый массив) (5.1.2.4.3): *именованный массив* (*named array*), имеющий *атрибут* (*attribute*) ALLOCATABLE. Только тогда, когда ему отведена память, определяется его *конфигурация* (*shape*), на него можно *ссылаться* (*reference*), его можно *определять* (*define*).

ARGUMENT (аргумент, в литературе часто - **параметр** (*прим. ред.*)) (12): *фактический аргумент* (*actual argument*) или *формальный аргумент* (*dummy argument*).

ARGUMENT ASSOCIATION (связь, связывание аргументов) (14.6.1.1): отношения между *фактическим аргументом* (*actual argument*) и *формальным аргументом* (*dummy argument*) во время выполнения *ссылки на процедуру* (*procedure reference*).

ARGUMENT KEYWORD (ключевое слово аргумента) (2.5.2): *имя формального аргумента* (*dummy argument name*). Оно может использоваться в *ссылке на процедуру* (*procedure reference*) перед знаком равенства (R1211) при условии, что процедура имеет *явный интерфейс* (*explicit interface*).

ARRAY (массив) (2.4.5): множество скалярных *данных* (*data*) одного *типа* (*type*) и с одинаковыми *параметрами типа* (*type parameters*), отдельные элементы которого выстроены по прямоугольной схеме. Это может быть *именованный* (*named*) массив, *секция массива* (*array section*), *компонент структуры* (*structure component*), значение *функции* (*function*), *выражение* (*expression*). *Размерность* (*rank*) его, как минимум, единица. Заметим, что в ФОРТРАНе 77 массивы были всегда именованными и никогда не являлись константами.

ARRAY ELEMENT (элемент массива) (2.4.5, 6.2.2.1): одно из *скалярных данных* (*scalar data*), образующих *массив* (*array*) - либо *именованный* (*named*), либо *компонент структуры* (*structure component*).

ARRAY POINTER (указатель на массив) (5.1.2.4.3): *указатель* (*pointer*) на *массив* (*array*).

ARRAY SECTION (секция массива) (6.2.2.3): *подобъект* (*subobject*), который является *массивом* (*array*), но не является *компонентом структуры* (*structure component*).

ARRAY-VALUED (со значением-массивом): обладающий свойствами *массива* (*array*).

ASSIGNMENT STATEMENT (оператор присваивания) (7.5.1.1): *оператор* (*statement*) вида "*переменная = выражение* (*variable = expression*)".

ASSOCIATION (связь) (14.6): *связь по имени* (*name association*), *связь по указателю* (*pointer association*) или *связь по памяти* (*storage association*).

ASSUMED-SIZE ARRAY (массив неопределенного размера) (5.1.2.4.4): *формальный массив* (*dummy array*), *размер* (*size*) которого определяется размером связанного с ним *фактического аргумента* (*actual argument*). Последняя верхняя граница такого формального массива задается звездочкой.

ATTRIBUTE (атрибут) (5): свойство *объекта данных* (*data object*), которое может быть специфицировано в *операторе объявления типа* (*type declaration statement*) (R501).

AUTOMATIC DATA OBJECT (автоматически размещаемый объект данных) (5.1): *объект данных* (*data object*), который, будучи *локальным объектом* (*local object*) в *процедуре Фортрана* (*subprogram*), не является *формальным аргументом* (*dummy argument*) и имеет неконстантную длину (для символьных) или неконстантную границу массива.

BELONG (принадлежит) (8.1.4.4.3, 8.1.4.4.4): если *оператор* (*statement*) EXIT или CYCLE содержит *имя конструкции* (*construct name*), оператор **принадлежит** конструкции DO, использующей это имя. В противном случае он **принадлежит** самой внутренней из конструкций DO, в которых он находится.

BLOCK (блок) (8.1): последовательность *выполняемых конструкций* (*executable constructs*), которая вложена в другую выполняемую конструкцию и ограничена *операторами* (*statements*), характерными для этой конструкции, и трактуется как целостная единица.

BLOCK DATA PROGRAM UNIT (программная единица - блок данных) (11.4): *программная единица*, обеспечивающая начальные значения для *объектов данных* (*data objects*) из *именованных общих блоков* (*named common blocks*).

BOUNDS (границы) (5.1.2.4.1): для *именованного массива* (*named array*) - границы, внутри которых должны находиться значения *индексов* (*subscripts*) *элементов массива* (*array elements*).

CHARACTER (символ) (3.1): буква, цифра или другая литера.

CHARACTERISTICS (характеристики) (12.2):

1) для *процедуры* (*procedure*)- это ее классификация как *функции* (*function*) или *подпрограммы* (*subroutine*), характеристики ее *формальных аргументов* (*dummy arguments*) и, если это функция, характеристики ее *результата* (*function result*);

2) для *формального аргумента* (*dummy argument*) - является ли он *объектом данных* (*data object*), *процедурой* (*procedure*) и имеет ли *атрибут* (*attribute*) OPTIONAL;

3) для *объекта данных* (*data object*) - это его *тип* (*type*), *параметры типа* (*type parameters*), *конфигурация* (*shape*), точная зависимость границ массива или длины (для символьных) от других объектов, *назначение* (*intent*), является ли он необязательным, является ли он *указателем* (*pointer*) или *адресатом* (*target*), являются ли неопределенными его конфигурация, размер или (для символьных) *длина* (*length*);

4) для *формальной процедуры* (*dummy procedure*) - явный ли у нее интерфейс, характеристики процедуры, если интерфейс явный, и является ли она необязательной;

5) для *результата функции* (*function result*) - его тип, параметры типа, является ли он указателем, размерность - если указатель, конфигурация - если не указатель, точная зависимость границы массива или длины (для символьных) от других объектов, и, наконец, является ли неопределенной длина (для символьных).

CHARACTER STRING (символьная строка) (4.3.2.1): последовательность *символов* (*characters*), пронумерованная слева направо 1, 2, 3,...

CHARACTER STORAGE UNIT (символьная единица памяти) (14.6.3.1): единица памяти для хранения скаляра - не *указателя* (*pointer*) стандартного символьного *типа* (*type*) и единичной длины.

COLLATING SEQUENCE (упорядочивающая последовательность) (4.3.2.1.1): упорядочение всех отличных друг от друга *символов* (*characters*) для конкретного *параметра вида* (*kind type parameter*).

COMMON BLOCK (общий блок) (5.5.2): блок физической памяти, к которому может быть доступ из любой *контекстной области* (*scoping unit*) в *выполняемой программе* (*executable program*).

COMPONENT (компонент) (4.4): составляющая *производного типа* (*derived type*).

CONFORMABLE (согласованный) (2.4.5): говорят, что два *массива* (*arrays*) **согласованы**, если у них одинаковая *конфигурация* (*shape*). *Скаляр* (*scalar*) **согласован** с любым массивом.

CONFORMANCE (соответствие) (1.4): *выполняемая программа* (*executable program*) соответствует стандарту, если она использует только те формы и отношения, которые описаны в нем, и если она имеет интерпретацию, соответствующую стандарту. *Программная единица* (*program unit*) соответствует стандарту, если ее можно включить в выполняемую программу способом, позволяющим выполняемой программе соответствовать стандарту. *Процессор* (*processor*) соответствует стандарту, если он выполняет стандартную программу так, чтобы осуществлялась интерпретация, предписанная стандартом.

CONNECTED (подсоединенный) (9.3.2):

1) для *внешнего устройства* (*external unit*) - состояние устройства, когда оно связано с *внешним файлом* (*external file*);

2) для *внешнего файла* (*external file*) - состояние файла, когда он связан с *внешним устройством* (*external unit*).

CONSTANT (константа) (2.4.3.1.2): *объект данных* (*data object*), значение которого не должно изменяться во время исполнения *выполняемой программы* (*executable program*). Это может быть либо *именованная константа* (*named constant*), либо *литеральная константа* (*literal constant*).

CONSTANT EXPRESSION (константное выражение) (7.1.6.1): *выражение* (*expression*), удовлетворяющее правилам, которые гарантируют неизменность его значения во время исполнения выполняемой программы.

CONSTRUCT (конструкция) (8): последовательность *операторов* (*statements*), начинающаяся с операторов SELECT CASE, DO, IF или WHERE и заканчивающаяся соответствующим конечным оператором.

DATA (данные): Множественное число от DATUM.

DATA ENTITY (данные) (2.4.3): *объект данных* (*data object*), результат вычисления *выражения* (*expression*) либо результат выполнения *ссылки на функцию* (*function reference*), называемый *результатом функции* (*function result*). Данные имеют *тип* (*data type*), *встроенный* (*intrinsic*) или *производный* (*derived*), и имеют или могут иметь значение данного, исключение - *неопределенная переменная* (*undefined variable*). Каждое данное имеет *размерность* (*rank*) и является, таким образом, либо *скаляром* (*scalar*), либо *массивом* (*array*).

DATA OBJECT (объект данных) (2.4.3): *данное* (*data entity*), которое является *константой* (*constant*), *переменной* (*variable*) или *подобъектом* (*subobject*) константы.

DATA TYPE (тип данных) (2.4.1): *именованная* (*named*) категория *данных*, которая характеризуется множеством значений, способом обозначения этих значений и множеством *операций* (*operations*), с помощью которых обрабатываются и интерпретируются эти значения. Для *встроенного* (*intrinsic*) типа множество значений данных зависит от значений *параметров типа* (*type parameters*).

DATUM (данное): отдельная величина, которая может принимать любое из множества значений, специфицированных для ее *типа* (*data type*).

DEFINABLE (определимый) (2.5.4): *переменная* (*variable*) является **определимой**, если ее значение может быть изменено включением ее *имени* (*name*) или *обозначения* (*designator*) в левую часть *оператора присваивания* (*assignment statement*). *Динамически размещаемый массив* (*allocatable array*), который еще не размещен, может служить примером неопределимого *объекта данных* (*data object*). Примером неопределимого *подобъекта* (*subobject*) является C(I), где C - *массив* (*array*), который является *константой* (*constant*), а I - целая переменная.

DEFINED (определено) (2.5.4): для *объекта данных* (*data object*) - статус объекта, который имеет или получил правильное значение.

DEFINED ASSIGNMENT STATEMENT (определяемый оператор присваивания) (7.5.1.3): *оператор присваивания* (*assignment statement*), который не является *встроенным* (*intrinsic*) оператором присваивания и определяется *подпрограммой* (*subroutine*) и блоком интерфейса со спецификацией ASSIGNMENT (=).

DEFINED OPERATION (определяемая операция) (7.1.3): *операция* (*operation*), которая не является *встроенной* (*intrinsic*) операцией и определяется *функцией* (*function*), связанной с *обобщенным идентификатором* (*generic identifier*).

DELETED FEATURE (исключенное средство) (1.6): средство из ФОРТРАНа 77, которое рассматривается как избыточное и малоиспользуемое. Ни одно из средств ФОРТРАНа 77 сейчас из стандарта не исключено. Заметим, что средство, обозначенное в стандарте как *отжившее* (*obsolescent feature*), может при следующей ревизии стать исключенным.

DERIVED TYPE (производный тип) (2.4.1.2): *тип* (*type*), *данные* (*data*) которого имеют *компоненты* (*components*), относящиеся либо к встроенному, либо к другому производному типу.

DESIGNATOR (обозначение): см. *SUBOBJECT DESIGNATOR*.

DISASSOCIATED (не связанный) (2.4.6): *указатель* (*pointer*) становится **не связанным** после выполнения *оператора* (*statement*) DEALLOCATE или NULLIFY, либо после присваивания ему не связанного указателя.

DUMMY ARGUMENT (формальный аргумент) (12.5.2.2, 12.5.2.3, 12.5.2.5, 12.5.4): языковый объект, *имя* (*name*) которого находится в заключенном в скобки списке, следующем за именем *процедуры* (*procedure*) в *операторе* (*statement*) FUNCTION, SUBROUTINE или ENTRY, либо в описании *операторной функции* (*statement function*).

DUMMY ARRAY (формальный массив): *формальный аргумент* (*dummy argument*), который является *массивом* (*array*).

DUMMY POINTER (формальный указатель): *формальный аргумент* (*dummy argument*), который является *указателем* (*pointer*).

DUMMY PROCEDURE (формальная процедура) (12.1.2.3): *формальный аргумент* (*dummy argument*), который специфицирован как *процедура* (*procedure*) или на который *ссылаются* (*reference*), как на процедуру.

ELEMENTAL (поэлементный) (12.4.3, 12.4.5): прилагательное, относящееся к *встроенной операции* (*intrinsic operation*), *процедуре* (*procedure*) или к *оператору присваивания* (*assignment statement*), которые применяются независимо к элементам *массива* (*array*) или к соответствующим элементам набора *согласованных* (*conformable*) *массивов* (*arrays*) и к *скалярам* (*scalars*).

ENTITY (языковый объект): термин, используемый для следующих вещей: *программная единица* (*program unit*), *процедура* (*procedure*), *знак операции* (*operator*), *блок интерфейса* (*interface block*), *общий блок* (*common block*), *внешнее устройство* (*external unit*), *операторная функция* (*statement function*), *тип* (*type*), *именованная переменная* (*named variable*), *выражение* (*expression*), *компонент структуры* (*component of a structure*), *именованная константа* (*named constant*), *метка оператора* (*statement label*), *конструкция* (*construct*), группа NAMELIST.

EXECUTABLE CONSTRUCT (выполняемая конструкция) (2.1): *конструкция* (*construct*) CASE, DO, IF, WHERE или *оператор действия* (*action statement*) (R216).

EXECUTABLE PROGRAM (выполняемая программа) (2.2.2): множество *программных единиц* (*program units*), в котором есть ровно одна *главная программная единица* (*main program*).

EXECUTABLE STATEMENT (выполняемый оператор) (2.3.1): инструкция для выполнения или управления одним или несколькими вычислительными действиями.

EXPLICIT INTERFACE (явный интерфейс) (12.3.1): *процедура* (*procedure*) имеет явный интерфейс, если на нее есть *ссылка* (*reference*) в контекстной области, если она является *внутренней процедурой* (*internal procedure*), *модульной процедурой* (*module procedure*), *встроенной процедурой* (*intrinsic procedure*), *внешней процедурой* (*external procedure*), имеющей *блок интерфейса* (*interface block*), *рекурсивной процедурой* (*recursive procedure*), на которую есть ссылка из ее собственной контекстной области, или *формальной процедурой* (*dummy procedure*), имеющей блок интерфейса.

EXPLICIT-SHAPE ARRAY (массив явной конфигурации) (5.1.2.4.1): *именованный массив* (*named array*), объявленный с явно заданными *границами* (*bounds*).

EXPRESSION (выражение) (7.1): последовательность *операндов* (*operands*), *знаков операций* (*operators*) и скобок (R723). Оно может быть *переменной* (*variable*), *константой* (*constant*), *ссылкой на функцию* (*function reference*) либо может представлять вычисление.

EXTENT (протяженность) (2.4.5): размер одного измерения *массива* (*array*).

EXTERNAL FILE (внешний файл) (9.2.1): последовательность *записей* (*records*), которая существует в среде, внешней по отношению к *выполняемой программе* (*executable program*).

EXTERNAL PROCEDURE (внешняя процедура) (2.2.3.1): *процедура* (*procedure*), определяемая либо через *внешнюю процедуру Фортрана* (*external subprogram*), либо отличным от Фортрана способом.

EXTERNAL SUBPROGRAM (внешняя процедура Фортрана) (2.2): *процедура Фортрана* (*subprogram*), не входящая ни в *главную программную единицу* (*program unit*), ни в модуль (*module*), ни в другую процедуру Фортрана. Заметим, что модуль не называется процедурой Фортрана. В ФОРТРАНе 77 *программная единица - блок данных* (*block data program unit*) называлась процедурой.

EXTERNAL UNIT (внешнее устройство) (9.3): механизм, используемый для ссылки на *внешний файл* (*external file*). Идентифицируется неотрицательным целым.

FILE (файл) (9.2): *внутренний файл* (*internal file*) или *внешний файл* (*external file*).

FUNCTION (функция) (2.2.3): *процедура* (*procedure*), вызываемая в *выражении* (*expression*).

FUNCTION RESULT (результат функции) (12.5.2.2): *объект данных* (*data object*), выдающий значение *функции* (*function*).

FUNCTION SUBPROGRAM (процедура-функция) (12.5.2.2): последовательность *операторов* (*statements*), начинающаяся с оператора FUNCTION, которая не является *блоком интерфейса* (*interface block*) и заканчивается соответствующим оператором END.

GENERIC IDENTIFIER (обобщенный идентификатор): *лексема* (*lexical token*), входящая в оператор INTERFACE и связанная со всеми *процедурами* (*procedures*) в *блоке интерфейса* (*interface block*).

GLOBAL ENTITY (глобальный объект) (14.1.1): *языковый объект* (*entity*), идентифицированный *лексемой* (*lexical token*), *областью действия* (*scope*) которого является *выполняемая программа* (*executable program*). Это может быть *программная единица* (*program unit*), *общий блок* (*common block*) или *внешняя процедура* (*external procedure*).

HOST (объемлющий контекст) (2.2.3.3): *главная программная единица* (*main program*) или *процедура* (*procedure*), которая содержит *внутреннюю процедуру* (*internal procedure*), называется **объемлющим контекстом** внутренней процедуры. *Модуль* (*module*), содержащий *модульную процедуру* (*module procedure*), называется **объемлющим контекстом** модульной процедуры.

HOST ASSOCIATION (связь по объемлющему контексту) (11.2.2): процесс, посредством которого *внутренняя процедура* (*internal procedure*), *модульная процедура* (*module procedure*) и описание *производного типа* (*derived type*) осуществляют доступ к *языковым объектам* (*entities*) своего *объемлющего контекста* (*host*).

IMPLICIT INTERFACE (неявный интерфейс) (12.3.1): говорят, что *процедура* (*procedure*), на которую есть *ссылка* (*reference*) в *контекстнойобласти* (*scoping unit*) (но не в ее собственной), имеет **неявный интерфейс**, если она является *внешней процедурой* (*external procedure*), не имеющей *блока интерфейса* (*interface block*), *формальной процедурой* (*dummy procedure*), не имеющей блока интерфейса, или *операторной функцией* (*statement function*).

INQUIRY FUNCTION (справочная функция) (13.1): *встроенная функция* (*intrinsic function*), результат которой зависит не от значения основного *аргумента* (*argument*), а от его свойств.

INTENT (назначение) (12.5.2.1): атрибут *формального аргумента* (*dummy argument*), не являющегося ни *процедурой* (*procedure*), ни *указателем* (*pointer*); определяет используется ли формальный аргумент для передачи данных в процедуру или из нее либо для того и для другого.

INSTANCE OF SUBPROGRAM (экземпляр процедуры) (12.5.2.4): копия *процедуры Фортрана* (*subprogram*), создаваемая при ее *вызове* (*invoke*).

INTERFACE BLOCK (блок интерфейса) (12.3.2.1): последовательность *операторов* (*statements*), начинающаяся с оператора INTERFACE и кончая соответствующим оператором END INTERFACE.

INTERFACE BODY (тело интерфейса) (12.3.2.1): последовательность *операторов* (*statements*) в *блоке интерфейса* (*interface block*), начиная с оператора FUNCTION или SUBROUTINE и кончая соответствующим оператором END.

INTERFACE OF PROCEDURE (интерфейс процедуры) (12.3): см. PROCEDURE INTERFACE.

INTERNAL FILE (внутренний файл) (9.2.2): символьная *переменная* (*variable*), которая используется для передачи *данных* (*data*) с преобразованием из внутренней памяти во внутреннюю память.

INTERNAL PROCEDURE (внутренняя процедура) (2.2.3.3): *процедура* (*procedure*), определяемая *внутренней процедурой Фортрана* (*internal subprogram*).

INTERNAL SUBPROGRAM (внутренняя процедура Фортрана) (2.2): *процедура Фортрана* (*subprogram*), расположенная внутри *главной программной единицы* (*main program*) или другой процедуры.

INTRINSIC (встроенный) (2.5.7): прилагательное, применяемое к *типам* (*types*), *операциям* (*operations*), *операторам присваивания* (*assignment statements*) и *процедурам* (*procedures*), которые определяются в стандарте и могут быть использованы в любой *контекстной области* (*scoping unit*) без последующего описания или спецификации.

INVOKE (вызов) (2.2.3):

1) вызов *подпрограммы* (*subroutine*) посредством *оператора* (*statement*) CALL или *оператора определяемого присваивания* (*defined assignment statement*);

2) вызов *функции* (*function*) посредством *ссылки* (*reference*) на ее имя или при помощи *знака операции* (*operator*) во время вычисления *выражения* (*expression*).

KEYWORD (ключевое слово) (2.5.2): *ключевое слово оператора* (*statement keyword*) или *ключевое слово аргумента* (*argument keyword*).

KIND TYPE PARAMETER (параметр вида): параметр, значения которого определяют доступные виды *встроенного типа* (*intrinsic type*).

LABEL (метка): см. *STATEMENT LABEL*.

LENGTH OF A CHARACTER STRING (длина символьной строки) (4.3.2.1): число *символов* (*characters*) в *символьной строке* (*character string*).

LEXICAL TOKEN (лексема) (3.2): последовательность из одного или нескольких символов с неделимой интерпретацией.

LINE (строка) (3.3.1): *запись* (*record*) исходного текста, содержащая от 0 до 132 символов.

LITERAL CONSTANT (литеральная константа) (2.4.3.1.2): *константа* (*constant*) без *имени* (*name*). Заметим, что в ФОРТРАНе 77 это называлось просто константой.

LOCAL ENTITY (локальный объект) (14.1.2): *языковый объект*, идентифицированный *лексемой* (*lexical token*), *областью действия* (*scope*) которой является некоторая *контекстная область* (*a scoping unit*).

MAIN PROGRAM (главная программная единица) (2.2.2): *программная единица* (*program unit*), которая не является ни *модулем* (*module*), ни *процедурой Фортрана* (*subprogram*), ни *программной единицей - блоком данных* (*block data program unit*).

MANY-ONE ARRAY SECTION (многозначная секция массива) (6.2.2.3.2): *секция массива* (*array section*) с *векторным индексом* (*vector subscript*), имеющим два или более элементов с одинаковым значением.

MODULE (модуль) (2.2.4): *программная единица* (*program unit*), содержащая описания (или имеющая доступ к описаниям), к которым имеют доступ другие программные единицы.

MODULE PROCEDURE (модульная процедура) (2.2.3.2): *процедура* (*procedure*), определяемая *модульной процедурой Фортрана* (*module subprogram*).

MODULE SUBPROGRAM (модульная процедура Фортрана) (2.2): *процедура Фортрана* (*subprogram*), находящаяся внутри *модуля* (*module*), но не являющаяся *внутренней процедурой* (*internal subprogram*).

NAME (имя) (3.2.2): *лексема* (*lexical token*), состоящая из буквы и следующими за ней до 30 алфавитно-цифровых символов (буквы, цифры, символы подчеркивания). Заметим, что в ФОРТРАНе 77 это называлось символическим именем.

NAME ASSOCIATION (связь по имени) (14.6.1): *связь по аргументу* (*argument association*), *связь по использованию* (*use association*), *связь по объемлющему контексту* (*host association*).

NAMED (именованный): обладающий *именем* (*name*).

NAMED CONSTANT (именованная константа) (2.4.3.1.2): *константа*, обладающая *именем*. Заметим, что в ФОРТРАНе 77 это называлось символической константой.

NUMERIC STORAGE UNIT (числовая единица памяти) (14.6.3.1): единица памяти для хранения *скаляра* (*scalar*), не являющегося указателем и имеющего стандартный вещественный, стандартный целый или стандартный логический *тип* (*type*).

NUMERIC TYPE (числовой тип): целый, вещественный или комплексный *тип* (*type*).

OBJECT (объект) (2.4.3.1): *объект данных* (*data object*).

OBSOLESCENT FEATURE (устаревшее средство) (1.6): средство из ФОРТРАНа 77, которое считается избыточным, но пока еще обычно используется.

OPERAND (операнд) (2.5.8): *выражение* (*expression*), стоящее до или после *знака операции* (*operator*).

OPERATION (операция) (7.1.2): вычисление, включающее один или два *операнда* (*operands*).

OPERATOR (знак операции) (2.5.8): *лексема* (*lexical token*), специфицирующая *операцию* (*operation*).

POINTER (указатель) (2.4.6): *переменная* (*variable*), которая имеет атрибут POINTER. Если *указатель* (*pointer*) не *связан* (*associated*) с *адресатом* (*target*), на него нельзя *ссылаться* (*reference*), его нельзя *определять* (*define*). Если он объявлен как *массив* (*array*), он не имеет *конфигурации* (*shape*) до тех пор, пока не станет связанным (*associated*) с адресатом.

POINTER ASSIGNMENT (присваивание указателю) (7.5.2): установление *связи* (*pointer association*) между *указателем* (*pointer*) и *адресатом* (*target*) посредством выполнения *оператора присваивания указателю* (*pointer assignment statement*) или *оператора присваивания* (*assignment statement*) для *объекта данных* (*data object*) *производного типа* (*derived type*), имеющего *подобъект* (*subobject*) - указатель.

POINTER ASSIGNMENT STATEMENT (оператор присваивания указателю) (7.5.2): *оператор* (*statement*) вида *"имя-указателя =>адресат"* ("*pointer-name => target* ").

POINTER ASSOCIATED (связанный по указателю) (6.3, 7.5.2): отношения между *указателем* (*pointer*) и *адресатом* (*target*), установленные посредством *присваивания указателю* (*pointer assignment*) или успешного выполнения *оператора* (*statement*) ALLOCATE.

POINTER ASSOCIATION (связь указателя) (14.6.2): процесс установления *связи* (*association*) *указателя* (*pointer*) с *адресатом* (*target*).

PRESENT (присутствует) (12.5.2.8): *формальный аргумент* (*dummy argument*) **присутствует** в *экземпляре процедуры* (*instance of a subprogram*), если он *связан* с *фактическим аргументом* (*actual argument*), и этот фактический аргумент либо является формальным аргументом, присутствующим в вызывающей *процедуре* (*procedure*), либо не является формальным аргументом вызывающей процедуры.

PROCEDURE (процедура) (2.2.3): вычисление, которое может быть *вызвано* (*invoked*) во время выполнения программы. Это может быть *функция* (*function*) или *подпрограмма* (*subroutine*). Это может быть *встроенная процедура* (*intrinsic procedure*), *внешняя процедура* (*external procedure*), *модульная процедура* (*module procedure*), *внутренняя процедура* (*internal procedure*), *формальная процедура* (*dummy procedure*) или *операторная функция* (*statement function*). *Процедура Фортрана* (*subprogram*) может определять более чем одну процедуру, если содержит *операторы* (*statements*) ENTRY.

PROCEDURE INTERFACE (интерфейс процедуры) (12.3): *характеристики процедуры* (*characteristics of a procedure*), ее *имя* (*name*), имя каждого из ее *формальных аргументов* (*dummy arguments*) и *обобщенные идентификаторы* (*generic identifiers*), если они есть, по которым на нее можно *ссылаться* (*reference*).

PROCESSOR (процессор) (1.2): сочетание вычислительной системы и механизма преобразования *выполняемых программ* (*executable programs*) для их использования в этой вычислительнойсистеме.

PROGRAM (программа): См. *выполняемая программа* (*executable program*) и *главная программная единица* (*main program*).

PROGRAM UNIT (программная единица) (2.2): основной компонент *выполняемой программы* (*executable program*). Последовательность *операторов* (*statements*) и строк комментария. Это может быть *главная программная единица* (*main program*), *модуль* (*module*), *внешняя процедура Фортрана* (*external subprogram*) или *программная единица - блок данных* (*block data program unit*).

RANK (размерность, в литературе часто - **ранг** (*прим. ред.*)) (2.4.5): число измерений *массива* (*array*). Для *скаляра* (*scalar*) - нуль.

RECORD (запись) (9.1): последовательность значений внутри *файла* (*file*), трактуемая как единое целое.

REFERENCE (ссылка) (2.5.5): вхождение *имени объекта данных* (*data object name*) или *обозначения подобъекта* (*subobject designator*) в контекст, требующий его значение в данной точке во время исполнения, либо появление имени *процедуры* (*procedure*), символа ее *знака операции* (*operator*) или *оператора определяемого присваивания* (*defined assignment statement*) в контексте, требующем исполнения процедуры в этой точке. Заметим, что ни определение *переменной* (*variable*), ни появление имени процедуры в качестве *фактического аргумента* (*actual argument*) не рассматривается как ссылка.

SCALAR (скаляр) (2.4.6):

1) отдельное *данное* (*datum*), не являющееся *массивом* и

2) не имеющее свойств *массива* (*array*).

SCOPE (область действия) (14): такая часть *выполняемой программы* (*executable program*), внутри которой *лексема* (*lexical token*) имеет единственную интерпретацию. Это может быть *выполняемая программа* (*executable program*), *контекстная область* (*scoping unit*), отдельный *оператор* (*statement*) или часть оператора.

SCOPING UNIT (контекстная область) (2.2): одно из следующих:

1) описание *производного типа* (*derived type*),

2) *тело интерфейса* (*interface body*), исключая содержащиеся в нем тела интерфейса и описания производного типа или

3) *программная единица* (*program unit*) или *процедура Фортрана* (*subprogram*), исключая описания производных типов, тела интерфейса и процедуры, содержащиеся в них.

SECTION SUBSCRIPT (индекс секции) (6.2.2): *индекс* (*subscript*), *векторный индекс* (*vector subscript*) или *индексный триплет* (*subscript triplet*) в *селекторе секции массива* (*array section selector*).

SELECTOR (селектор): синтаксический механизм для обозначения следующего:

1) части *объекта данных* (*data object*). Он может обозначать *подстроку* (*substring*), *элемент массива* (*array element*), *секцию массива* (*array section*) или *компонент структуры* (*structure component*);

2) множества значений, для которых выполняется *блок* (*block*) CASE.

SHAPE (конфигурация) (2.4.5): для *массива* (*array*) - *размерность* (*rank*) и *протяженности* (*extents*). Конфигурация может быть представлена одномерным массивом, элементами которого являются протяженности в каждом измерении.

SIZE (размер) (2.4.5): для *массива* (*array*) - общее число элементов.

STANDARD MODULE (стандартный модуль) (1.7): *модуль* (*module*), стандартизованный как отдельный сопутствующий стандарт.

STATEMENT (оператор) (3.3): последовательность *лексем* (*lexical tokens*). Обычно состоит из одной строки, но можно использовать символ "&" для продолжения оператора с одной строки на другую или символ ";" для разделения операторов в одной строке.

STATEMENT ENTITY (операторный объект) (14): *языковый объект* (*entity*), идентифицированный *лексемой* (*lexical token*), *областью действия* (*scope*) которого является один *оператор* (*statement*) или часть оператора.

STATEMENT FUNCTION (операторная функция) (12.5.4): *процедура* (*procedure*), специфицированная одним *оператором* (*statement*), похожим по форме на *оператор присваивания* (*assignment statement*).

STATEMENT KEYWORD (ключевое слова оператора) (2.5.2): слово, которое является частью синтаксиса *оператора* (*statement*) и может использоваться для его идентификации.

STATEMENT LABEL (метка оператора) (3.2.5): *лексема* (*lexical token*), образованная из цифр (от одной до пяти), которая предшествует *оператору* (*statement*) и может быть использована для ссылки на него.

STORAGE ASSOCIATION (связь по памяти) (14.6.3): отношение между двумя *последовательностями памяти* (*storage sequences*), когда единица памяти одной из них совпадает с единицей памяти другой.

STORAGE SEQUENCE (последовательность памяти) (14.6.3.1): последовательность смежных *единиц памяти* (*storage units*).

STORAGE UNIT (единица памяти) (14.6.3.1): *символьная единица памяти* (*character storage unit*), *числовая единица памяти* (*numeric storage unit*), либо *неспецифицированная единица памяти* (*unspecified storage unit*).

STRIDE (шаг) (6.2.2.3.1): приращение, заданное в *индексном триплете* (*subscript triplet*).

STRUCTURE (структура) (2.4.1.2): *скалярный объект данных производного типа* (*scalar data object of derived type*).

STRUCTURE COMPONENT (компонент структуры) (6.1.2): часть *объекта данных производного типа* (*data object of derived type*), соответствующая *компоненту* (*component*) в описании ее *типа* (*type*).

SUBOBJECT (подобъект) (2.4.3.1): часть *именованного объекта данных* (*named data object*), на которую можно *ссылаться* (*reference*) и которую можно *определять* (*define*) независимо от других частей. Это может быть *элемент массива* (*array element*), *секция массива* (*array section*), *компонент структуры* (*structure component*) или *подстрока* (*substring*).

SUBOBJECT DESIGNATOR (обозначение подобъекта) (2.5.1): *имя* (*name*), за которым следует один или несколько *селекторов компонента* (*component selectors*), *селекторов секции массива* (*array section selectors*), *селекторов элементов массива* (*array element selectors*) или *селекторов подстроки* (*substring selectors*).

SUBPROGRAM (процедура Фортрана) (2.2): *функция* (*function*) или *подпрограмма* (*subroutine*). Заметим, что в ФОРТРАНе 77 к этой категории относилась и *программная единица - блок данных* (*block data program unit*).

SUBROUTINE (подпрограмма) (2.2.3): *процедура* (*procedure*), *вызываемая* (*invoked*) при помощи *оператора* (*statement*) CALL или *оператора определяемого присваивания* (*defined assignment statement*).

SUBROUTINE SUBPROGRAM (процедура-подпрограмма) (12.5.2.3): последовательность *операторов* (*statements*), начинающаяся оператором SUBROUTINE, не входящим в *блок интерфейса* (*interface block*), и заканчивающаяся соответствующим оператором END.

SUBSCRIPT (индекс) (6.2.2): одно из *скалярных* (*scalar*) целых *выражений* (*expressions*), образующих список в *селекторе элемента массива* (*array element selector*). Заметим, что в ФОРТРАНе 77 весь список назывался индексом.

SUBSCRIPT TRIPLET (индексный триплет) (6.2.2): Элемент из списка в *селекторе секции массива* (*array section selector*), который содержит двоеточие и специфицирует регулярную последовательность целых значений.

SUBSTRING (подстрока) (6.1.1): непрерывная часть *скалярной символьной строки* (*scalar character string*). Заметим, что *секция массива* (*array section*) может включать *селектор подстроки* (*substring selector*); результат называется секцией массива, а не подстрокой.

TARGET (адресат) (5.1.2.8): *именованный объект данных* (*named data object*), специфицированный в операторе TARGET или в *операторе объявления типа* (*type declaration statement*), содержащем *атрибут* (*attribute*) TARGET, или объект данных, созданный оператором ALLOCATE для указателя, либо *подобъект* (*subobject*) такого объекта.

TRANSFORMATIONAL FUNCTION (функция преобразования): *встроенная функция* (*intrinsic function*), которая не является ни *поэлементной функцией* (*elemental function*), ни *справочной функцией* (*inquire function*). Обычно она имеет *аргументы-массивы* (*array arguments*) и результат-массив, значения элементов которого зависят от значений многих элементов аргументов.

TYPE (тип) (4): См. *DATA TYPE*.

TYPE DECLARATION STATEMENT (оператор объявления типа) (5): *оператор* (*statement*) INTEGER, REAL, DOUBLE PRECISION, COMPLEX, CHARACTER, LOGICAL или TYPE (*имя типа*).

TYPE PARAMETER (параметр типа) (2.4.1.1): параметр *встроенного типа данных* (*intrinsic data type*). KIND= и LEN= являются параметрами типа.

TYPE PARAMETER VALUES (значения параметра типа) (4.3): значения *параметров типа* (*type parameters*) для *категорий данных встроенного типа* (*data entity of intrinsic type*).

ULTIMATE COMPONENT (конечный компонент) (4.4): для *производного типа* (*derived type*), или *структуры* (*structure*) - *компонент* (*component*), который имеет встроенный тип или атрибут POINTER, либо *конечный компонент* (*ultimate component*) компонента, который имеет производный тип и не имеет атрибута POINTER.

UNDEFINED (не определено) (2.5.4): статус *объекта данных* (*data object*), не имеющего определенного значения.

UNSPECIFIED STORAGE UNIT (неспецифицированная единица памяти) (14.6.3.1): единица памяти для хранения *указателя* (*pointer*) или *скаляра* (*scalar*) - не указателя, *тип* (*type*) которого не является стандартным целым, стандартным символьным, стандартным вещественным, стандартным логическим или стандартным комплексным.

USE ASSOCIATION (связь по использованию) (14.6.1.2): связь *имен* (*names*) в разных *контекстных областях* (*scoping units*), специфицированная *оператором* (*statement*) USE.

VARIABLE (переменная) (2.4.3.1.1): *объект данных* (*data object*), значение которого может быть *определено* (*defined*) или переопределено во время исполнения *выполняемой программы* (*executable program*). Это может быть именованный (*named*) объект данных, *элемент массива* (*array element*), *секция массива* (*array section*), *компонент структуры* (*structure component*) или *подстрока* (*substring*). Заметим, что в ФОРТРАНе 77 переменная всегда была *скаляром* (*scalar*) и имела имя.

VECTOR SUBSCRIPT (векторный индекс) (6.2.2.3.2): *индекс секции* (*section subscript*), который является целым *выражением* (*expression*) *размерности* (*rank*) 1.

WHOLE ARRAY (массив целиком) (6.2.1): *именованный массив* (*named array*).

ИЗДАТЕЛЬСТВО "ФИНАНСЫ И СТАТИСТИКА"

предлагает справочные издания

Э.А. Якубайтис

"ИНФОРМАЦИОННЫЕ СЕТИ И СИСТЕМЫ"

368 с.

Книга освещает основные вопросы информатики, телеобработки, информационных сетей и вычислительной техники. Книга построена на базе модели, именуемой гиперсредой. Благодаря этому читатель может всесторонне изучить выбранные *им темы на основе того материала*, который предоставляет книга. Приведено много рисунков, которые усиливают информативность книги.

В.Г. Лазарев

"ИНТЕЛЛЕКТУАЛЬНЫЕ ЦИФРОВЫЕ СЕТИ"

224 с.

Приведены сведения о современных и перспективных цифровых сетях интегрального обслуживания, использовании элементов искусственного интеллекта для новейшей технологии обслуживания запросов пользователей сетей, на базе которых могут создаваться учрежденческие и корпоративные сети.

Книги Вы можете приобрести в издательстве по адресу:
101000, Москва, ул. Покровка, 7
(метро "Китай-город", выход на ул. Маросейка)
Тел.: (095) **925-35-02, 923-80-42**
Факс (095) **925-09-57**

Отдел **"Книга-почтой" – 191186, СПБ, Невский пр-т, 28**
Тел.: (812) **219-63-01**, Факс (812) **311-92-34**

Приложение В
(информационное)

УСТАРЕВШИЕ СРЕДСТВА

В.1. Исключенные средства

К исключенным относятся те средства ФОРТРАНа 77, которые являются избыточными и широко не используются. Природа исключенных средств описана в разделе 1.6.1. Список этих средств в данном Международном стандарте пуст.

В.2. Отживающие средства

К отживающим относятся те средства ФОРТРАНа 77, которые являются избыточными и для которых в ФОРТРАНе 77 доступны лучшие методы. Природа отживающих средств описана в разделе 1.6.2. К ним относятся:

1) арифметический IF - используйте оператор IF (8.1.2.4) или конструкцию IF (8.1.2);
2) вещественные и двойной точности управляющие переменные и выражения в циклах DO - используйте целые (8.1.4.1);
3) общее завершение цикла или оператор окончания цикла, отличный от END DO и CONTINUE - используйте оператор END DO или CONTINUE для каждого оператора DO;
4) передача управления извне на оператор END IF - переходите на оператор, следующий за END IF;
5) альтернативный возврат - см. В.2.1;
6) оператор PAUSE - см. В.2.2;
7) операторы ASSIGN и GO TO по предписанию - см. В.2.3;
8) присваиваемые спецификаторы формата - см. В.2.4;
9) описатель редактирования dH - см.В.2.5.

В.2.1. Альтернативный возврат

Альтернативный возврат вводит метки в список аргументов, что позволяет вызванной процедуре управлять исполнением вызывающей процедуры после возврата. Того же эффекта можно достичь, используя код возврата в вычисляемом GO TO или в конструкции CASE при возврате. Это позволяет избежать нерегулярности в синтаксисе и семантике связи аргументов. Например:

```
CALL SUBR_NAME (X, Y, Z, *100, *200, *300)
```

можно заменить на

```
CALL SUBR_NAME (X, Y, Z, RETURN_CODE)
SELECT CASE (RETURN_CODE)
    CASE (1)
    ...
    CASE (2)
    ...
    CASE (3)
    ...
    CASE DEFAULT
    ...
END SELECT
```

B.2.2. Оператор PAUSE

Выполнение оператора PAUSE требует операторского или системно-специфического вмешательства для возобновления исполнения. В большинстве случаев этой же возможности (но более эффективно и более мобильным способом) можно достичь путем использования подходящего оператора READ, ожидающего ввода некоторых данных.

B.2.3. Операторы ASSIGN и GO TO по предписанию

Оператор ASSIGN позволяет динамически присваивать целой переменной метку, а оператор GO TO по предписанию обеспечивает "непрямое ветвление" с использованием этой переменной. Это ухудшает читабельность программы, особенно если целая переменная используется также в арифметических операциях. Два совершенно различных использования целой переменной могут стать скрытым источником ошибки.

Эти операторы обычно использовались для моделирования внутренних процедур, которые сейчас можно программировать прямо.

B.2.4. Присваиваемые спецификаторы формата

Оператор ASSIGN позволял также динамически присваивать метку оператора FORMAT целой переменной, которую потом можно было использовать как спецификатор формата в операторах READ, WRITE или PRINT. Это ухудшает читабельность, допускает несогласованное использование целой переменной и может быть скрытым источником ошибки.

Эта же возможность реализуется через символьные переменные, массивы и константы.

B.2.5. Описатель редактирования H

Этот описатель редактирования может быть источником ошибки. Та же возможность достижима при использовании описателя редактирования - символьная константа.

Приложение С
(информационное)

ПРИМЕЧАНИЯ К ГЛАВАМ

С.1. Примечания к главе 1

С.1.1. Соответствие стандарту (1.4)

Стандарт требует, чтобы соответствующий стандарту процессор был способен выявлять и сообщать об использовании в программной единице форм, обозначенных как исключенные, устаревшие и нерекомендуемые, и об использовании дополнительных форм и отношений там, где такое использование можно выявить ссылкой на пронумерованные синтаксические правила и связанные с ними ограничения. Рекомендуется сопровождать процессор документацией, в которой были бы оговорены ограничения на размер и сложность обрабатываемой им программы и раз'яснен смысл сообщений, если эти ограничения превышены, определялись бы допускаемые процессором дополнительные формы и отношения и смысл сообщений по поводу использования дополнительных форм и отношений, а также исключенных, устаревших и нерекомендуемых форм. Заметим, что в этом контексте использование исключенной формы является использованием дополнительной формы.

Рекомендуется, чтобы процессор сопровождался документацией, описывающей процессорно-зависимые средства и их семантику.

С.2. Примечания к главе 2

С.2.1. Ключевые слова

Ключевые слова аргумента могут сделать ссылку на процедуру более удобочитаемой и позволяют располагать фактические аргументы в любом порядке. Эта последняя возможность допускает необязательные аргументы (2.5.2).

С.3. Примечания к главе 3

С.3.1. Представимые символы (3.1.5)

ФОРТРАН 77 разрешал появление в символьном контексте любых символов. Данный стандарт предоставляет новую возможность - исходная программа может содержать символы нескольких видов (4.3.2.1). Символы разных видов часто идентифицируются с помощью управляющих символов (называемых "shift" или "escape"-символами). Очень трудно, иногда и невозможно обрабатывать, редактировать или печатать файлы, где, например, управляющие символы могут не иметь своего первоначального смысла (как в ФОРТРАНе 77) и где другие вхождения этих же символов могут иметь управляющие функции. Чтобы обеспечить совместимость с ФОРТРАНом 77 и позволить данному стандарту обеспечить повышение мобильности, предлагается следующий подход:

1) в фиксированной форме исходного текста определение *представимого-символа* не меняется;
2) в свободной форме исходного текста управляющие символы в символьных контекстах запрещены.

С.3.2. Строки комментария (3.3.1.1, 3.3.2.1)

Стандарт не ограничивает число последовательных строк комментария. Ограничение на число строк продолжения, разрешенных для оператора, не истолковывается как ограничение на число последовательных строк комментария.

С.3.3. Метки оператора (3.2.5)

Существует 99999 уникальных меток операторов, и процессор должен воспринимать любую из них как метку оператора. Однако процессор может иметь ограничение на общее число меток операторов в одной программной единице.

С.3.4. Форма исходного текста (3.3)

В фиксированной форме исходного текста восклицательный знак (!) в символьной позиции 6 интерпретируется как признак продолжения, если только он не находится внутри комментария, обозначенного символом "C" или "*" в символьной позиции 1 либо другим символом "!" в символьных позициях 1-5 (3.3.2.3).

Форма исходного текста в ФОРТРАНе 77, ФОРТРАНе 66 и самом первом Фортране 1954 года была обусловлена обычной формой исходного текста, 80-колонными перфокартами. Однако, начиная с IBM 704, стали использоваться только 72 колонки, а оставшиеся восемь воспринимались как комментарии. В некоторых реализациях ФОРТРАНа 77 эти колонки тоже используются. Они содержат "номера строк" и используются редактором для управления изменениями в программе (3.3.2).

Подкомитет по стандартам Фортрана считает, что 66 позиций неудобны для адекватного представления читабельного исходного текста на Фортране, особенно с "длинными" именами и отступами. В результате данный стандарт ослабил ограничения ФОРТРАНа 77 на размер исходной строки.

Необходимость введения несовместимой новой формы исходного текста ослабила остальные ограничения на жесткую форму карты. Позиции 6 и 7 больше не "специальные", а признак продолжения находится на продолжаемой строке, а не на строке продолжения. Символы пробела в общем случае являются в новой форме исходного текста значащими, но остальные свойства новой формы относятся к любой из форм исходного текста и разрешены для любой из них (3.3.1).

Правило, разрешающее пробелы в некоторых ключевых словах, предназначено для того, чтобы дать пользователю, привыкшему к незначащим пробелам, разумный выбор.

В некоторых случаях, например, если код содержится в файле INCLUDE для использования его в программах с обеими формами исходного текста, такой код можно использовать с любой из форм исходного текста при условии соблюдения приведенных ниже правил:

1) помещайте метку оператора в позиции 1–5, а оператор - в позиции 7–72;
2) относитесь к пробелам, как к значащим;
3) для комментария используйте только восклицательный знак, но не помещайте его в позицию 6;
4) если оператор надо продолжить, помещайте амперсанд (&) в позицию 73 продолжаемой строки и в позицию 6 строки продолжения.

С.4. Примечания к главе 4

С.4.1. Нуль (4.3.1)

Процессор не должен отличать отрицательный нуль от положительного.

С.4.2. Символы (4.2)

Свободная форма исходного текста позволяет считать представимыми только графические символы. Почти все управляющие символы имеют применения или назначения, что препятствует их использованию в символьных литералах. Тем не менее для совместимости с ФОРТРАНом 77 использование управляющих символов в фиксированной форме исходного текста, в основном, разрешено.

С.4.3. Встроенные и производные типы данных (4.3, 4.4)

ФОРТРАН 77 поддерживал только явно определенные в стандарте типы данных (логический, целый, вещественный, двойной точности, комплексный и символьный). Данный стандарт разрешает все эти встроенные типы и разрешает производные типы, позволяющие создавать новые типы данных. Описание производного типа специфицирует структуру данных, состоящую из компонентов встроенного или другого производного типа. Такое описание типа представляет не объекты данных, а, скорее, шаблон для объявленных с этим типом именованных объектов. Например, описание

```
TYPE POINT
    INTEGER X_COORD
    INTEGER Y_COORD
END TYPE POINT
```

определяет новый производный тип с именем POINT, образованный из двух компонентов встроенного целого типа (X_COORD и Y_COORD). Оператор TYPE (POINT) FIRST, LAST объявляет два объекта данных, FIRST и LAST, которые могут получать значения типа POINT.

ФОРТРАН 77 поддерживал типы REAL и DOUBLE PRECISION как аппроксимацию математических вещественных чисел. Данный стандарт обобщает REAL как встроенный тип с параметром, который и выбирает способ аппроксимации. Параметр типа называется видом, а значения его зависят от процессора. DOUBLE PRECISION трактуется как синоним REAL(*k*), где *k* - определяемое реализацией значение параметра типа KIND(0.0D0).

Вещественные литеральные константы могут быть специфицированы с параметром вида, этим обеспечивается конкретное значение параметра вида для константы. Например, при наличии объявлений

```
INTEGER Q
PARAMETER (Q=8)
REAL (Q) B
```

литеральная константа 10.93_Q имеет ту же точность, что переменная B. ФОРТРАН 77 запрещал символьные строки нулевой длины. В данном стандарте они разрешены (4.3.2.1).

Объекты относятся к разным производным типам, если они описаны с использованием разных описаний производного типа. Например:

```
TYPE APPLES
    INTEGER NUMBER
END TYPE APPLES

TYPE ORANGES
    INTEGER NUMBER
END TYPE ORANGES

TYPE(APPLES) COUNT1
TYPE(ORANGES)COUNT2
COUNT1=COUNT2                ! Неверное присваивание, смешивание типов
```

Несмотря на то, что все компоненты объектов типа APPLES и объектов типа ORANGES имеют идентичные встроенные типы, объекты относятся к разным типам.

С.4.4 Выбор способов аппроксимации

Данный стандарт позволяет выбрать способ аппроксимации вещественных для всей программы с использованием параметризованного вещественного типа и модулей. Это можно сделать, определив именованную константу, скажем, FLOAT, имеющую значение конкретного параметра вида, и используя потом эту именованную константу во всех объявлениях вещественных, комплексных и производных типов. Например, операторы спецификации

```
INTEGER FLOAT
PARAMETER (FLOAT=8)
REAL(FLOAT) X,Y
COMPLEX (FLOAT) Z
```

определяют, что способ аппроксимации, соответствующий значению параметра вида 8, относится в программной единице к объектам данных X, Y и Z. Значение параметра вида FLOAT можно сделать доступным для всей программы, поместив операторы INTEGER и PARAMETER в модуль и ссылаясь на именованную константу FLOAT с помощью оператора USE. Заметим, что заменив один раз в модуле 8 на 4, можно выбрать другой способ аппроксимации.

Чтобы избежать процессорно-зависимых значений 4 или 8, можно заменить 8 на KIND(0.0) или KIND(0.0D0). Другой способ избежать этих процессорно-зависимых значений - выбрать значение вида, используя встроенную справочную функцию SELECTED_REAL_KIND. Эта функция с целыми аргументами P и R, специфицирующими минимальные требования к десятичной точности и диапазону десятичного порядка, выдает значение параметра вида для способа аппроксимации, который имеет, как минимум, P десятичных цифр точности и диапазон положительных чисел от 10**(-R) до 10**R. В приведенных выше операторах можно, например, заменить 8 на SELECTED_REAL_KIND(10,50), что приведет к выбору способа аппроксимации с точностью, как минимум, 10 десятичных цифр и диапазоном порядка от 10**(-50) до 10**50.

В языке нет никаких ограничений на величину и упорядоченность значений видов, для того, чтобы реализаторы имели свободу в выборе таких значений и в добавлении новых видов без изменения значений старых.

Поскольку значения видов не имеют мобильного смысла, целесообразно использовать их в программе только через именованные константы, как описано выше (например, SINGLE, IEEE_SINGLE, DOUBLE и QUAD), а не использовать значения параметров вида непосредственно.

С.4.5. Память для производных типов (4.4.1)

Структура разворачивается в последовательность компонентов встроенного типа. Если она не содержит оператора SEQUENCE, использование термина "структура" никоим образом не подразумевает какой-либо упорядоченности в расположении ее компонентов в памяти. Нет и каких-либо требований по использованию непрерывной памяти. Последовательность только указывает на тот факт, что описания должны быть записаны в таком же порядке, в котором появляются компоненты, и этот порядок и будет определять последовательность компонентов. Важность этого порядка ограничена, т.к. к компоненту объекта производного типа всегда обращаются через имя компонента, кроме следующих контекстов: последовательность выражений в конструкторе значения производного типа, значения данных при вводе по списку NAMELIST, включение структуры в список ввода/вывода при форматной передаче данных. В этих случаях понятие порядка элементов распространяется и на соответствующую последовательность компонентов. При условии, что в этих случаях обеспечивается определенный порядок, в остальных - процессор может организовывать хранение в памяти компонентов любой структуры таким образом, чтобы оно наиболее соответствовало конкретной архитектуре.

С.4.6. Указатели

Данный стандарт вводит указатели как имена, которые могут динамически менять свои связи с объектами-адресатами. В этом смысле обычная переменная - это имя, имеющее фиксированную связь с конкретным объектом. Имя обычной переменной ссылается на одно и то же пространство памяти на протяжении всего времени жизни переменной. Имя указателя может в разные моменты времени ссылаться на разные участки памяти и даже не ссылаться ни на один участок памяти. Переменную можно рассматривать как дескриптор для пространства, где хранятся значения с соответствующим типом, параметрами типа и размерности массива, и значения, заносимые в дескриптор, фиксируются в момент создания переменной при ее

объявлении. Указатель тоже можно рассматривать как дескриптор, но такой, значения которого могут динамически изменяться, описывая разные участки памяти. При описании указателя отводится место для хранения дескриптора, но для объекта-адресата память не отводится.

Производный тип может иметь один или несколько компонентов, определенных как указатели. Этот компонент может быть указателем на объект того же производного типа. Это "рекурсивное" определение данных позволяет конструировать динамические структуры данных, такие, как связанные списки, деревья, графы. Например:

```
TYPE CELL                                  ! Определение "рекурсивного" типа
    INTEGER :: VAL
    TYPE (CELL), POINTER :: NEXT_CELL
END TYPE CELL

TYPE (CELL), TARGET :: HEAD
TYPE (CELL), POINTER :: CURRENT, TEMP      ! Объявление указателей
INTEGER :: IOEM, K

CURRENT => HEAD                            ! CURRENT указывает на голову списка
NULLIFY (CURRENT%NEXT_CELL))


DO
    READ (*,*,IOSTAT=IOEM) K               ! Чтение следующего значения,
                                           ! если оно есть
    IF (IOEM /= 0) EXIT
    ALLOCATE (TEMP)                        ! Создание новой ячейки при
                                           ! каждой итерации
    TEMP % VAL = K                         ! Присваивание ей значения
    NULLIFY (TEMP%NEXT_CELL)               ! Установить статус как
                                           ! "несвязанный"
    CURRENT%NEXT_CELL=>TEMP                ! Присоединить новую ячейку
                                           ! к списку
    CURRENT => TEMP                        ! CURRENT указывает на новый
                                           ! конец списка
END DO
```

Список сконструирован, и последняя связанная ячейка содержит несвязанный указатель. Для "просмотра" списка можно использовать цикл

```
CURRENT => HEAD
DO
    IF (.NOT. ASSOCIATED (CURRENT % NEXT_CELL)) EXIT
    CURRENT => CURRENT % NEXT_CELL
    WRITE (*,*) CURRENT % VAL
END DO
```

С.5. Примечания к главе 5

С.5.1. Операторы объявления типа (5.1)

Операторы объявления типа в ФОРТРАНе 77 были организованы так, что для объявления разных атрибутов объектов требовались разные операторы (INTEGER, SAVE, DATA и т.д.). Данный стандарт позволяет объявлять атрибуты объекта в одном операторе типа расширенной формы. Например:

```
INTEGER, DIMENSION (10,10), SAVE:: A, B, C
REAL, PARAMETER:: P1=3.14159265, E=2.718281828
```

Для сохранения совместимости с ФОРТРАНом 77 большая часть атрибутов, которые можно задать в расширенном операторе типа, может быть задана (альтернативно) и в отдельном операторе.

С.5.2. Атрибут POINTER (5.1.2.7)

Для объявления указателя нужно специфицировать атрибут POINTER. Тип, параметры типа и размерность, которые могут быть специфицированы в одном операторе, либо в одном или в нескольких операторах спецификации атрибутов, определяют характеристики объектов-адресатов, которые могут быть связаны с указателями, объявленными в операторе. Очевидная модель интерпретации объявления указателей: такие объявления создают для каждого имени дескриптор, который включает в себя все данные, необходимые для того, чтобы полностью описать и разместить в памяти объект и все подобъекты с заданными типом, параметрами типа и размерностью. При создании дескриптор пуст, в нем нет информации о том, как обратиться к фактическому участку памяти. Эти значения дескриптора будут заполнены, когда указатель будет связан с фактической памятью адресата.

Приведенный ниже пример иллюстрирует использование указателей в итеративном алгоритме:

```
PROGRAM DYNAM_ITER
  REAL,DIMENSION (:,:),POINTER::A,B,SWAP     ! Объявление указателей
  READ (*,*) N, M
  ALLOCATE (A(N,M),B(N,M))          ! Размещение массивов-адресатов
  !       Чтение значений в A
  . . .

  ITER: DO
    . . .
    !     Преобразование значений A для получения значений в B
    . . .
    IF (CONVERGED) EXIT ITER
    !     Поменять местами A и B
    SWAP => A; A => B; B => SWAP
  END DO ITER
  . . .
END
```

С.5.3. Атрибут TARGET (5.1.2.8)

Атрибут TARGET должен быть специфицирован для любого объекта (не указателя), который во время выполнения программы может быть связан с указателем. Этот атрибут определен исключительно в целях оптимизации. Это дает возможность процессору полагать, что для любого объекта, не являющегося указателем и не объявленного явно адресатом, единственным способом ссылки является то имя, с которым он объявлен. Правило из 5.1.2.8 обеспечивает, чтобы это было верно, даже если объект - из общего блока и соответствующий объект из этого же общего блока в другой программной единице имеет атрибут TARGET. Это также означает, что неявно объявленные объекты нельзя использовать как адресаты для указателей. Это позволит процессору выполнять оптимизации, которые были бы невозможны при наличии некоторых указателей.

Приведенный ниже пример иллюстрирует использование атрибута TARGET в итеративном алгоритме:

```
PROGRAM ITER
    REAL, DIMENSION (1000, 1000),TARGET :: A, B
    REAL, DIMENSION (:, :), POINTER     :: IN, OUT, SWAP
    . . .
```

```
                    ! Чтение значений в A
    . . .
    IN => A         ! IN связывается с адресатом A
    OUT => B        ! OUT связывается с адресатом B
    . . .
    ITER: DO
        . . .
                    ! Преобразование значений IN для получения OUT
        . . .
        IF (CONVERGED) EXIT ITER
                    ! Перестановка значений IN и OUT
        SWAP => IN; IN => OUT; OUT => SWAP
    END DO ITER
    . . .
END
```

C.5.4. Операторы PARAMETER и IMPLICIT NONE (5.2.10, 5.3)

Поскольку неявно типизированная именованная константа может предшествовать оператору IMPLICIT только в том случае, если этот оператор IMPLICIT не входит в конфликт с типом именованной константы, то отсюда следует, что если есть оператор IMPLICIT NONE, он должен предшествовать всем операторам PARAMETER.

C.5.5. Расширения оператора EQUIVALENCE (5.5.1)

Оператор EQUIVALENCE расширен, чтобы разрешить эквивалентность для последовательных структур и объектов нестандартного встроенного типа, но при этом имеются правила, ограничивающие появление таких объектов в операторе EQUIVALENCE.

Структуры, входящие в оператор EQUIVALENCE, должны быть последовательными структурами. Если последовательная структура не относится к числовому последовательному типу или к символьному последовательному типу, она может быть эквивалентна только объектам одинакового с ней типа.

Числовая последовательная структура может быть эквивалентна другой числовой последовательной структуре, объекту стандартного целого, стандартного вещественного, вещественного двойной точности, стандартного комплексного или стандартного логического типа таким образом, чтобы каждый из компонентов такой структуры в конце концов был связан только с объектами этих типов.

Символьная последовательная структура может быть эквивалентна только объекту стандартного символьного типа или другой символьной последовательной структуре.

Все остальные объекты могут быть эквивалентны только объектам того же типа и с теми же параметрами типа.

C.5.6. Расширения оператора COMMON (5.5.2)

Модули обеспечивают глобальный доступ ко всем объектам, однако и оператор COMMON тоже был расширен, чтобы разрешить в более чем одной контекстной области доступ к объектам с атрибутом POINTER, к последовательным структурам и к объектам нестандартных встроенных типов.

Допускается, чтобы общий блок содержал последовательности из разных единиц памяти при условии, что каждая контекстная область, которая ссылается на этот общий блок, задает для него идентичную последовательность единиц памяти. Это расширение позволяет одному общему блоку включать и числовые, и символьные единицы памяти.

Связь между объектами стандартных встроенных типов, вещественного двойной точности типа и последовательных структур из разных контекстных областей разрешена в соответствии с теми же правилами, что и для эквивалентных объектов (5.5.1).

С.6. Примечания к главе 6

С.6.1. Подстроки (6.1.1)

Длина подстроки считается нулевой, когда значение начальной точки превышает значение конечной. Это не разрешалось в ФОРТРАНе 77. Кроме того, данный стандарт допускает подстроки символьных литеральных констант и именованных констант.

С.6.2. Ссылки на элементы массива (6.2.2)

Ссылка на элемент с индексом, расположенным вне описанных границ, не соответствует стандарту, как и в ФОРТРАНе 77.

С.6.3. Компоненты структуры (6.1.2)

На компонент структуры ссылаются, записывая компоненты всех последовательных уровней до тех пор, пока не будет описан нужный элемент. Например:

```
TYPE ID_NUMBERS
    INTEGER SSN
    INTEGER EMPLOYEE_NUMBER
END TYPE ID_NUMBERS

TYPE PERSON_ID
    CHARACTER (LEN=30) LAST_NAME
    CHARACTER (LEN=1) MIDDLE_INITIAL
    CHARACTER (LEN=30) FIRST_NAME
    TYPE (ID_NUMBERS) NUMBER
END TYPE PERSON_ID

TYPE PERSON
    INTEGER AGE
    TYPE (PERSON_ID) ID
END TYPE PERSON

TYPE(PERSON) GEORGE, MARY
    PRINT *, GEORGE % AGE                  ! Печать компонента AGE
    PRINT *, MARY % ID % LAST_NAME         ! Печать LAST_NAME для MARY
    PRINT *, MARY % ID % NUMBER % SSN      ! Печать SSN для MARY
    PRINT *, GEORGE % ID % NUMBER          ! Печать SSN и
                                           ! EMPLOYEE_NUMBER для GEORGE
```

Компонент структуры может быть объектом данных встроенного типа, как в случае GEORGE % AGE, либо производного типа, как в случае GEORGE % ID % NUMBER. Результирующий компонент может быть скаляром или массивом встроенного или производного типа.

```
TYPE LARGE
    INTEGER ELT(10)
    INTEGER VAL
END TYPE LARGE

TYPE (LARGE) A(5)          ! 5 элементов массива, каждый из которых
                           !     содержит 10 элементов массива ELT и скаляр VAL
PRINT *, A(1)              ! Печатает 10 элементов массива ELT и
                           !     скаляр VAL
```

```
PRINT *,A(1)%ELT(3)        ! Печатает скалярный элемент 3 из
                           !     первого элемента A
PRINT *,A(2:4)%VAL         ! Печатает скаляр VAL для элементов 2-4 массива A
```

C.6.4. Размещение и связывание указателей

Действие операторов ALLOCATE, DEALLOCATE, NULLIFY и оператора присваивания указателю интерпретируется как изменение значений в дескрипторе, который является указателем. ALLOCATE подразумевает отведение памяти для соответствующего объекта и "присваивание" указателю значений, необходимых для описания этой области памяти. NULLIFY разрывает связь указателя с областью памяти. DEALLOCATE разрывает связь и освобождает память. В зависимости от реализации это может выглядеть как установка флага в указателе, который показывает, правильны ли значения в дескрипторе, либо как очистка дескрипторных значений с помощью установки некоторых (скажем, нулевых) значений, означающих, что указатель ни на что в данный момент не указывает. Присваивание указателю выполняет копирование значений, необходимых для описания области памяти, занимаемой адресатом, в дескриптор, который является указателем. Дескрипторы копируются, значения объектов - нет.

Если PA и PB - указатели и PB в данный момент связан с адресатом, тогда

```
PA => PB
```

связывает PA с тем же адресатом, что PB. Если PB не связан, PA тоже становится не связанным.

В стандарте специфицировано, что эта связь является прямой и независимой. Оператор

```
PB => D
```

или

```
ALLOCATE (PB)
```

не влияет на связь PA с его адресатом. Оператор

```
DEALLOCATE (PB)
```

превращает PA в "подвешенный указатель" на область памяти, которая была освобождена. Программа не должна использовать PA до тех пор, пока он не станет связанным через присваивание указателю или с помощью оператора ALLOCATE.

DEALLOCATE можно использовать для освобождения только такой области памяти, которая была создана предварительным выполнением оператора ALLOCATE. Таким образом, следующее неверно:

```
REAL, TARGET :: T
REAL, POINTER :: P
. . .
P => T
DEALLOCATE (P)             ! Нельзя: адресат для P не размещен
```

Основным принципом является следующее: ALLOCATE, NULLIFY и присваивание указателю касаются указателя, а не адресата. ALLOCATE создает новый адресат, но на старый влияет только разрывом его связи с конкретным указателем. Ни оператор NULLIFY, ни присваивание указателю никак не влияют на адресатов. Данный участок памяти, который был отведен и связан с указателем, становится недоступным для программы, если указатель аннулирован и никакой другой указатель не был связан с этим участком памяти. Эти участки памяти могут быть переиспользованы процессором, если это целесообразно. Однако это полностью зависит от процессора.

С.7. Примечания к главе 7

С.7.1. Символьное присваивание

Снято ограничение ФОРТРАНа 77 относительно того, что ни на одну из символьных позиций, определяемых оператором символьного присваивания, нельзя ссылаться в выражении (7.5.1.5).

С.7.2. Вычисление ссылок на функции

Если в выражении есть несколько ссылок на функции, их можно вычислять в любом порядке (при условии, что результат функции вычисляется после вычисления ее аргументов), и их значения не должны зависеть от порядка вычисления. Такое отсутствие зависимости от порядка вычисления позволяет ссылки на функции выполнять параллельно (7.1.7.1).

С.7.3. Указатели в выражениях

Когда указатель используется в качестве первичного в выражении, он, в основном, рассматривается аналогично всем другим переменным. Если он используется как операнд в операции, требующей значения, указатель автоматически выдает значение, размещенное в том участке памяти, который в данный момент описан указателем, т.е. значение объекта-адресата, в данный момент связанного с указателем. В контекстах выражений, запрашивающих значение, указатели разыменовываются.

С.7.4. Указатели в левой части присваивания

Указатель, который находится слева в операторе встроенного присваивания, тоже разыменовывается и берется как ссылка на ту область памяти, которая является его текущим адресатом. Таким образом, оператор присваивания задает обычное копирование значения выражения из правой части в область памяти этого адресата. Сохраняются все обычные правила встроенного присваивания. Тип, параметры типа и конфигурация выражения и адресата указателя должны согласовываться.

При встроенном присваивании производных типов для компонентов-не указателей действует обычное присваивание, для компонентов-указателей - присваивание указателям. Разыменование применяется только к полностью скалярным объектам, а не выборочно к подобъектам указателя.

Например, предположим, что описан такой тип:

```
TYPE CELL
    INTEGER :: VAL
    TYPE (CELL), POINTER :: NEXT_CELL
END TYPE
```

и объявлены объекты:

```
TYPE (CELL), TARGET  :: HEAD
TYPE (CELL), POINTER :: CURRENT
```

Если создан связанный список и присоединен к HEAD, а указателю CURRENT отведена область памяти, тогда операторы

```
CURRENT = HEAD
CURRENT = CURRENT % NEXT_CELL
```

вызовут копирование содержимого ячеек, на которые ссылаются правые части, в ячейку, на которую ссылается CURRENT. В частности, правая часть второго оператора выбирает компонент-указатель в ячейке CURRENT. Этот указатель, поскольку он находится в

контексте выражения, разыменовывается для получения целого значения адресата и указателя на ячейку, которая содержится в компоненте NEXT_CELL адресата. Левая часть оператора вызывает разыменование указателя CURRENT для получения его адресата на данный момент, а именно, участка памяти для хранения ячейки (целого и указателя ячейки). Целое значение из правой части копируется в область памяти для целого из левой части, а для компонентов-указателей выполняется присваивание указателям (дескриптор из правой части копируется в память для дескриптора из левой части). Когда выполняется оператор

```
CURRENT => CURRENT % NEXT_CELL
```

значение дескриптора в CURRENT%NEXT_CELL копируется в дескриптор с именем CURRENT. В этом случае CURRENT указывает на другого адресата. Во встроенном операторе присваивания область памяти, связанная с текущим указателем, не меняется, а хранимые в ней значения меняются. В присваивании указателю текущий указатель связывается с другим участком памяти. Использование встроенного присваивания будет прогонять связанный список ячеек через текущее "окно", присваивание указателю вызовет перемещение текущего указателя вниз по списку ячеек.

С.8. Примечания к главе 8

С.8.1. Управление циклом

Фортран обеспечивает несколько форм управления циклом:

1) со счетчиком числа итераций и DO-переменной (это классический цикл Фортрана);
2) проверка логического условия перед каждым исполнением цикла (DO WHILE);
3) "бесконечный" цикл.

С.8.2. Конструкция CASE

В конструкции CASE выполняется максимум один блок и нет "проваливания" из одного блока в другой. Таким образом, пользователю не обязательно организовывать явный выход из блока.

С.8.3. Примеры неверных конструкций DO

Пример 1:

```
DO I=1,10
    ...
END DO LOOP                  ! Нет соответствующего имени конструкции
```

Пример 2:

```
LOOP:DO 1000 I=1,10          ! Нет соответствующего имени конструкции
    ...
1000 CONTINUE
```

Пример 3:

```
LOOP1: DO
    ...
END DO LOOP2                 ! Несоответствие имен конструкции
```

Пример 4:

```
    DO I=1,10                ! Требуется метка или ...
    ...
1010 CONTINUE                !     ...требуется END DO
```

Пример 5:

```
    DO 1020 I=1,10
   ...
1021 END DO                    ! Несоответствие метки
```

Пример 6:

```
FIRST: DO I=1,10
    SECOND: DO J=1,5
 ...
END DO FIRST                   ! Неверная вложенность циклов
    END DO SECOND
```

C.9. Примечания к главе 9

C.9.1. Записи ввода/вывода (9.1)

То, что в Фортране называется "записью", обычно называют "логической записью". В Фортране нет понятия "физической записи".

Запись "конец файла" необязательно должна иметь физическое воплощение. Процессор может использовать счетчик числа записей или другой способ регистрации позиции файла в момент выполнения оператора ENDFILE и потом предпринять соответствующие действия при повторном достижении этой позиции во время операции чтения. Однако, если запись "конец файла" реализована, она считается существующей при выполнении оператора BACKSPACE (9.1.3).

C.9.2. Файлы (9.2)

Данный стандарт приспособлен к каталогизации файлов, хотя и не требует ее. Для этого вводится несколько понятий.

C.9.2.1. Подсоединение файла (9.3)

Любой файл до выполнения какого-либо ввода/вывода должен быть подсоединен к устройству. Устройство затем служит для обозначения файла на все время существования соединения. Подсоединение не предполагает, будет или не будет выделяться память под буферы, будут или не будут заполняться таблицы управления файлами и будут ли использоваться какие-либо другие способы реализации. Подсоединение означает, что (за некоторыми исключениями) оператор READ или WRITE может выполняться для устройства, а не для файла. Без подсоединения операторы READ или WRITE не могут быть выполнены.

C.9.2.2. Существование файла (9.2.1.1)

В целом независимость состояния подсоединения - это свойство существования, которое является характеристикой файла. Процессор "знает" набор файлов, которые существуют в данный момент для данной выполняемой программы. В этот набор могут входить ленты, готовые к чтению, файлы в каталоге, клавиатура, принтер и т.д. Этот набор может исключить файлы, недоступные для выполняемой программы либо по причине защиты, либо потому, что они уже используются другой выполняемой программой, и т.д. В данном стандарте не задано, какие файлы существуют, так что процессору предоставляются широкие возможности для реализации защиты, блокировок, привилегий и т.д. Существование - это удобное понятие для обозначения всех файлов, которые выполняемая программа может потенциально обрабатывать.

Могут иметь место четыре комбинации подсоединения и существования:

Подсоединение	Существование	Примеры
Да	Да	Читающее устройство, загруженное и готовое к чтению
Да	Нет	Принтер до вывода первой строки
Нет	Да	Файл с именем 'JOAN' в каталоге
Нет	Нет	Файл на бобине, неизвестной процессору

Имеются средства для создания, уничтожения, подсоединения и отсоединения файлов.

С.9.2.3. Имена файлов (9.3.4.1)

Файл может иметь имя. Форма имени файла не специфицируется. Если в системе нет каких-то форм каталогизации или маркировки лент хотя бы для некоторых ее файлов, все имена файлов исчезают при завершении выполнения. Это допустимая реализация. Нигде в данном стандарте нет требования, чтобы эти имена сохранялись какое-то время после окончания выполнения программы. Таким образом, данный стандарт не предполагает каталогизацию как предпосылку. Средство именования предназначено для того, чтобы позволить использование системы каталогизации там, где она имеется.

С.9.2.4. Доступ к файлу (9.2.1.2)

Данный стандарт не касается проблем секретности, защиты, блокировок и многих других понятий, которые могут быть частью понятия "право доступа". Считается, что эти понятия относятся к компетенции операционной системы.

Операторы OPEN и INQUIRE могут быть расширены, чтобы учесть эти вещи.

Возможные способы доступа к файлу - прямой и последовательный. Процессор может реализовать два разных типа файлов, каждый со своим собственным способом доступа. Он также может реализовать один тип файла с двумя разными способами доступа.

Прямой способ доступа прост и общедоступен, это записи фиксированной длины. Ключом является положительное целое.

С.9.2.5. Ввод/вывод без продвижения (9.2.1.3.1)

Операторы передачи данных влияют на установку позиции внешнего файла. В ФОРТРАНе 77, если не возникала ситуация ошибки или конца файла, позиция файла устанавливалась после только что считанной или записанной записи, так что эта запись становилась предшествующей записью. Данный стандарт содержит в операторе передачи данных спецификатор ADVANCE=, который дает возможность управлять позицией внутри текущей записи от одного форматного оператора передачи данных к следующему оператору передачи данных. Значение NO обеспечивает эту возможность. Значение YES устанавливает позицию файла после только что считанной или записанной записи. По умолчанию принимается значение YES.

Табуляцию и описатель редактирования "/" можно использовать с этим способом доступа к записи, но при этом табуляция не может возвращаться за левую границу табуляции.

BACKSPACE для файла, позиция которого находится внутри записи, устанавливает позицию заданного устройства перед началом текущей записи.

Если последним оператором передачи данных был WRITE и позиция файла в данный момент находится внутри записи, неявно позиция файла будет установлена после текущей записи, перед записанной в файл записью "конец файла", т.е операторы REWIND, BACKSPACE и ENDFILE, следующие после оператора WRITE без продвижения, вызывают установку позиции файла на конец текущей записи, перед записанной в файл записью "конец файла".

Данный стандарт позволяет использовать в операторах передачи данных без продвижения спецификатор SIZE=. Переменная в спецификаторе SIZE= будет содержать счетчик числа символов, образующих последовательность значений, считанных описателями редактирования в этом операторе ввода.

Счетчик особенно полезен, если в списке ввода всего один элемент, тогда он содержит число символов, которые имеются для этого элемента.

Спецификатор EOR= позволяет обнаружить ситуацию "конец записи" во время оператора передачи данных без продвижения. Эта ситуация не является ошибочной. Если этот спецификатор присутствует, текущий элемент списка ввода, потребовавший больше символов, чем содержит запись, будет дополняться пробелами, если задано PAD='YES'. Это значит, что элемент списка ввода/вывода будет благополучно заполнен. Затем позиция файла устанавливается после текущей записи. Если присутствует спецификатор IOSTAT=, он определяется отрицательным значением, зависящим от процессора, и оператор передачи данных завершается. Выполнение программы продолжится с оператора, заданного в спецификаторе EOR=. Спецификатор EOR= дает возможность осуществлять контроль за исполнением, когда будет обнаружен конец записи. При возникновении ситуации "конец записи" переменные неявных циклов сохраняют свое последнее значение, а остальные элементы *списка-вв* - свой статус определенности. Спецификатор SIZE=, если он присутствует, будет содержать число символов, считанных описателями редактирования во время этого оператора READ.

При вводе без продвижения процессор не обязан читать частичные записи. Он может считать запись целиком во внутренний буфер и сделать нужные части записи доступными для последующих операторов ввода.

С.9.3. Оператор OPEN (9.3.4)

Файл может быть подсоединен к устройству любым из двух способов: предварительным подсоединением или выполнением оператора OPEN. Предварительное подсоединение осуществляется до начала исполнения выполняемой программы средствами, внешними по отношению к Фортрану. Например, это может быть сделано через управление задачами или установлено процессором по умолчанию. Выполнение оператора OPEN не требует доступа к предварительно подсоединенным файлам (9.3.3).

Оператор OPEN дает способ доступа к существующим файлам, которые не были предварительно подсоединены. Использовать оператор OPEN можно двумя способами: с именем файла (открыть по имени) и без имени файла (открыть по устройству). Устройство задается в любом случае. Открытие по имени подсоединяет заданный файл к заданному устройству. Открытие по устройству подсоединяет определенный процессором по умолчанию файл к заданному устройству (этот файл может иметь имя, а может и не иметь).

Таким образом, имеются три способа подсоединить, а значит, и обработать файл: предварительное подсоединение, открытие по имени и открытие по устройству. Если файл однажды был подсоединен, в стандартном Фортране нет способа определить, как это было сделано.

Оператор OPEN можно также использовать для создания нового файла. Фактически любой из трех перечисленных выше способов подсоединения можно применить к файлу, который не существует. Когда устройство подсоединено предварительно, запись первой записи создает файл. В двух других способах выполнение оператора OPEN создает файл.

Когда выполняется оператор OPEN, заданное в нем устройство может быть уже подсоединено к файлу, а может и не быть подсоединено. Если оно подсоединено (либо

предварительно, либо предыдущим оператором OPEN), тогда отсутствие спецификатора FILE= в операторе OPEN означает, что файл остается подсоединенным к устройству. Такой оператор OPEN можно использовать для изменения значений спецификаторов BLANK=, DELIM= и PAD=.

Заметим, что поскольку оператор OPEN, в котором специфицирован STATUS='SCRATCH', не может содержать спецификатор FILE=, такой OPEN всегда пытается сохранить любое подсоединение, которое может иметь указанное устройство. Если устройство уже было подсоединено к файлу и это соединение не имело спецификатора STATUS='SCRATCH', оператор OPEN будет неверным, т.к. значение спецификатора STATUS= нельзя менять в операторе OPEN.

Если значение спецификатора ACTION= есть WRITE, оператор READ не должен ссылаться на это подсоединение. ACTION='WRITE' не ограничивает установку позиции файла в операторе BACKSPACE или установку позиции файла, специфицированной спецификатором POSITION= со значением APPEND. Однако оператор BACKSPACE и оператор OPEN, содержащий POSITION='APPEND', могут привести к ошибке, если процессор требует чтения файла для установки позиции файла.

Приведенные ниже примеры иллюстрируют эти правила. В первом примере устройство 10 предварительно подсоединено к SCRATCH-файлу; оператор OPEN меняет значение PAD= на YES.

```
CHARACTER (LEN = 20) CH1
WRITE (10, '(A)') 'THIS IS RECORD 1'
OPEN (UNIT = 10, STATUS = 'SCRATCH', PAD = 'YES')
REWIND 10
READ (10, '(A20)') CH1                  ! Сейчас CH1 имеет значение
                                        ! 'THIS IS RECORD 1'
```

В следующем примере устройство 12 сначала подсоединяется к файлу по имени FRED со статусом OLD. Затем второй оператор OPEN снова откроет устройство 12, сохранив его соединение с файлом FRED, но изменив значение спецификатора DELIM= на QUOTE.

```
CHARACTER (LEN = 25) CH2, CH3
OPEN (12, FILE ='FRED', STATUS = 'OLD', DELIM = 'NONE')
CH2 = '"THIS STRING HAS QUOTES."'
                                        ! Кавычки в строке CH2
WRITE (12,*) CH2                        ! Записано без ограничителей
OPEN (12, DELIM = 'QUOTE')              ! Теперь кавычки - ограничитель
REWIND 12
READ (12, *) CH3                        ! Сейчас CH3 имеет значение
                                        ! 'THIS STRING HAS QUOTES.'
```

Следующий пример неверен, т.к. он пытается изменить значение спецификатора STATUS=.

```
OPEN (10, FILE = 'FRED', STATUS = 'OLD')
WRITE (10, *) A, B, C
OPEN (10, STATUS = 'SCRATCH')           ! Попытка сделать FRED файлом
                                        ! со статусом SCRATCH
```

Предыдущий пример можно исправить, закрыв сначала устройство, как в следующем примере:

```
OPEN (10, FILE = 'FRED', STATUS = 'OLD')
WRITE (10, *) A, B, C
CLOSE (10)
OPEN (10, STATUS = 'SCRATCH')           ! Открывается другой SCRATCH-файл
```

C.9.4. Характеристики подсоединения (9.3.2)

Когда устройство подсоединяется к файлу, либо при предварительном подсоединении, либо при выполнении оператора OPEN, могут быть установлены следующие характеристики подсоединения.

1. Устанавливается способ доступа для подсоединения: прямой или последовательный (9.3.4.3).
2. Устанавливается форма, форматная или неформатная, для подсоединения к файлу, который существует или будет создан подсоединением. Если форма не задана, для подсоединения, являющегося результатом выполнения оператора OPEN, устанавливается форма по умолчанию (которая зависит от способа доступа, как это описано в 9.2.1.2). Для существующего предварительно подсоединенного файла форма устанавливается при предварительном подсоединении. Для не существующего предварительно определенного файла форма может быть установлена, либо ее установление может быть отложено до создания файла (например, при выполнении форматного или неформатного оператора WRITE) (9.3.4.4).
3. Может быть установлена длина записи. Если способ доступа - прямой, подсоединение устанавливает длину записи, которая задает длину каждой записи в файле. Существующий файл, в котором не все записи одинаковой длины, нельзя подсоединять для прямого доступа.

   Если способ доступа - последовательный, записи разной длины допускаются. В этом случае установленная длина записи задает максимальную длину записи в файле (9.3.4.5).
4. Характеристика значимости пробелов, равная NULL или ZERO, устанавливается для форматного подсоединения. Она никак не влияет на вывод. Если подсоединение является результатом выполнения оператора OPEN и интерпретация пробелов не задана, значением этой характеристики по умолчанию будет NULL. Для подсоединенного предварительно файла эта характеристика устанавливается предварительным подсоединением.

   Характеристика значимости пробелов для подсоединения важна в начале каждого оператора форматного ввода. Во время выполнения оператора любой встретившийся описатель редактирования BN или BZ может временно изменить действие внутренних или конечных пробелов (9.3.4.6).

ФОРТРАН 77 не определял по умолчанию значений характеристик значимости пробелов для внутренних и предварительно подсоединенных файлов. Данный стандарт определяет это значение как 'NULL' для тех файлов, которые подсоединялись с помощью оператора OPEN.

Процессор имеет широкие возможности адаптировать эти понятия к своим условиям каталогизации и управления заданиями. Некоторые процессоры могут потребовать каких-то действий по управлению заданиями для того, чтобы задать набор файлов, которые существуют или которые будут созданы выполняемой программой. Некоторые процессоры могут не требовать никаких действий до выполнения. Данный стандарт разрешает процессору выполнять динамически операции открытия, закрытия и создания файла, но он не требует от процессоров таких возможностей.

Смысл термина "открыть" в контекстах, отличных от Фортрана, может включить такие вещи, как установка лент, сообщения на пульт, подкачка данных, проверка меток и защиты и т.д. Эти действия могут выполняться при работе управления заданиями (внешней по отношению к Фортрану), при выполнении оператора OPEN или при первом исполнении чтения или записи файла. Оператор OPEN описывает характеристики подсоединения к файлу и может привести (или не привести) к выполнению конкретных физических действий. Это сфера реализации - определить характеристики помимо тех, что требуются стандартом Фортрана.

C.9.5. Оператор CLOSE (9.3.5)

Аналогично, при "закрытии" в конце работы программы могут подразумеваться действия по снятию лент, защите и пр. Оператор CLOSE может вызвать, а может и не вызывать эти действия. Это еще одна область для расширения характеристик файла помимо требуемых стандартом Фортрана. Заметим однако, что выполнение оператора CLOSE для устройства , за которым следует оператор OPEN для этого же устройства и для того же или другого файла - допустимая последовательность событий. Процессор не должен ее запрещать только потому, что для реализации предпочтительно выполнять физический акт закрытия при завершении исполнения программы.

C.9.6. Оператор INQUIRE

В таблице C.1 приведены значения, присваиваемые переменным в спецификаторах оператора INQUIRE при отсутствии ошибок во время исполнения оператора.

C.9.7. Спецификаторы с ключевыми словами

Форма спецификаторов с ключевыми словами используется потому, что спецификаторов много и позиционная нотация трудна для запоминания. Форма с ключевыми словами устанавливает стиль для процессорных расширений. Ключевые слова UNIT= и FMT= включены для полноты картины, но их использование является необязательным. Таким образом, достигается совместимость с ANSI X3.9-1966 (ФОРТРАН 66) и ФОРТРАНом 77.

C.9.8. Спецификации формата (9.4.1.1)

Спецификации формата можно включать в операторы READ и WRITE, например,

READ(UNIT=10, FMT='(I3, A4, F10.2)') K, ALPH, X

C.9.9. Неформатный ввод/вывод (9.4.4.4.1)

Неформатный ввод/вывод, включающий элементы списка производного типа, образует единственное исключение из правила, что вхождение агрегатного элемента списка (например, массива) эквивалентно вхождению развернутого списка его компонентных частей. Это исключение дает большую свободу в повышении эффективности или в достижении соответствия процессорно-зависимой последовательности значений для объектов производного типа с аналогичными последовательностями агрегатных объектов, используемых вне Фортрана. Однако форматный ввод/вывод для всех элементов списка и неформатный ввод/вывод для элементов списка, имеющих тип, отличный от производного типа, подчиняется приведенному выше правилу.

Таблица С.1

Значения, присваиваемые переменным в операторе INQUIRE

Спецификатор	Запрос о файле		Запрос об устройстве	
	Не подсоединен	Подсоединен	Подсоединен	Не подсоединен
EXIST=	.TRUE., если файл существует, иначе .FALSE.		.TRUE., если устройство существует, иначе .FALSE.	
OPENED=	.FALSE.	.TRUE.	.TRUE.	.FALSE.
NUMBER=	-1	номер устройства		-1
NAMED=	.TRUE.	.TRUE.	.TRUE., если файл именован, иначе.FALSE.	.FALSE.
NAME=	имя файла (может отличаться от значения FILE=)		имя файла (если есть), иначе не определено	не определено
ACCESS=	UNDEFINED	SEQUENTIAL или DIRECT		UNDEFINED
SEQUENTIAL=	YES, NO или UNKNOWN			UNKNOWN
DIRECT=	YES, NO или UNKNOWN			UNKNOWN
FORM=	UNDEFINED	FORMATTED или UNFORMATTED		UNDEFINED
FORMATTED=	YES, NO или UNKNOWN			UNKNOWN
UNFORMATTED=	YES, NO или UNKNOWN			UNKNOWN
RECL=	не определено	при прямом доступе - длина записи, иначе - максимальная длина		не определено
NEXTREC=	не определено	при прямом доступе - номер следующей записи, иначе - не определено		не определено
BLANK=	UNDEFINED	NULL, ZERO, UNDEFINED		UNDEFINED
DELIM=	UNDEFINED	APOSTROPHE, QUOTE, NONE или UNDEFINED		UNDEFINED
PAD=	YES	YES или NO		YES
POSITION=	UNDEFINED	REWIND, APPEND, ASIS, UNDEFINED или процессорно-зависимое значение		UNDEFINED
ACTION=	UNDEFINED	READ, WRITE, или READWRITE		UNDEFINED
IOLENGTH=	RECL= значение для *элемента-списка-вывода*			

С.9.10. Ограничения на ввод/вывод

Примером такого ограничения (9.8) может служить следующее утверждение: нельзя в операторе ввода задавать чтение с принтера.

С.9.11. Указатели в списке ввода/вывода

Передача данных всегда включает перемещение значений между файлом и внутренней областью памяти. Указатель, как таковой, нельзя ни прочитать, ни записать. Тем не менее, он может быть включен в список ввода/вывода, если он в данный момент

связан с адресатом, который способен принять значение (ввод) или выдать его (вывод). Объект производного типа с одним или несколькими компонентами-указателями не может быть элементом списка ввода/вывода, т.к. значением компонента-указателя является дескриптор для элемента памяти, который не имеет процессорно-независимого представления.

C.9.12. Объекты производного типа в списке ввода/вывода (9.4.2)

Компонент производного типа может быть объявлен с атрибутом PRIVATE (4.4, 5.1.2.2). Объект производного типа не должен находиться в списке ввода/вывода, если какой-либо из его компонентов или подобъект компонента объявлен с атрибутом PRIVATE и описание этого производного типа не находится в одном модуле с оператором передачи данных.

В форматном операторе ввода/вывода описатели редактирования связываются с соответствующими элементами списка, которые всегда являются скалярами встроенных типов. Правила из 9.4.2 определяют множество элементов списка, соответствующих каждому фактическому элементу списка в операторе. Эти правила могут применяться повторно, пока все действующие элементы списка являются скалярными элементами встроенного типа.

C.10. Примечания к главе 10

C.10.1. Спецификация формата в виде символьной константы (10.1.2, 10.7.1)

Если в операторе ввода/вывода в качестве спецификатора формата используется символьная константа, следует обратить внимание на то, чтобы ее значение было правильной спецификацией формата. В частности, если спецификация формата содержит описатель редактирования в форме символьной строки, заключенной в апострофы, в качестве ограничителя должны использоваться сдвоенные апострофы, а вместо каждого апострофа внутри него должно быть записано по четыре апострофа. Например, текст:

```
2 ISN'T 3
```

можно записать разными комбинациями операторов вывода и спецификаций:

```
    WRITE(6,100) 2,3
100 FORMAT(1X,I1,'ISN''T',1X,I1)
    WRITE(6,'(1X,I1,1X,''ISN''''T'',1X,I1)') 2,3
    WRITE(6,'(A)') '2 ISN''T 3'
```

Заметим, что дублирования внутренних апострофов обычно можно избежать, используя в качестве ограничителей для спецификации формата кавычки, а дублирования внутренних кавычек можно избежать, используя в качестве ограничителей для спецификации формата апострофы.

C.10.2. Описатель редактирования T (10.6.1.1)

Описатель редактирования T включает число пропущенных строк (9.4.5) в строках, которые должны быть напечатаны. T1 специфицирует число пропущенных строк, а T2 - первый символ, который должен быть напечатан.

C.10.3. Длины форматных записей

Длина форматной записи не всегда специфицируется точно и может зависеть от процессора (10.8.2, 10.9.2).

C.10.4. Количество записей (10.3, 10.4, 10.6.2)

Количество записей, считываемых явно форматным оператором ввода с продвижением, можно определить по следующему правилу: запись читается в начале сканирования формата (даже если список ввода пуст), когда в формате встретился описатель редактирования "дробная черта" и когда повторное сканирование попадает на конец формата.

Число записей, записанных явно форматным оператором вывода, определяется следующим правилом: запись выводится , когда в формате встречается описатель редактирования "дробная черта"; когда после завершения спецификации формата выполняется повторное ее сканирование и при окончании выполнения оператора вывода (даже если список вывода пуст). Таким образом, *N* последовательных символов "дробная черта", расположенных между двумя другими описателями редактирования, вызывают печать *N*-1 пустых строк (если запись выводится на печать). Вхождение *N* символов "дробная черта" в начале или в конце всей спецификации формата вызывает печать *N* пустых строк. Однако полная спецификация формата, содержащая *N* символов "дробная черта" (*N*>0) и не содержащая больше никаких других описателей редактирования, вызывает печать *N*+1 пустых строк. Например, операторы

```
      PRINT 3
3     FORMAT(/)
```

запишут две записи, что вызовет печать двух пустых строк, если записи будут распечатываться.

C.10.5. Ввод/вывод, управляемый списком (10.8)

Вввод/вывод, управляемый списком, позволяет управлять редактированием данных в соответствии с типом элементов списка, а не в соответствии со спецификацией формата. Он также разрешает, чтобы данные находились в форме свободного поля, т.е. разделялись запятыми и пробелами.

Если в операторе ввода/вывода, управляемого списком, нет элементов списка, при вводе пропускается одна запись, а при выводе - выводится одна пустая запись.

C.10.6. Ввод, управляемый списком (10.8.1)

Приведенные ниже примеры иллюстрируют ввод, управляемый списком. Символ пробела изображается как *b*.

Пример 1.

Программа:

```
J=3
READ *,I
READ *,J
```

Файл последовательного ввода:

запись 1: *b1b,4bbbbb*
запись 2: *,2bbbbbbbb*

Результат: I=1, J=3

Объяснение: второй оператор READ читает вторую запись. Начальная запятая в записи обозначает пустое значение; таким образом, J не переопределяется.

Пример 2.

Программа:

```
CHARACTER A*8, B*1
READ *,A,B
```

Файл последовательного ввода:

запись 1: '*bbbbbbbb*'
запись 2: 'QXY'*b*'Z'

Результат: A='*bbbbbbbb*', B='Q'

Объяснение: в первой записи правый апостроф интерпретируется как ограничивающий константу (он не может быть первым в паре внутренних апострофов, представляющих один апостроф, т.к. запрещено разрывать спаренные апострофы записью "конец-записи"); таким образом, A получает значение, равное символьной константе '*bbbbbbbb*'. Конец записи действует как пробел, который в данном случае является разделителем значений, поскольку находится между двумя константами.

C.10.7. Элементы списка NAMELIST для ввода символьных данных (10.9.1.3)

Символьная константа, соответствующая символьному элементу списка ввода по NAMELIST, должна быть ограничена либо апострофами, либо кавычками. Ограничители нужны, чтобы избежать неоднозначности между символьными константами, не обрамленными ограничителями, и именами объектов. При вводе по NAMELIST игнорируется значение спецификатора DELIM= (если он есть) в операторе OPEN (9.3.4.9).

C.10.8. Записи вывода по NAMELIST (10.9.2.2)

Записи вывода по NAMELIST, полученные при значении NONE спецификатора DELIM= и содержащие символьные константы, могут быть непригодны для ввода по NAMELIST.

C.11. Примечания к главе 11

C.11.1. Главная программная единица и программная единица - блок данных (11.1, 11.4)

Имя главной программной единицы или программной единицы - блока данных явно в языке Фортран не используется. Оно нужно для документации и возможного использования в вычислительной среде.

Процессор может реализовать главную программную единицу без имени или неименованный блок данных, присвоив им какие-то имена по умолчанию. Однако эти имена в стандартной программе не должны вступать в конфликт с другими глобальными именами. Этого можно добиться, сделав такое имя запрещенным в стандартной программе (включив в него, например, символ, обычно в именах не разрешенный) или обеспечив некоторый механизм, меняющий в каждой конкретной программе это имя на другое, которое в ней нигде не используется.

C.11.2. Зависимая компиляция (11.3)

Данный стандарт, как и его предшественники, допускает реализацию стандартных процессоров, в которых программа может быть разбита на множество единиц, каждая из которых транслируется отдельно. В общем случае такие процессоры обычно описываются как поддерживающие раздельную компиляцию. Имеется существенная разница между способами реализации раздельной трансляции в рамках данного стандарта и в рамках предыдущих стандартов. В предыдущих стандартах любая информация, необходимая для трансляции программной единицы, задавалась прямо в

ней и, таким образом, каждая трансляция была полностью независима от других. В данном стандарте программная единица может использовать информацию, задаваемую отдельным модулем, и поэтому может зависеть от этого модуля. Реализация такой зависимости в процессоре может заключаться в том, что трансляция программной единицы зависит от результатов трансляции одного или нескольких модулей. Процессоры, реализующие зависимость такого рода, в общем случае рассматриваются как поддерживающие зависимую компиляцию.

Эти зависимости являются новыми только в том смысле, что Фортран-процессор сейчас знает о них. Те же информационные зависимости существовали и в предыдущих стандартах, но ответственность за передачу информации, необходимой для их решения путем дополнительных спецификаций во множестве программных единиц, ложилась на программиста. Наличие раздельной, но зависимой компиляции имеет некоторые потенциальные преимущества перед спецификацией дополнительной текстовой информации.

1. Задавая информацию в одном месте программы, можно быть уверенным в том, что разные программные единицы, использующие эту информацию, будут транслироваться согласованно. Дополнительная спецификация сохраняет вероятность ошибок в задании. Даже если для гарантии идентичности спецификаций, включаемых в разные программные единицы, используются средства включения текстов, наличие других спецификаций (например, оператора IMPLICIT) может изменить интерпретацию этого текста.
2. При переделке программы процессор может помочь определить, транслировались ли разные программные единицы с использованием разных (несовместимых) версий модуля, хотя стандарт и не требует от процессора обеспечения такой возможности. Противоречия в дополнительных текстовых спецификациях выявить гораздо труднее.
3. Помещение информации в модуль обеспечивает способ ее пакетирования. Без модулей дополнительные спецификации должны часто перемежаться другими спецификациями, что затрудняет пакетирование такой информации.
4. Поскольку процессор может быть реализован так, что спецификации в модуле транслируются один раз, а затем на них имеются повторные ссылки, появляются предпосылки для повышения эффективности по сравнению с вариантом, когда процессор должен транслировать дополнительные спецификации во множестве программных единиц.

Точный смысл требования, чтобы общедоступные части модуля были доступны во время ссылки, зависит от процессора. Например, процессор мог бы рассматривать модуль как доступный только после того, как он был скомпилирован, и потребовать, что, если модуль был скомпилирован раздельно, результат этой компиляции должен идентифицироваться компилятором в тот момент, когда компилируемые программные единицы используют его.

C.11.2.1. Оператор USE и зависимая компиляция (11.3.2)

Другим преимуществом оператора USE является расширение в нем возможности управления именами. Если кому-то надо использовать только некоторые объекты в модуле, он может это сделать, не беспокоясь об именах всех других объектов в этом модуле. Если кому-то надо использовать два разных модуля и может случиться, что они содержат объекты с одинаковыми именами, есть два способа устранить этот конфликт. Если ни один из одноименных объектов не используется, их можно просто проигнорировать. Может случиться, что это имя указывает на один и тот же объект в обоих модулях (если, например, оба модуля получили его из третьего модуля), тогда нет никаких сомнений в том, что это имя обозначает. Если же это имя обозначает разные объекты, причем один из них (или оба) используются, можно воспользоваться средством локального переименования в операторе USE, задавая этим объектам разные имена в программной единице, содержащей операторы USE.

Типичная реализация зависимой, но раздельной компиляции может включать запоминание результата трансляции модуля в файле (или в элементе файла), имя которого образуется из имени модуля. Отметим, однако, что имя модуля ограничивается только правилами Фортрана, а не именами, разрешенными в системе файлов. В этом случае процессор должен обеспечить отображение системных имен в фортрановские и наоборот.

Результат трансляции модуля может содержать либо только ту информацию, которая текстуально задана в модуле (с "указателями" на информацию, первоначальный текст которой задан в других модулях), либо содержать всю информацию, специфицированную в модуле (включая копии информации, первоначальный текст которой содержится в других модулях). Первый подход экономит память, зато второй сильно упрощает логику обработки оператора USE и позволяет избежать необходимости вводить ограничение на логическую "вложенность" модулей через оператор USE.

Переменные, описанные в модуле, сохраняют свой статус определенности на тех же основаниях, что переменные из общих блоков, т.е. сохраняемые переменные сохраняют его на все время работы программы, а несохраняемые - только на время исполнения контекстных областей, ссылающихся на этот модуль. Иногда может быть удобно поместить оператор USE вида:

```
USE MODULE, ONLY:
```

в контекстную область для того, чтобы процедуры, ссылающиеся на него, могли связываться через модуль. В этом случае программная единица не обращается к объектам из модуля, но несохраняемые переменные из модуля сохраняли бы свой статус определенности во время исполнения этой контекстной области.

В контекстной области, использующей и неявную типизацию, и оператор USE, повышается вероятность необнаруживаемых ошибок. Например, в программном фрагменте

```
SUBROUTINE SUB
    USE MY_MODULE
    IMPLICIT INTEGER (I-N), REAL (A-H,O-Z)
    X=F(B)
    A=G(X)+H(X+1)
END SUBROUTINE
```

X может быть либо вещественной переменной, тип которой определяется неявно, либо переменной, полученной из модуля MY_MODULE, и этот статус переменной X может меняться из-за изменений в модуле, не связанных с действиями, выполняемыми в SUB. Логические ошибки, возникающие в результате такого рода ситуаций, очень трудно обнаружить. Поэтому одновременное использование этих двух средств нежелательно.

С.11.2.2. Атрибуты доступности (11.3.1)

Атрибуты PUBLIC и PRIVATE, которые могут быть заданы только в модулях, делят объекты в модуле на действительно имеющие отношение к контекстной области, которая ссылается на модуль, и на не имеющие такого отношения. Эту информацию можно использовать для повышения производительности Фортран-процессора. Например, можно большую часть информации о приватных (PRIVATE) объектах выбросить сразу же по окончании трансляции модуля. Это сэкономит и память, и время на ее поиск. Аналогично можно определить, что две версии модуля отличаются только включенными в них приватными объектами, что дает возможность избежать перетрансляции программных единиц, использующих этот модуль, при переключении с одной версии модуля на другую.

С.11.3. Указатели в модулях

Указатель, объявленный в модуле, может быть доступен процедуре через связь по использованию. Время жизни таких указателей больше, чем у адресатов, описанных в процедуре (и несохраняемых). Таким образом, если такой указатель связан с локальным адресатом, имеется вероятность того, что по завершении выполнения процедуры адресат прекратит свое существование, а указатель станет "подвешенным". Данный стандарт считает статус таких указателей неопределенным. Их нельзя связывать и делать несвязанными. Их нельзя снова использовать в программе до тех пор, пока их статус не установится вновь. К процессору не пред'является никаких требований по поводу обнаружения того, что адресат указателя перестал существовать.

С.11.4. Пример модуля (11.3)

Помимо того, что модуль обеспечивает мобильные средства, позволяющие избежать дополнительного задания информации во многих программных единицах, модуль предоставляет также удобный способ "пакетирования" родственных объектов, таких как описания представления абстрактных типов данных и операций над ними. Приведенный ниже пример модуля определяет полную абстракцию данных для типа данных SET, где элементы каждого множества относятся к целому типу. Обеспечиваются стандартные операции над множествами - UNION, INTERSECTION и DIFFERENCE. Функция CARDINALITY выдает мощность (число элементов) своего аргумента-множества. Включены две функции, выдающие логические значения, ELEMENT и SUBSET. ELEMENT определяет операцию .IN., а SUBSET расширяет операцию <=. ELEMENT определяет, является ли данное скалярное целое значение элементом данного множества, а SUBSET определяет, является ли данное множество подмножеством данного множества. (Два множества можно проверить на равенство либо сравнением мощностей и проверкой, является ли одно подмножеством другого, либо проверкой, является ли каждое из них подмножеством другого).

Функция передачи SETF преобразует вектор целых значений в соответствующее множество, удаляя повторяющиеся значения. Так можно вектор константных значений использовать как константы множества. Функция инверсной передачи VECTOR выдает элементы множества как вектор значений в порядке возрастания. В такой реализации типа SET объекты множества имеют максимальную мощность, равную 200.

```
MODULE INTEGER_SETS
    ! Этот модуль предназначен для иллюстрации использования
    ! модульных возможностей в определении новых типов данных
    ! вместе с соответствующими операциями

    INTEGER, PARAMETER :: MAX_SET_CARD = 200
    TYPE SET                                  ! Описание типа данных SET
        PRIVATE
        INTEGER CARD
        INTEGER ELEMENT (MAX_SET_CARD)
    END TYPE SET

    INTERFACE OPERATOR (.IN.)
        MODULE PROCEDURE ELEMENT
    END INTERFACE

    INTERFACE OPERATOR (<=)
        MODULE PROCEDURE SUBSET
    END INTERFACE

    INTERFACE OPERATOR (+)
        MODULE PROCEDURE UNION
    END INTERFACE
```

```
INTERFACE OPERATOR (-)
    MODULE INTERFACE DIFFERENCE
END INTERFACE

INTERFACE OPERATOR (*)
    MODULE PROCEDURE INTERSECTION
END INTERFACE

CONTAINS

INTEGER FUNCTION CARDINALITY(A)  ! Выдает мощность множества A
    TYPE (SET) A
    CARDINALITY = A % CARD
END FUNCTION CARDINALITY

LOGICAL FUNCTION ELEMENT (X,A)   !  Определяет, является ли
    INTEGER X                    !  элемент X членом множества A
    TYPE (SET) A
    ELEMENT = ANY (A % ELEMENT (1 : A%CARD).EQ. X)
END FUNCTION ELEMENT

FUNCTION UNION (A, B)            ! Объединение множеств A и B
    TYPE (SET) A, B, UNION
    INTEGER J
    UNION = A
    DO J = 1, B % CARD
        IF (.NOT. (B % ELEMENT (J) .IN. A)) THEN
            IF (UNION % CARD < MAX_SET_CARD) THEN
             UNION % CARD = UNION % CARD + 1
             UNION % ELEMENT (UNION % CARD ) = &
                 B % ELEMENT (J)
            ELSE
                     ! Превышен максимальный размер множества
            END IF
        END IF
    END DO
END FUNCTION UNION

FUNCTION DIFFERENCE (A, B)           ! Разность множеств A и B
    TYPE (SET) A, B, DIFFERENCE
    INTEGER J, X
    DIFFERENCE % CARD = 0            ! Пустое множество
    DO J = 1, A % CARD
      X = A % ELEMENT (J)
      IF (.NOT.(X.IN.B)) DIFFERENCE = DIFFERENCE+SET(1, X)
    END DO
END FUNCTION DIFFERENCE

FUNCTION INTERSECTION (A,B)          ! Пересечение множеств A и B
    TYPE (SET) A, B, INTERSECTION
    INTERSECTION = A - (A - B)
END FUNCTION INTERSECTION
```

```
    LOGICAL FUNCTION SUBSET (A,B)          ! Определяет, является ли
      TYPE (SET) A, B                      ! множество A подмножеством B
      INTEGER I
      SUBSET = A % CARD <= B % CARD
      IF (.NOT.SUBSET) RETURN              ! Для эффективности
      DO I = 1, A % CARD
        SUBSET = SUBSET .AND. (A % ELEMENT (I) .IN. B)
      END DO
    END FUNCTION SUBSET

    TYPE(SET) FUNCTION SETF(V)   ! Функция передачи между вектором
        INTEGER V (:)            ! элементов и множеством элементов
        INTEGER J                ! с удалением дублированных значений
        SETF % CARD = 0
        DO J = 1, SIZE (V)
            IF (.NOT. (V (J) .IN. SETF)) THEN
                IF(SETF % CARD < MAX_SET_CARD) THEN
                    SETF % CARD = SETF % CARD + 1
                    SETF % ELEMENT (SETF % CARD) = V(J)
                ELSE
                          ! Превышен максимальный размер множества
                END IF
            END IF
        END DO
    END FUNCTION SETF

    FUNCTION VECTOR (A)          ! Передача значений из множества
        TYPE (SET) A             ! в вектор в порядке возрастания
        INTEGER, POINTER :: VECTOR (:)
        INTEGER I, J, K
        ALLOCATE (VECTOR (A % CARD))
        VECTOR = A % ELEMENT (1 : A % CARD)
        DO I = 1, A % CARD - 1       ! Если A%CARD большое, исполь-
          DO J = I+1, A % CARD       ! зуйте лучшую сортировку
            IF (VECTOR (I) > VECTOR (J)) THEN
             K=VECTOR(J); VECTOR(J)=VECTOR(I); VECTOR(I)=K
            END IF
          END DO
        END DO
    END FUNCTION VECTOR
END MODULE INTEGER_SETS
```

Примеры использования INTEGER_SETS (A, B и C - переменные типа SET, X - целая переменная):

```
        ! Проверка, имеет ли A больше 10 элементов
IF (CARDINALITY (A) > 10) ...
        ! Проверка, является ли X элементом A, а не элементом B
IF (X .IN. A - B) ...
        ! C есть объединение A и результата пересечения B с целыми от 1 до 100
C = A + B * SETF ((/ (I, I = 1, 100 ) /))
        ! Есть ли в A четные числа из диапазона 1 : 100 ?
IF (CARDINALITY (A * SETF ((/ (I, I = 2, 100, 2) /))) > 0)...
PRINT *, VECTOR (B)              ! Печать элементов множества B
                                 ! в порядке возрастания
```

C.12. Примечания к главе 12

C.12.1. Примеры связи по объемлющему контексту (12.1.2.2.1)

Первые два примера являются примерами правильной связи по объемлющему контексту. Третий пример - пример неправильной связи по объемлющему контексту.

Пример 1:

```
PROGRAM A
   INTEGER I, J
      ...
   CONTAINS
   SUBROUTINE B
      INTEGER I       ! Объявление I перекрывает объявление
       ...            !   I в программе A
      I = J           ! Использование переменной J из программы
                      !   A через связь по объемлющему контексту
   END SUBROUTINE B
END PROGRAM A
```

Пример 2:

```
PROGRAM A
   TYPE T
   ...
   END TYPE T
   ...
CONTAINS
   SUBROUTINE B
      IMPLICIT TYPE (T) (C)     ! Ссылка на тип T, описанный ниже
                                ! в подпрограмме B, а не на тип T,
                                ! описанный в программе A
       ...
      TYPE T
      ...
      END TYPE T
      ...
   END SUBROUTINE B
END PROGRAM A
```

Пример 3:

```
PROGRAM Q
   REAL (KIND = 1) :: C
   ...
   CONTAINS
   SUBROUTINE R
      REAL (KIND=KIND(C))::D     ! Неверное объявление (см. ниже)
      REAL (KIND = 2) :: C
        ...
   END SUBROUTINE R
END PROGRAM Q
```

В объявлении D в подпрограмме R использование C ссылается на объявление C из подпрограммы R, а не из программы Q. Однако это неверно, поскольку объявление C должно располагаться ранее своего использования в объявлении D.

C.12.2. Внешние процедуры (12.3.2.2)

Среди разных типов процедур, описанных в этой главе, только внешние процедуры имеют глобальные имена. Реализация может пожелать присвоить глобальные имена другим объектам Фортран-программы, таким, как внутренние процедуры, встроенные процедуры, процедуры, реализующие встроенные операции или операции ввода/вывода и т.д. В этом случае ответственность за отсутствие конфликта в стандартной программе между этими именами и именами внешних процедур или других глобальных объектов ложится на процессор. Можно, например, решить эту проблему включением в каждое такое имя символа, запрещенного для имени, соответствующего стандарту, или использовать такой символ для комбинации обозначения локального имени с глобальным именем программной единицы, в которой находится имя.

В контекстной области, которая ссылается на внешнюю процедуру без объявления ее в операторе EXTERNAL и без блока интерфейса, возникает проблема мобильности программы. На одном процессоре имя этой процедуры может оказаться именем встроенной процедуры, и процессор может проинтерпретировать ссылку на эту процедуру как ссылку на встроенную процедуру (на таком процессоре программа будет считаться не соответствующей стандарту из-за ссылки на нестандартную встроенную процедуру). Объявление в операторе EXTERNAL или блок интерфейса процедуры приводят к ссылке на внешнюю процедуру независимо от наличия встроенной процедуры с тем же именем. Заметим, что объявления типа процедуры недостаточно, чтобы считать ее внешней, даже если ее тип противоречит типу результата встроенной функции с тем же именем.

C.12.3. Процедуры, определяемые способом, отличным от Фортрана (12.5.3)

Процессор не обязан обеспечивать способы определения внешних процедур, отличные от Фортрана. Среди способов, которые могут поддерживаться, могут быть другие языки высокого уровня, язык машинного ассемблера, язык Фортран, расширенный нестандартными средствами, и язык Фортран, поддерживаемый другим процессором (например, реализованным ранее процессором с ФОРТРАНа 77).

Процедуры, описанные способом, отличным от Фортрана, рассматриваются как внешние процедуры, поскольку их определения не находятся в программных единицах Фортрана, и ссылки на них осуществляются с использованием глобальных имен. Использование термина "внешняя" не следует толковать как некоторое ограничение способа определения этой процедуры. Например, если способ, отличный от Фортрана, может создать "внутреннюю" процедуру с глобальным именем, допустимо, чтобы такая "внутренняя" процедура рассматривалась Фортраном как внешняя. Отличный от Фортрана способ описания внешних процедур, включая все ограничения на структуру организации этих процедур, полностью зависит от процессора.

Фортран-процессор может ограничить свою поддержку процедур, определенных способом, отличным от Фортрана, так, чтобы эти процедуры воздействовали на объекты из фортрановской среды только на той же основе, что и процедуры Фортрана. Например, можно разрешить, чтобы ссылка на процедуру меняла значение локальной переменной только тогда, когда та является формальным аргументом процедуры.

C.12.4. Интерфейсы процедур (12.3)

В ФОРТРАНе 77 интерфейс с внешней процедурой всегда выводился из форм ссылок на нее и всех объявлений имени процедуры в вызывающей программной единице. В данном стандарте такие средства, как необязательные аргументы и аргументы с ключевыми словами, делают невозможным извлечение правильной информации о формальных аргументах процедуры из характера связанных с ними фактических аргументов, а такие средства, как функции с результатом-массивом и с результатом-указателем, делают необходимыми расширения в объявлении процедур, которые невозможно сделать способами, имеющими аналоги при обработке объявлений в

ФОРТРАНе 77. Таким образом, требуются механизмы, посредством которых всю информацию об интерфейсе процедуры можно было сделать доступной в вызывающей ее контекстной области. Процедура, интерфейс которой надо устанавливать, как в ФОРТРАНе 77, описывается как процедура с неявным интерфейсом. Процедура, интерфейс которой известен полностью, описывается как процедура с явным интерфейсом.

Разрешается, чтобы контекстная область содержала блок интерфейса для процедуры, которой нет в выполняемой программе, при условии, что на описанную процедуру никогда не ссылаются. Цель этого правила - разрешить реализации, в которых использование модуля с блоками интерфейса процедур, описывающими интерфейс каждой программы в библиотеке, не требовало бы, чтобы каждая из этих библиотечных программ была бы частью выполняемой программы. (В терминах реализации - простое наличие блока интерфейса процедуры не должно в такой реализации вызывать генерацию внешней ссылки).

C.12.5. Связь и вычисление аргументов (12.4.1)

Есть существенная разница между допустимой в данном стандарте связью аргументов и той, что поддерживалась ФОРТРАНом 77 и ФОРТРАНом 66. Там фактические аргументы были ограничены последовательными единицами памяти. Кроме случая символьных формальных аргументов неопределенной длины, структура построения этой последовательности всегда определялась в вызываемой процедуре, а не выводилась из фактических аргументов. Таким образом, в ФОРТРАНе 77 и ФОРТРАНе 66 можно было реализовать связь аргументов, предоставив только адрес первой единицы памяти (для символьных аргументов добавив еще и длину). Данный же стандарт допускает аргументы, не занимающие последовательные ячейки памяти (например, секции массива), и формальные аргументы, рассчитанные на дополнительную структурную информацию из фактических аргументов (например, формальные аргументы неопределенной конфигурации). Таким образом, механизм реализации связи аргументов для данного стандарта должен быть более общим.

Поскольку на практике предпочтительнее процессор, который способен поддерживать ссылки на процедуры, определенные в ФОРТРАНе 77, и обратно, добавляется требование явного интерфейса, чтобы можно было определить, можно ли использовать простой (ФОРТРАН 77/ФОРТРАН 66) механизм реализации связи аргументов, или же нужен более общий механизм (12.3.1.1). Таким образом, можно реализовать процессор, процедуры которого будут вызывать либо простой механизм, если интерфейс процедур использует только средства ФОРТРАНа 77, либо более общий механизм в противном случае (например, если есть необязательные аргументы или аргументы неопределенной конфигурации). Определить, какой именно механизм нужен, можно в точке ссылки по интерфейсу процедуры. Заметим, что если выяснится, что процедуре нужен простой механизм, от процессора могут потребоваться следующие действия: отвести временную непрерывную память для фактического аргумента, скопировать в эту область памяти фактический аргумент, обратиться к процедуре, использующей временную память вместо фактического аргумента, скопировать содержимое временной памяти обратно в фактический аргумент, освободить временную память.

Заметим, что это конкретный, но не единственно возможный, способ реализации, предназначенный для поддержания этих правил. Например, на некоторых процессорах можно будет обеспечить более общую связь аргументов таким образом, чтобы информацию, включенную в связь аргументов в ФОРТРАНе 77, можно было оставить в том же виде, а вся дополнительная информация размещалась бы так, чтобы не повредить процедуре, ориентированной только на связь аргументов в ФОРТРАНе 77. В такой реализации связь аргументов может транслироваться независимо от того, явный интерфейс или неявный. Можно также запретить разрывные аргументы при вызове процедур, определенных в процессоре с ФОРТРАНа 77, и устроить в программе копирование в непрерывную память и из нее явным образом. И еще одна возможность - запретить ссылки на процедуры, определенные процессором для ФОРТРАНа 77.

Условия вычисления выражений дают процессору большую гибкость для получения значения выражений наиболее эффективным из возможных способом. Примером может служить отказ от вычисления или частичное вычисление операнда, если значение выражения можно определить иначе (7.1.7.1). То же можно сказать и о вычислении аргументов функции, включая последовательность их вычисления, откладывание вычисления и вообще отмену вычисления аргументов. Процессор может отложить вычисление аргумента в ссылке на процедуру до выполнения процедуры, ссылающейся на этот аргумент, при условии, что отсрочка вычисления не скажется на результатах выполняемой программы. Процессор может, с аналогичными ограничениями, вообще опустить вычисление аргумента, на который нет ссылки при выполнении процедуры. Это дает процессору простор для оптимизации (например, при параллельной обработке).

Заметим, что нельзя использовать последовательные запятые, чтобы опустить необязательные аргументы.

C.12.6. Спецификация назначения аргументов (12.4.1.1)

Спецификации назначения аргументов, помимо документирования назначения при использовании формальных аргументов, служат и для других целей. Процессор может проверить, не используется ли формальный аргумент с атрибутом INTENT(IN) таким образом, что он может быть переопределен. Несколько более совершенный процессор может проверить, есть ли возможность ссылки на формальный аргумент с атрибутом INTENT(OUT) до его определения.

Если у процедуры явный интерфейс, процессор может удостовериться в том, что фактические аргументы, соответствующие формальным аргументам с атрибутами INTENT(OUT) и INTENT(INOUT), являются определимыми. Более совершенный процессор может использовать эту информацию для оптимизации трансляции вызывающей контекстной области, используя тот факт, что фактические аргументы, соответствующие формальным аргументам с атрибутом INTENT(IN), не будут изменяться, а любое первоначальное значение фактического аргумента, соответствующего формальному аргументу с атрибутом INTENT(OUT), не будет использовано и его можно игнорировать.

Заметим, что атрибут INTENT(OUT) подразумевает, что значение аргумента после вызова процедуры целиком является результатом выполнения этой процедуры. Если есть хоть какая-то вероятность того, что аргумент сохранит свое текущее значение, а не будет переопределен, атрибут будет скорее INTENT(INOUT) , чем INTENT(OUT), даже если нет явной ссылки на значение формального аргумента.

Заметим, что значение атрибута INOUT не эквивалентно умолчанию. Аргумент, соответствующий формальному аргументу с атрибутом INTENT(INOUT), должен всегда быть определимым, тогда как аргумент, соответствующий формальному аргументу с атрибутом назначения по умолчанию, обязан быть определимым только в случае, если формальный аргумент действительно переопределяется.

C.12.7. Ограничения на формальные аргументы (12.5.2.9)

Ограничения на объекты, связанные с формальными аргументами, позволяют процессору транслировать процедуру в предположении, что каждый формальный аргумент отличается от любого другого объекта, доступного в процедуре. Это позволяет выполнять разнообразные оптимизации при трансляции процедуры, включая реализации с такой связью аргументов, при которой значение фактического аргумента хранится в регистре или в локальной памяти.

C.12.8. Указатели и адресаты в качестве аргументов

Если формальный аргумент описан как указатель, он может ставиться в соответствие только фактическому аргументу-указателю и характеристики обоих аргументов должны согласовываться. Моделью такой связи является копирование значений дескриптора фактического указателя в формальный аргумент. Если у

фактического указателя есть связанный с ним адресат, он становится доступным через формальный указатель. Если формальный указатель будет во время исполнения процедуры связан с другим адресатом, этот адресат будет доступен через фактический указатель после исполнения процедуры. Если формальный указатель становится связанным с локальным адресатом, который прекратит свое существование после завершения процедуры, фактический указатель останется "подвешенным" и статус его - не определен. Такие "подвешенные" указатели использовать нельзя.

Т.к. все это предназначается для того, чтобы позволить реализаторам делать копии фактических аргументов в локальной области памяти, указатель, связанный с фактическим аргументом - не указателем, не связывается с соответствующим формальным аргументом, поскольку позднее это может привести к запоминанию копии в локальной области памяти. Аналогично при возврате указатель, связанный с формальным аргументом - не указателем, становится неопределенным, т.к. формальный аргумент мог быть сохранен в локальной области памяти, которая далее не будет доступна. Если же нужно предусмотреть именно такую связь указателя, формальный аргумент следует заменить указателем на объект-адресат. Например, рассмотрим процедуру:

```
SUBROUTINE BEST (P,A)
   REAL, POINTER :: P
   REAL, TARGET :: A (:)
   ...                                  ! Найти наилучший элемент, A(I)
   P => A(I)
   RETURN
END
```

и ее вызов

```
REAL, POINTER :: PBEST
REAL, TARGET  :: B(10000)
    ...
CALL BEST (PBEST, PB)
```

Здесь PBEST остается неопределенным. Однако, если задать A с атрибутом POINTER вместо TARGET, т.е.

```
SUBROUTINE BEST (P,A)
   REAL, POINTER :: P
   REAL, POINTER :: A (:)
   ...                                  ! Найти наилучший элемент, A(I)
   P => A(I)
   RETURN
END
```

тогда вызов

```
REAL, POINTER :: PB(:), PBEST
REAL, TARGET  :: B(10000)
PB => B
...                                     ! Вычисляется B
CALL BEST (PBEST, B)
```

оставляет PBEST связанным с "лучшим" элементом B.

C.12.9. Функция ASSOCIATED (13.13.13)

Чтобы проверить, связан ли указатель с адресатом, можно использовать встроенную функцию ASSOCIATED. Для этих целей используется ее форма с одним аргументом. В форме с двумя аргументами функция проверяет, связан ли указатель - первый аргумент - с областью памяти, на которую ссылается второй аргумент. Большей частью так проверяют, связаны ли два указателя с одним адресатом.

C.12.10. Ограничения на внутренние процедуры

Данный стандарт не разрешает использовать внутренние процедуры в качестве фактических аргументов, чтобы было удобнее контролировать, что внутренние процедуры с рекурсивным объемлющим контекстом обращаются к объектам из правильного экземпляра объемлющего контекста. Если же, как расширение, процессор разрешает использовать внутреннюю процедуру в качестве фактического аргумента, корректным считается тот экземпляр, в который процедура передается как фактический аргумент, даже если соответствующий формальный аргумент случайно вызван из другого экземпляра.

C.12.11. Результирующая переменная (12.5.2.2)

Результирующая переменная аналогична любой другой переменной, локальной для данной функции. Ее существование начинается, когда инициируется выполнение функции, и заканчивается с завершением выполнения функции. Однако, поскольку окончательное значение этой переменной используется в дальнейшем при вычислении значения выражения, вызвавшего функцию, реализация может отложить освобождение занимаемой ею памяти до конца использования ее значения в вычислении выражения.

C.13. Примечания к главе 13

C.13.1. Резюме

Этот раздел представляет собой краткое изложение основных свойств массивов.

C.13.1.1. Выражения и присваивания для массивов (7.5.1.2, 7.5.1.5)

Важным нововведением является разрешение выражений и присваиваний для полных массивов.

Например, оператор

```
A = B + C * SIN (D)
```

где A, B, C и D - массивы одинаковой конфигурации, является допустимым. Он интерпретируется поэлементно, т.е. функция синуса берется для каждого элемента D, каждый результат умножается на соответствующий элемент C, добавляется к соответствующему элементу B и присваивается соответствующему элементу A. Функции, включая функции, написанные пользователем, могут иметь значения-массивы и могут настраиваться на скалярные варианты. Все массивы в выражении или в левой части присваивания должны согласовываться, т.е. должны иметь одинаковую "конфигурацию" (число измерений и множество длин для каждого измерения), при этом скаляры могут включаться свободно и интерпретируются как размноженные в соответствии с конфигурацией массива. Выражения вычисляются прежде, чем производится любое присваивание.

C.13.1.2. Секции массива. (2.4.5, 6.2.2.3)

Везде, где можно использовать полные массивы, допустимо использование подмассивов, называемых "секциями". Например:

```
A (:, 1:N, 2, 3:1:-1)
```

образует подмассив, содержащий целиком первое измерение, позиции с 1 по N второго измерения, позицию 2 третьего измерения и позиции с 1 по 3 из четвертого измерения в обратном порядке. Это искусственный пример, выбранный для иллюстрации разнообразия форм. Конечно, более обычным применением может быть выбор строки или столбца из массива, как в примере:

```
A(:,J)
```

C.13.1.3. Оператор WHERE (7.5.3)

Оператор WHERE использует согласованный логический массив в качестве маски для отдельных операций в выражении и в присваивании. Например:

```
WHERE (A.GT.0) B = LOG (A)
```

берется логарифм только от положительных компонентов A и производится присваивание только в этих позициях.

Оператор WHERE имеет также блочную форму (конструкция WHERE).

C.13.1.4. Динамически и автоматически размещаемые массивы (5.1, 5.1.2.4.3)

Существенное преимущество при написании модульного программного обеспечения дает наличие автоматически размещаемых массивов, создаваемых при входе в процедуру и уничтожаемых при выходе из нее, и динамически размещаемых массивов, размерность которых фиксирована, но фактический размер и время жизни полностью управляются программистом явно с помощью операторов ALLOCATE и DEALLOCATE. Объявления

```
SUBROUTINE X (N,A,B)
    REAL WORK(N,N); REAL, ALLOCATABLE::HEAP(:,:)
```

задают автоматически размещаемый массив WORK и динамически размещаемый массив HEAP. Отметим, что для реализации автоматически размещаемых массивов адекватным механизмом памяти является стек, а для динамически размещаемых массивов нужна динамическая область - "куча".

C.13.1.5. Конструкторы массива (4.5)

Массивы и, в частности, константы-массивы могут быть сконструированы при помощи конструкторов массива, как в приведенном примере:

```
(/1.0, 3.0, 7.2/)    ,
```

который представляет одномерный массив размера 3 или

```
(/(1.3, 2.7, L=1,10), 7.1/)
```

это тоже массив размерности 1, но размера 21, содержащий пару вещественных констант 1.3 и 2.7, повторенную 10 раз, а затем 7.1, и, наконец,

```
(/ (I, I = 1,N) /)    ,
```

массив, содержащий целые числа 1,2,...,N. Таким способом можно сконструировать только одномерные массивы, а массивы большей размерности можно получить из них посредством встроенной функции RESHAPE.

C.13.1.6. Встроенные функции

Все встроенные функции ФОРТРАНа 77, кроме LEN, и все добавленные в язык скалярные встроенные функции, кроме REPEAT и TRIM, применимы и к массивам. Каждая такая функция применяется поэлементно, вырабатывая массив той же конфигурации. Кроме того, были добавлены следующие встроенные функции для массивов, причем многие из них выдают в качестве результата значение-массив.

C.13.1.6.1. Функции векторного и матричного умножения

DOT_PRODUCT (VECTOR_A, VECTOR_B)	скалярное произведение двух массивов
MATMUL (MATRIX_A, MATRIX_B)	умножение матриц

C.13.1.6.2. Функции редукции массивов

ALL (MASK, DIM)	"истина", если все значения - "истина"
ANY (MASK, DIM)	"истина", если какое-либо значение - "истина"
COUNT (MASK, DIM)	число элементов в массиве, имеющих значение "истина"
MAXVAL (ARRAY, DIM, MASK)	максимальное значение в массиве
MINVAL (ARRAY, DIM, MASK)	минимальное значение в массиве
PRODUCT (ARRAY, DIM, MASK)	произведение элементов массива
SUM (ARRAY, DIM, MASK)	сумма элементов массива

C.13.1.6.3. Функции запроса о массиве

ALLOCATED (ARRAY)	статус размещенности массива
LBOUND (ARRAY, DIM)	нижние границы измерений массива
SHAPE (SOURCE)	конфигурация массива или скаляра
SIZE (ARRAY, DIM)	общее число элементов массива
UBOUND (ARRAY, DIM)	верхние границы измерений массива

C.13.1.6.4. Функции конструирования массивов

MERGE (TSOURCE, FSOURCE, MASK)	слияние по маске
PACK (ARRAY, MASK, VECTOR)	упаковка массива в вектор по маске
SPREAD (SOURCE, DIM, NCOPIES)	копирование массива добавлением измерений
UNPACK (VECTOR, MASK, FIELD)	распаковка вектора в массив по маске

C.13.1.6.5. Функция изменения конфигурации массива

RESHAPE (SOURCE, SHAPE, PAD, ORDER)	изменение конфигурации массива

C.13.1.6.6. Функции манипуляций над массивами

CSHIFT (ARRAY, DIM, SHIFT)	циклический сдвиг
EOSHIFT (ARRAY, SHIFT, BOUNDARY, DIM)	нециклический сдвиг
TRANSPOSE (MATRIX)	транспонирование матрицы

C.13.1.6.7. Функции определения положения в массиве

MAXLOC (ARRAY, MASK) положение максимального значения в массиве
MINLOC (ARRAY, MASK) положение минимального значения в массиве

C.13.2. Примеры

Средства работы с массивами дают большие возможности, позволяющие упростить способ организации и написания почти всех программ, использующих массивы. Многие алгоритмы, включающие массивы, сейчас могут быть более удобно записаны как последовательность вычислений над полными массивами.

C.13.2.1. Безусловные вычисления массива

На простейшем уровне такие операторы, как

```
A=B+C
```

или

```
S=SUM(A)
```

могут заменить полные DO-циклы, необходимые для того, чтобы произвести суммирование всех элементов массива или сложение массивов.

Несколько примеров безусловных операций над массивами, которые можно просто записать следующим образом:

умножение матриц	P=MATMUL(Q,R)
наибольший элемент массива	L=MAXVAL(P)
N-факториал	F=PRODUCT ((/ K,K=2,N /))

Сумму Фурье $F = \sum_{i=1}^{N} a_i \times \cos x_i$

тоже можно вычислить без записи цикла DO, если использовать поэлементное определение выражений-массивов, как описано в главе 7. Таким образом, мы можем записать:

```
F = SUM(A * COS(X))
```

Тогда в последовательных этапах вычисления F будут участвовать массивы:

```
        A    = (/A(1), ..., A(N)/)
        X    = (/X(1), ..., X(N)/)
     COS(X)  = (/COS(X(1)), ..., COS(X(N))/)
A*COS(X)     = (/A(1)*COS(X(1)), ..., A(N)*COS(X(N))/)
```

Окончательный скалярный результат будет получен просто суммированием элементов последнего из этих массивов. Таким образом, на каждом шаге вычисления процессор имеет дело с массивами.

C.13.2.2. Условные вычисления массива

Предположим, мы хотим вычислить сумму Фурье в предыдущем примере, но включать в нее только те термы *a(i)cos x(i)* которые удовлетворяют условию, что коэффициент *a(i)* по абсолютному значению меньше 0.01. Если более точно, сейчас мы хотим вычислить условную сумму Фурье

$$CF = \sum_{|a_i|<0.01} a_i \times \cos x_i$$

где индекс меняется от 1 до N, как и раньше.

Это можно сделать с помощью параметра MASK в функции SUM, который ограничивает суммирование элементов массива A*COS(X) теми элементами, которые соответствуют элементам MASK со значением "истина". Ясно, что нужной маской является логическое выражение-массив ABS(A) .LT. 0.01. Этапы вычисления этого выражения следующие:

```
            A    = (/A(1),...,A(N)/)
       ABS(A)    = (/ABS(A(1)),...,ABS(A(N))/)
ABS(A).LT.0.01   = (/ABS(A(1)).LT.0.01,...,ABS(A(N)).LT.0.01/)
```

Условную сумму Фурье мы получаем так:

```
CF = SUM (A * COS(X), MASK = ABS (A) .LT. 0.01)
```

Если все элементы имеют значение "ложь", значением CF является нуль.

Использование маски для определения подмножества является определяющим в действии оператора WHERE. Например, чтобы присвоить нуль всему массиву, мы можем просто написать A=0; но чтобы присвоить нуль только отрицательным элементам, потребуется записать условное присваивание:

```
WHERE (A.LT.0) A=0
```

Оператор WHERE дополняет обычное присваивание массиву, обеспечивая возможность присваивания любому подмножеству массива, которое может быть ограничено логическим выражением.

В описанной ниже модели Изинга оказывается более предпочтительным использование оператора WHERE, чем обычного оператора присваивания массиву.

C.13.2.3. Простая программа: модель Изинга

Модель Изинга - это широко известное моделирование методом Монте-Карло в трехмерном эвклидовом пространстве, используемое в некоторых физических исследованиях. Рассмотрим некоторые детали того, как это можно запрограммировать. Модель может быть описана в терминах логического массива конфигурации N на N на N. Каждый узел сетки - простая логическая переменная, которая должна интерпретироваться либо как верхний спин ("истина"), либо как нижний спин ("ложь").

Модель Изинга в процессе работы проходит через множество последовательных этапов. Переход на следующий этап управляется локальным вероятностным процессом. При каждом переходе все узлы сетки меняют состояние одновременно. Каждый спин либо перебрасывается в противоположное состояние, либо нет, в зависимости от состояний 6 ближайших соседних узлов сетки. Ближайшие соседние узлы сетки на граничных гранях модельного куба определяются в предположении кубичной периодичности. В действительности, сетка периодически расширяется повторением ее во всех пространственных направлениях.

Правило устанавливает, что спин перебрасывается в противоположную четность для определенных узлов, у которых не более, чем 3 из 6 ближайших соседних узлов имеют ту же четность. Кроме того, переброс выполняется только с вероятностью P(4), P(5) или P(6) в зависимости от того, 4, 5 или 6 из них имеют то же значение.

C.13.2.3.1. Задачи, которые должны быть решены

Для того, чтобы перевести модель Изинга в операторы Фортрана, использующие полные массивы, надо решить некоторые программистские задачи:

1) подсчитать ближайшие величины с тем же спином;
2) обеспечить функции со значением-массивом для выдачи массива случайных чисел;
3) определить, какие узлы сетки должны быть переброшены.

C.13.2.3.2. Решения в Фортране

Нужны массивы:

```
LOGICAL ISING (N, N, N), FLIPS (N, N, N)
INTEGER ONES (N, N, N), COUNT (N, N, N)
REAL THRESHOLD (N, N, N)
```

Нужна функция со значением-массивом:

```
FUNCTION RAND(N,N,N)
REAL RAND(N,N,N)
```

Вероятности перехода могут быть переданы через массив:

```
REAL P(6)
```

Первая задача - подсчитать количество ближайших величин каждого узла сетки G, имеющих тот же спин, что у G.

Предположим, что ISING задан, тогда операторы

```
ONES = 0
WHERE (ISING) ONES = 1
```

делают массив ONES точным аналогом ISING, в котором 1 устанавливается для верхнего спина, а 0 - для нижнего спина.

Следующий массив, который мы создаем, COUNT, будет записывать для каждого узла ISING число спинов, которые надо найти среди 6 ближайших величин этого узла. COUNT будет вычислен сложением вместе 6 массивов, по одному для каждой относительной позиции, в которой найдена такая ближайшая величина. Каждый из этих 6 массивов получается из массива ONES циклическим сдвигом на 1 позицию вокруг одного из его измерений. Такое использование циклического сдвига передает кубичную периодичность.

```
COUNT = CSHIFT (ONES, SHIFT=-1, DIM=1) &
      + CSHIFT (ONES, SHIFT= 1, DIM=1) &
      + CSHIFT (ONES, SHIFT=-1, DIM=2) &
      + CSHIFT (ONES, SHIFT= 1, DIM=2) &
      + CSHIFT (ONES, SHIFT=-1, DIM=3) &
      + CSHIFT (ONES, SHIFT= 1, DIM=3)
```

В этой точке COUNT содержит число верхних спинов ближайших величин даже в узлах, где модель Изинга имеет нижний спин. А нам нужно число нижних спинов в этих узлах, поэтому мы корректируем COUNT в нижних (имеющих значение "ложь") точках Изинга, записав:

```
WHERE (.NOT. ISING) COUNT = 6 - COUNT
```

Теперь наш объект должен использовать эти счетчики чего-то, что можно назвать "одинаково выглядящими ближайшими величинами", чтобы решить, какие узлы должны быть переброшены. Это решение должно быть записано элементами массива FLIP, имеющими значение "истина". Решение о переброске должно базироваться на равномерном распределении случайных чисел из интервала $0 \leq p < 1$. Это должна обеспечить в каждом узле функция RAND со значением-массивом. Переброс в данной точке произойдет тогда и только тогда, когда случайное число в ней меньше определенного порогового значения. В частности, установив пороговое значение равным 1 в точках, где есть 3 или меньше одинаково выглядящих ближайших величины, мы гарантируем, что в этих точках произойдет переброс (поскольку p всегда меньше 1). Аналогично, пороговые значения, соответствующие счетчикам 4, 5 и 6, устанавливаются равными P(4), P(5) и P(6) для того, чтобы получить нужные вероятности переброса в этих точках (P(4), P(5) и P(6) - входные параметры в диапазоне от 0 до 1).

Пороги устанавливаются операторами:

```
THERESHOLD = 0.0
WHERE (COUNT .EQ. 4) THRESHOLD = P(4)
WHERE (COUNT .EQ. 5) THRESHOLD = P(5)
WHERE (COUNT .EQ. 6) THRESHOLD = P(6)
```

а спины, которые должны быть переброшены, устанавливаются оператором

```
FLIPS = RAND (N) .LE. THRESHOLD
```

Теперь для завершения одного перехода на следующий этап модели Изинга остается перебросить спины в массиве ISING везде, где FLIPS есть "истина".

```
WHERE (FLIPS) ISING = .NOT. ISING
```

C.13.2.3.3. Полная подпрограмма на Фортране

Вся программа заключена в подпрограмму, выполняющую последовательность переходов, и выглядит следующим образом:

```
SUBROUTINE TRANSITION (N, ISING, ITERATIONS, P)
    LOGICAL ISING (N, N, N), FLIPS (N, N, N)
    INTEGER ONES (N, N, N), COUNT (N, N, N)
    REAL THRESHOLD (N, N, N), P(6)
```

```
    DO I = 1, ITERATIONS
        ONES = 0
        WHERE (ISING) ONES = 1
        COUNT = CSHIFT (ONES,-1, 1) + CSHIFT (ONES, 1, 1) &
              + CSHIFT (ONES,-1, 2) + CSHIFT (ONES, 1, 2) &
              + CSHIFT (ONES,-1, 3) + CSHIFT (ONES, 1, 3)
        WHERE (.NOT. ISING) COUNT = 6 - COUNT
        THRESHOLD = 1.0
        WHERE (COUNT .EQ. 4) THRESHOLD = P(4)
        WHERE (COUNT .EQ. 5) THRESHOLD = P(5)
        WHERE (COUNT .EQ. 6) THRESHOLD = P(6)
        FLIPS = RAND (N) .LE. THRESHOLD
        WHERE (FLIPS) ISING = .NOT. ISING
    END DO
    CONTAINS
    FUNCTION RAND (N)
        REAL RAND (N, N, N)
        CALL RANDOM_NUMBER (HARVEST = RAND)
        RETURN
    END FUNCTION RAND
END
```

C.13.2.3.4. Уменьшение памяти

Массив ISING можно удалить (с некоторой потерей в наглядности) за счет представления модели все время в массиве ONES. Без массива FLIPS можно обойтись, скомбинировав два оператора, которые его используют

```
WHERE(RAND(N).LE.THRESHOLD) ISING = .NOT. ISING
```

но дополнительный временный массив может понадобиться. Таким образом, область сохраняемой памяти во время выполнения операций над массивами ограничена. Если N невелико, в этом нет резона, а использование операций над массивами удобно для поддержания высокой скорости исполнения. Если N велико, память может оказаться очень важна и адекватную эффективность можно, вероятно, получить, выполняя операции плоскость за плоскостью. Результирующий код не получится таким элегантным, но зато все массивы, кроме ISING, будут иметь порядок N^2 вместо N^3.

C.13.3. Перевод формул и обработка массивов

Многие математические формулы можно переводить прямо на Фортран, если использовать средства обработки массивов. Предположим, имеются следующие объявления массивов:

```
REAL X(N),A(M,N)
```

Несколько примеров математических формул и соответствующих фортрановских выражений приведены ниже.

C.13.3.1. Сумма произведений

Выражение

$$\sum_{j=1}^{N}\prod_{i=1}^{M} a_{ij}$$

может быть образовано с использованием фортрановского выражения

```
SUM (PRODUCT (A,DIM=1))
```

Аргумент DIM=1 означает, что произведение вычисляется по каждому столбцу A. Если A имеет значение

$$\begin{bmatrix} B & C & D \\ E & F & G \end{bmatrix}$$

результатом выражения будет B*E + C*F + D*G.

C.13.3.2. Произведение сумм

Выражение

$$\prod_{i=1}^{M}\sum_{j=1}^{N} a_{ij}$$

может быть сформировано при помощи фортрановского выражения

```
PRODUCT(SUM(A,DIM=2))
```

аргумент DIM=2 означает, что суммы должны вычисляться по каждой строке A. Если A имеет значение

$$\begin{bmatrix} B & C & D \\ E & F & G \end{bmatrix}$$

результатом этого выражения будет (B+C+D)(E+F+G).

C.13.3.3. Сложение выбираемых элементов

Выражение

$$\sum_{x_i>0.0} x_i$$

может быть сформировано с помощью фортрановского выражения

```
SUM(X,MASK=X.GT.0.0)
```

Маска выделяет положительные элементы одномерного массива. Если X имеет значение (0.0, -0.1, 0.2, 0.3, 0.2, -0.1, 0.0), результат выражения равен 0.7.

C.13.4. Сумма квадратичных отклонений

Выражение

$$\sum_{i=1}^{N}(x_i - x_{mean})^2$$

можно сформировать с использованием Фортрановских операторов:

```
XMEAN=SUM(X)/SIZE(X)
SS=SUM((X-XMEAN)**2)
```

Таким образом, SS есть сумма квадратичных отклонений.

C.13.5. Нормы вектора: норма-бесконечность и норма-один

Норма-бесконечность вектора X=(X(1), ..., X(N)) определяется как наибольшее из чисел ABS(X(1)), ..., ABS(X(N)) и имеет, таким образом, значение MAXVAL (ABS (X)).

Норма-один вектора X определяется как сумма чисел ABS (X (1)), ..., ABS (X (N)) и имеет, таким образом, значение SUM (ABS (X)).

С.13.6. Нормы матрицы: норма-бесконечность и норма-один

Норма-бесконечность матрицы A = (A(I, J)) есть наибольшая из сумм элементов каждой строки матрицы ABS(A(I, J)), значением которой является, следовательно, MAXVAL (SUM (ABS (A), DIM=2)).

Норма-один матрицы A = (A(I, J)) есть наибольшая из сумм элементов каждого столбца матрицы ABS (A(I, J)), значением которой, следовательно, является MAXVAL (SUM (ABS (A), DIM=1)).

С.13.7. Логические вопросы

Встроенные функции позволяют получать ответы на довольно сложные вопросы о табличных данных без использования циклов или условных конструкций. Рассмотрим, к примеру, вопросы по поводу простой таблицы оценок при тестировании студентов.

Предположим, что прямоугольная таблица T(M, N) содержит оценки M студентов, получивших N разных тестов. T - матрица целого типа с элементами в диапазоне от 0 до 100.

Пример: оценки трех студентов по четырем тестам представлены таблицей

$$T = \begin{bmatrix} 85 & 76 & 90 & 60 \\ 71 & 45 & 50 & 80 \\ 66 & 45 & 21 & 55 \end{bmatrix}$$

Вопрос: какая у каждого из студентов самая высокая оценка?

Ответ: MAXVAL(T, DIM=2); в примере: [90, 80, 66].

Вопрос: какая средняя оценка из числа всех оценок?

Ответ: SUM(T)/SIZE(T); в примере: 62.

Вопрос: сколько в таблице оценок выше средней?

Ответ: ABOVE = T .GT. SUM(T)/SIZE(T): N = COUNT (ABOVE); в примере ABOVE - логический массив (*t* = "истина", • = "ложь"):

$$\begin{bmatrix} t & t & t & \bullet \\ t & \bullet & \bullet & t \\ t & \bullet & \bullet & \bullet \end{bmatrix}$$

а COUNT(ABOVE) - 6.

Вопрос: какая самая низкая оценка среди превышающих среднюю?

Ответ: MINVAL(T, MASK=ABOVE), где ABOVE определено выше; в примере: 66.

Вопрос: был ли студент, чьи оценки все были выше средней?

Ответ: при определенном выше ABOVE ответ "да" или "нет" в зависимости от того, истинно или ложно значение выражения ANY (ALL (ABOVE, DIM=2)); в примере ответ: "нет".

С.13.8. Параллельные вычисления

Самым простым видом параллельной обработки является одновременное выполнение одного и того же действия над многими операндами. Очень хорошим примером такой простейшей формы параллельной обработки является сложение матриц. Таким образом, присваивание массиву A=B+C задает параллельное сложение соответствующих элементов массивов одной конфигурации B и C и параллельное присваивание результирующих сумм массиву A.

Процесс, выполняемый параллельно в примере со сложением матриц, является, конечно, процессом сложения. А средство работы с массивами, позволяющее так успешно реализовать сложение матриц как параллельный процесс, есть поэлементное вычисление выражений-массивов.

Эти наблюдения приводят нас к рассмотрению поэлементных вычислений как способу реализации других простых алгоритмов параллельной обработки.

C.13.9. Примеры поэлементного вычисления

Несколько полиномов одной степени можно вычислить в одной точке, расположив их коэффициенты как строки матрицы и применив к столбцам образованной таким образом матрицы метод Горнера для вычисления полинома.

Эта процедура иллюстрируется программой, вычисляющей три кубических полинома:

$$P(t) = 1 + 2t - 3t^2 + 4t^3$$

$$Q(t) = 2 - 3t + 4t^2 - 5t^3$$

$$R(t) = 3 + 4t - 5t^2 + 6t^3$$

параллельно в точке t=X и помещающей результирующий вектор из чисел [P(X),Q(X),R(X)] в вещественный массив RESULT(3).

Программа вычисления RESULT состоит из одного оператора:

```
RESULT = M(:, 1) + X * (M(:, 2) + X * (M(:,3) + X * M(:,4)))
```

где M представляет матрицу M(3,4) со значением

$$\begin{bmatrix} 1 & 2 & -3 & 4 \\ 2 & -3 & 4 & -5 \\ 3 & 4 & -5 & 6 \end{bmatrix}$$

C.13.10. Процедуры операций с битами

Процедуры IOR, IAND, NOT, IEOR, ISHFT, ISHFTC, IBITS, MVBITS, BTEST, IBSET и IBCLR определены в MIL-STD 1753 для скалярных аргументов и расширены в данном стандарте за счет аргументов-массивов и результирующих значений-массивов.

C.14. Примечания к главе 14

C.14.1. Связь по памяти для объектов нулевого размера

В результате изменения именованных констант в контексте связи по памяти могут встретиться объекты нулевого размера. Например, программа может содержать следующие объявления:

```
INTEGER, PARAMETER :: PROBSIZE=10
INTEGER, PARAMETER :: ARRAYSIZE=PROBSIZE*100
REAL, DIMENSION (ARRAYSIZE) ::X
INTEGER, DIMENSION (ARRAYSIZE) ::IX
...
COMMON /EXAMPLE/ A,B,C,X,Y,Z
EQUIVALENCE (X,IX)
...
```

Если в дальнейшем изменится первый оператор, устанавливающий PROBSIZE равным нулю, программа останется соответствующей стандарту.

ИЗДАТЕЛЬСТВО "ФИНАНСЫ И СТАТИСТИКА"

выпускает в 1998 г.

А. Саймон
СТРАТЕГИЧЕСКАЯ ТЕХНОЛОГИЯ БАЗ ДАННЫХ. Менеджмент на 2000 г.
Пер. с англ. – 30 л.

Alan R. Simon
STRATEGIC DATABASE TECHNOLOGY:
Management for the Year 2000. Morgan Kaufmann Publishers. Inc. San Francisco, California, 1995

Дается многоаспектный анализ ключевых областей современной массовой технологии баз данных, представлены основные стандарты, на которые она опирается, оцениваются ближайшие и долговременные перспективы ее развития.

Для руководителей и персонала информационных служб компаний, использующих или планирующих использовать системы баз данных; для разработчиков этих систем; специалистов компаний–поставщиков программного обеспечения и консультационных фирм; научных сотрудников, преподавателей и студентов вузов.

Книгу можно приобрести в издательстве по адресу:

101000, Москва, ул. Покровка, 7
(метро "Китай-город", выход на ул. Маросейка)

Тел.: **(095) 925-35-02, 923-18-68, 923-80-42**
Факс **(095) 925-09-57**

Приложение D
(информационное)

СИНТАКСИЧЕСКИЕ ПРАВИЛА

Это приложение содержит две части. Первую часть образуют все синтаксические правила и ограничения в порядке их вхождения в Международный стандарт. Вторая часть - перекрестные ссылки на входы всех терминальных и нетерминальных символов в синтаксические правила. Символы в алфавитном порядке рассортированы по трем категориям: нетерминальные символы, которые определены, нетерминальные символы, которые не определены, и терминальные символы.

Кроме тех, символов, которые начинаются с префикса "*имя-*", неопределяемыми нетерминальными символами являются только *буква, цифра, специальный-символ* и *представимый-символ*. Как это описано в 1.5.2, предполагаются следующие правила. Буквы "*xyz*" означают для каждой правильной фразы синтаксического класса:

список-xyz **есть** *xyz* [,*xyz*]...

имя-xyz **есть** *имя*

скалярное-xyz **есть** *xyz*

Ограничение: *скалярное-xyz* должно быть скаляром.

D.1. Синтаксические правила и ограничения

Каждый из последующих разделов содержит синтаксические правила и ограничения для одной из глав стандарта, т.е. раздел D.1.1 содержит правила и ограничения для главы 1 стандарта, раздел D.1.2 содержит то же для главы 2 и т.д. Заметим, что главы 1, 13 и 14 не содержат синтаксических правил.

D.1.1. Общие сведения

D.1.2. Термины и понятия Фортрана

R201 *выполняемая-программа* **есть** *программная-единица*
[*программная-единица*]...

R202 *программная-единица* **есть** *главная-программная-единица*
или *внешняя-процедура*
или *модуль*
или *блок-данных*

R1101 главная-программная-единица **есть** [*оператор-PROGRAM*]
[*раздел-спецификаций*]
[*раздел-выполняемых-операторов*]
[*раздел-внутренних-процедур*]
оператор-конца-программы

R203 *внешняя-процедура* **есть** *процедура-функция*
или *процедура-подпрограмма*

R1215 *процедура-функция* **есть** *оператор-FUNCTION*
[*раздел-спецификаций*]
[*раздел-выполняемых-операторов*]
[*раздел-внутренних-процедур*]
оператор-конца-функции

R1219 *процедура-подпрограмма* **есть** *оператор-SUBROUTINE*
[*раздел-спецификаций*]
[*раздел-выполняемых-операторов*]
[*раздел-внутренних-процедур*]
оператор-конца-подпрограммы

R1104 *модуль* **есть** *оператор-MODULE*
[*раздел-спецификаций*]
[*раздел-модульных-процедур*]
оператор-конца-модуля

R1110 *блок-данных* **есть** *оператор-BLOCK-DATA*
[*раздел-спецификаций*]
оператор-конца-блока-данных

R204 *раздел-спецификаций* **есть** [*оператор-USE*]...
[*раздел-IMPLICIT*]
[*конструкция-объявления*]...

R205 *раздел-IMPLICIT* **есть** [*оператор-раздела-IMPLICIT*]...
оператор-IMPLICIT

R206 *оператор-раздела-IMPLICIT* **есть** *оператор-IMPLICIT*
или *оператор-PARAMETER*
или *оператор-FORMAT*
или *оператор-ENTRY*

R207 *конструкция-объявления* **есть** *описание-производного-типа*
или *блок-интерфейса*
или *оператор-объявления-типа*
или *оператор-спецификации*
или *оператор-PARAMETER*
или *оператор-FORMAT*
или *оператор-ENTRY*
или *описание-операторной-функции*

R208 *раздел-выполняемых-операторов* **есть** *выполняемая-конструкция*
[*конструкция-раздела-выполняемых-операторов*]...

R209 *конструкция-раздела-выполняемых-операторов* ♦
♦ **есть** *выполняемая-конструкция*
или *оператор-FORMAT*
или *оператор-DATA*
или *оператор-ENTRY*

R210 *раздел-внутренних-процедур* **есть** *оператор-CONTAINS*
внутренняя-процедура
[*внутренняя-процедура*]...

R211 *внутренняя-процедура* **есть** *процедура-функция*
или *процедура-подпрограмма*

R212 *раздел-модульных-процедур* **есть** *оператор-CONTAINS*
модульная-процедура
[*модульная-процедура*]...

R213 *модульная-процедура* **есть** *процедура-функция*
или *процедура-подпрограмма*

R214 *оператор-спецификации* **есть** *оператор-доступности*
или *оператор-ALLOCATABLE*
или *оператор-COMMON*
или *оператор-DATA*
или *оператор-DIMENSION*
или *оператор-EQUIVALENCE*
или *оператор-EXTERNAL*
или *оператор-INTENT*
или *оператор-INTRINSIC*
или *оператор-NAMELIST*
или *оператор-OPTIONAL*
или *оператор-POINTER*
или *оператор-SAVE*
или *оператор-TARGET*

R215 *выполняемая-конструкция* **есть** *оператор-действия*
или *конструкция-CASE*
или *конструкция-DO*
или *конструкция-IF*
или *конструкция-WHERE*

R216 *оператор-действия* **есть** *оператор-ALLOCATE*
или *оператор-BACKSPACE*
или *оператор-CALL*
или *оператор-CLOSE*
или *оператор-CONTINUE*
или *оператор-CYCLE*
или *оператор-DEALLOCATE*
или *оператор-ENDFILE*
или *оператор-EXIT*
или *оператор-GOTO*
или *оператор-вычисляемый-GOTO*
или *оператор-IF*
или *оператор-INQUIRE*
или *оператор-NULLIFY*
или *оператор-OPEN*
или *оператор-PRINT*
или *оператор-READ*
или *оператор-RETURN*
или *оператор-REWIND*
или *оператор-STOP*
или *оператор-WHERE*
или *оператор-WRITE*
или *оператор-конца-программы*
или *оператор-конца-подпрограммы*
или *оператор-конца-функции*
или *оператор-присваивания*
или *оператор-присваивания-указателю*
или *арифметический-оператор-IF*
или *оператор-ASSIGN*
или *оператор-GOTO-по-предписанию*
или *оператор-PAUSE*

Ограничение: *раздел-выполняемых-операторов* не должен содержать *оператор-конца-программы*, *оператор-конца-подпрограммы* или *оператор-конца-функции*.

D.1.3. Символы, лексические элементы и форма исходного текста

R301 *символ* **есть** *буквенно-цифровой-символ*
или *специальный-символ*

R302 *буквенно-цифровой-символ* **есть** *буква*
или *цифра*
или *подчеркивание*

R303 *подчеркивание* **есть** _

R304 *имя* **есть** *буква* [*буквенно-цифровой-символ*]...

Ограничение: максимальная длина *имени* - 31 символ.

R305 *константа* **есть** *литеральная-константа*
или *именованная-константа*

R306 *литеральная-константа* **есть** *целая-литеральная-константа*
или *вещественная-литеральная-константа*
или *комплексная-литеральная-константа*
или *логическая-литеральная-константа*

или *символьная-литеральная-константа*
или BOZ-*литеральная-константа*

R307 *именованная-константа* **есть** *имя*

R308 *целая-константа* **есть** *константа*

Ограничение: *целая-константа* должна быть целого типа.

R309 *символьная-константа* **есть** *константа*

Ограничение: *символьная-константа* должна быть символьного типа.

R310 *знак-встроенной-операции* **есть** *оп-степени*
или *оп-умножения*
или *оп-сложения*
или *оп-конкатенации*
или *оп-отношения*
или *оп-отрицания*
или *оп-коньюнкции*
или *оп-дизъюнкции*
или *оп-эквивалентности*

R708 *оп-степени* **есть** **

R709 *оп-умножения* **есть** *
или /

R710 *оп-сложения* **есть** +
или -

R712 *оп-конкатенации* **есть** //

R714 *оп-отношения* **есть** .EQ.
или .NE.
или .LT.
или .LE.
или .GT.
или .GE.
или ==
или /=
или <
или <=
или >
или >=

R719 *оп-отрицания* **есть** .NOT.

R720 *оп-дизъюнкции* **есть** .AND.

R721 *оп-коньюнкции* **есть** .OR.

R722 *оп-эквивалентности* **есть** .EQV.
или .NEQV.

R311 *знак-определяемой-операции* **есть** *знак-определяемой-унарной-операции*
или *знак-определяемой-бинарной-операции*
или *знак-расширенной-встроенной-операции*

R704 *знак-определяемой-унарной-операции* **есть** . *буква*[*буква*]... .

R724 *знак-определяемой-бинарной-операции* **есть** . *буква*[*буква*]... .

R312 *знак-расширенной-встроенной-операции* **есть** *знак-встроенной-операции*

Ограничение: *знак-определяемой-унарной-операции* и *знак-определяемой-бинарной-операции* не должны содержать больше 31 символа и не могут совпадать со знаком какой-либо встроенной операции или с *логической-литеральной-константой*.

R313 *метка* **есть** *цифра*[*цифра*[*цифра*[*цифра*[*цифра*]]]]

Ограничение: по крайней мере одна цифра в метке должна быть отличной от нуля.

D.1.4. Встроенные и производные типы данных

R401 *строка-цифр-со-знаком* **есть** [*знак*] *строка-цифр*

R402 *строка-цифр* **есть** *цифра* [*цифра*]...

R403 *целая-литеральная-константа-со-знаком* **есть** [*знак*] ♦
♦ *целая-литеральная-константа*

R404 *целая-литеральная-константа* **есть** *строка-цифр* [_*параметр-вида*]

R405 *параметр-вида* **есть** *строка-цифр*
или *имя-скалярной-целой-константы*

R406 *знак* **есть** +
или -

Ограничение: значение *параметра-вида* должно быть неотрицательным.

Ограничение: значение *параметра-вида* должно задавать существующий в процессоре способ представления.

R407 **BOZ**-*литеральная-константа* **есть** *двоичная-константа*
или *восьмеричная-константа*
или *шестнадцатеричная-константа*

Ограничение: **BOZ**-*литеральная-константа* может находиться только в операторе DATA.

R408 *двоичная-константа* **есть** B'*цифра* [*цифра*]...'
или B"*цифра* [*цифра*]..."

Ограничение: *цифра* может иметь значения только 0 или 1.

R409 *восьмеричная-константа* **есть** O'*цифра* [*цифра*]...'
или O"*цифра* [*цифра*]..."

Ограничение: *цифры* могут иметь значения от 0 до 7.

R410 *шестнадцатеричная-константа* ♦
♦ **есть** Z'*шестнадцатеричная-цифра* [*шестнадцатеричная-цифра*]...'
или Z"*шестнадцатеричная-цифра* [*шестнадцатеричная-цифра*]..."

R411 *шестнадцатеричная-цифра* **есть** *цифра*
или A
или B
или C
или D
или E
или F

R412 *вещественная-литеральная-константа-со-знаком* **есть** [*знак*] ♦
♦ *вещественная-литеральная-константа*

R413 *вещественная-литеральная-константа* **есть** *значащая-часть* ♦
♦ [*буква-экспоненты экспонента*] [_*параметр-вида*]
или *строка-цифр* ♦
♦ *буква-экспоненты экспонента* [_*параметр-вида*]

R414 *значащая-часть* **есть** *строка-цифр*. [*строка-цифр*]
или .*строка-цифр*

R415 *буква-экспоненты* **есть** E
или D

R416 *экспонента* **есть** *строка-цифр-со-знаком*

Ограничение: если присутствуют и *параметр-вида* и *буква-экспоненты*, буквой экспоненты должна быть E.

Ограничение: значение *параметра-вида* должно задавать способ аппроксимации, существующий в процессоре.

R417 *комплексная-литеральная-константа* **есть** (*действит-часть*, *мнимая-часть*)

R418 *действит-часть* **есть** *целая-литеральная-константа-со-знаком*
или *вещественная-литеральная-константа-со-знаком*

R419 *мнимая-часть* **есть** *целая-литеральная-константа-со-знаком*
или *вещественная-литеральная-константа-со-знаком*

R420 *символьная-литеральная-константа* ♦
♦ **есть** [*параметр-вида_*] '[*представимый-символ*]...'
или [*параметр-вида_*] "[*представимый-символ*]..."

Ограничение: значение *параметра-вида* должно задавать способ представления, существующий в процессоре.

R421 *логическая-литеральная-константа* **есть** .TRUE. [*_параметр-вида*]
или .FALSE. [*_параметр-вида*]

Ограничение: значение *параметра-вида* должно задавать способ представления, существующий в процессоре.

R422 *описание-производного-типа* **есть** *оператор-заголовка-типа*
[*оператор-PRIVATE-SEQUENCE*]...
оператор-описания-компонента
[*оператор-описания-компонента*]...
оператор-конца-типа

R423 *оператор-PRIVATE-SEQUENCE* **есть** PRIVATE
или SEQUENCE

R424 *оператор-заголовка-типа* **есть** TYPE [,*специф-доступности*] [::] *имя-типа*

Ограничение: один и тот же *оператор-PRIVATE-SEQUENCE* не может появиться более одного раза в одном *описании-производного-типа*.

Ограничение: если есть ключевое слово SEQUENCE, все производные типы, задаваемые в описаниях компонентов, должны быть последовательными типами.

Ограничение: *специф-доступности* (5.1.2.2) или оператор PRIVATE в описании типа разрешены только в том случае, когда описание типа находится в разделе спецификаций модуля.

Ограничение: если компонент производного типа относится к типу, объявленному как приватный, то либо описание типа должно содержать оператор PRIVATE, либо производный тип должен быть приватным.

Ограничение: *имя-типа* не должно совпадать ни с одним из имен встроенных типов, ни с именем другого доступного производного типа.

R425 *оператор-конца-типа* **есть** END TYPE [*имя-типа*]

Ограничение: если за END TYPE следует *имя-типа*, оно должно совпадать с именем в соответствующем *операторе-заголовка-типа*.

R426 *оператор-описания-компонента* **есть** *специф-типа* ♦
♦ [[,(*список-специф-атрибутов-компонентов*)][::] ♦
♦ *список-объявлений-компонентов*

R427 *специф-атрибутов-компонента* **есть** POINTER
или DIMENSION (*специф-компонента-массива*)

Ограничение: ни один *специф-атрибутов-компонента* не может входить в один *оператор-описания-компонента* более одного раза.

Ограничение: если для компонента не задан атрибут POINTER, *специф-типа* в *операторе-описания-компонента* должен специфицировать встроенный или предварительно описанный производный тип.

Ограничение: если для компонента задан атрибут POINTER, *специф-типа* в *операторе-описания-компонента* должен специфицировать встроенный тип или любой доступный производный тип, включая тот, который описывается.

R428 *специф-компонента-массива* **есть** *список-специф-явной-конфигурации*
или *список-специф-определяемой-конфигурации*

R429 *объявление-компонента* **есть** *имя-компонента* ♦
♦ [(*специф-компонента-массива*)][* *длина-символьного*]

Ограничение: если атрибут POINTER не специфицирован, каждый *специф-компонента-массива* должен быть *списком-специф-явной-конфигурации*.

Ограничение: если атрибут POINTER специфицирован, каждый *специф-компонента-массива* должен быть *списком-специф-определяемой-конфигурации*.

Ограничение: спецификация "* *длина-символьного* " разрешена, только если специфицируемый тип является символьным.

Ограничение: *длина-символьного* в *описании-компонента* должна быть константным выражением спецификации (7.1.6.2).

Ограничение: каждая граница в *специф-явной-конфигурации* (R428) должна быть константным выражением спецификации (7.1.6.2).

R430 *конструктор-структуры* **есть** *имя-типа (список-выражений)*

R431 *конструктор-массива* **есть** (/*список-значений-констр-массива*/)

R432 *значение-констр-массива* **есть** *выражение*
или *неявный-цикл-констр-массива*

R433 *неявный-цикл-констр-массива* **есть** (*список-значений-констр-* ♦
♦ *массива, управление-нцкл-констр-массива*)

R434 *управление-нцкл-констр-массива* **есть** *переменная-нцкл-* ♦
♦ *констр-массива* = *скалярное-целое-выражение*, ♦
♦ *скалярное-целое-выражение* [,*скалярное-целое-выражение*]

R435 *переменная-нцкл-констр-массива* **есть** *скалярная-целая-переменная*

Ограничение: *переменная-нцкл-констр-массива* должна быть именованной переменной.

Ограничение: все выражения *значения-констр-массива* в конструкторе массива должны иметь одинаковый тип и одинаковые параметры типа.

D.1.5. Спецификации и описания объектов данных

R501 *оператор-объявления-типа* **есть** *специф-типа* ♦
♦ [[,*специф-атрибута*] ... ::] *список-объявлений-объектов*

R502 *специф-типа* **есть** INTEGER [*селектор-вида*]
или REAL [*селектор-вида*]
или DOUBLE PRECISION
или COMPLEX [*селектор-вида*]
или CHARACTER [*селектор-символьного*]
или LOGICAL [*селектор-вида*]
или TYPE (*имя-типа*)

R503 *специф-атрибута* **есть** PARAMETER
или *специф-доступности*
или ALLOCATABLE
или DIMENSION (*специф-массива*)
или EXTERNAL
или INTENT (*специф-назначения*)
или INTRINSIC
или OPTIONAL
или POINTER
или SAVE
или TARGET

R504 *объявление-объекта* **есть** *имя-объекта* [(*специф-массива*)] ♦
♦ [* *длина-символьного*] [= *выражение-инициализации*]
или *имя-функции* [(*специф-массива*)] [*длина-символьного*]

R505 *селектор-вида* **есть** ([KIND=] *скалярное-целое-выражение-инициализации*)

Ограничение: ни один *специф-атрибута* не должен входить более одного раза в один и тот же *оператор-объявления-типа*.

Ограничение: *имя-функции* должно быть именем внешней, встроенной, формальной или операторной функции.

Ограничение: конструкция "= *выражение-инициализации*" должна присутствовать, когда оператор содержит атрибут PARAMETER (5.1.2.1).

Ограничение: если присутствует "= *выражение-инициализации*", то перед *списком-объявлений-объектов* должен присутствовать разделитель :: .

Ограничение: если *имя-объекта* - формальный аргумент, результат функции, объект из именованного общего блока (при условии, что объявление типа данных не находится в программной единице - блоке данных), объект из неименованного блока данных, динамически размещаемый массив, указатель, имя внешней или встроенной процедуры, либо автоматически размещаемый объект - то конструкция "= *выражение-инициализации*" должна отсутствовать.

Ограничение: спецификация "* *длина-символьного*" разрешена только при объявлении данных символьного типа.

Ограничение: атрибут ALLOCATABLE можно использовать только при условии, что объявляемый массив не является формальным массивом или результатом функции.

Ограничение: для массива, объявленного с атрибутом POINTER или ALLOCATABLE, *специф-массива* должен иметь вид: *список-специф-определяемой-конфигурации* (5.1.2.4.3).

Ограничение: *специф-массива* для *имени-функции*, не имеющего атрибута POINTER, должен иметь *вид: список-специф-явной-конфигурации*.

Ограничение: *специф-массива* для *имени-функции*, имеющего атрибут POINTER, должен иметь вид: *список-специф-определяемой-конфигурации*.

Ограничение: если присутствует атрибут POINTER, нельзя задавать атрибуты TARGET, INTENT, EXTERNAL и INTRINSIC.

Ограничение: если присутствует атрибут TARGET, нельзя задавать атрибуты POINTER, EXTERNAL, INTRINSIC и PARAMETER.

Ограничение: атрибут PARAMETER нельзя задавать для формальных аргументов, указателей, функций, динамически размещаемых массивов и объектов из общего блока.

Ограничение: атрибуты INTENT и OPTIONAL можно задавать только для формальных аргументов.

Ограничение: объект не должен иметь атрибут PUBLIC, если его тип имеет атрибут PRIVATE.

Ограничение: атрибут SAVE нельзя задавать для объекта из общего блока, формального аргумента, процедуры, результата функции и автоматически размещаемого объекта.

Ограничение: объект не должен иметь одновременно и атрибут EXTERNAL, и атрибут INTRINSIC.

Ограничение: объект в *операторе-объявления-типа* не может иметь ни атрибута EXTERNAL, ни атрибута INTRINSIC, если он не является функцией.

Ограничение: массив не должен одновременно иметь и атрибут ALLOCATABLE, и атрибут POINTER.

Ограничение: ни один из атрибутов объекта нельзя задать более одного раза в одной контекстной области.

Ограничение: значение, заданное в *скалярном-целом-выражении-инициализации*, должно быть неотрицательным и должно специфицировать способ представления, существующий в процессоре.

R'06 *селектор-символьного* **есть** *селектор-длины*
или (LEN = *значение-параметра-типа*, ♦
♦ KIND = *скалярное-целое-выражение-инициализации*)
или (*значение-параметра-типа*, ♦
♦ [KIND=] *скалярное-целое-выражение-инициализации*)

или (KIND = *скалярное-целое-выражение-инициализации* ♦
♦ [,LEN = *значение-параметра-типа*]

R507 *селектор-длины* **есть** ([LEN=]*значение-параметра-типа*)
или * *длина-символьного* [,]

R508 *длина-символьного* **есть** (*значение-параметра-типа*)
или *скалярная-целая-литеральная-константа*

Ограничение: необязательная запятая в *селекторе-длины* разрешена только в *специф-типа* в *операторе-объявления-типа*.

Ограничение: необязательная запятая в *селекторе-длины* может появиться только тогда, когда в *операторе-объявления-типа* отсутствует разделитель ::.

Ограничение: значение *скалярного-целого-выражения-инициализации* должно быть неотрицательным и должно специфицировать способ представления, существующий в процессоре.

Ограничение: *скалярная-целая-литеральная-константа* не должна включать *параметр-вида*.

R509 *значение-параметра-типа* **есть** *выражение-спецификации*
или *

Ограничение: имя функции нельзя объявлять со *значением-параметра-типа* в виде звездочки, если функция является внутренней, модульной, рекурсивной или ее значением является массив или указатель.

R510 *специф-доступности* **есть** PUBLIC
или PRIVATE

Ограничение: атрибут *специф-доступности* может находиться только в контекстной области модуля.

R511 *специф-назначения* **есть** IN
или OUT
или INOUT

Ограничение: нельзя специфицировать атрибут INTENT для формального аргумента, который является формальной процедурой или формальным указателем.

R512 *специф-массива* **есть** *список-специф-явной-конфигурации*
или *список-специф-неопределенной-конфигурации*
или *список-специф-определяемой-конфигурации*
или *специф-неопределенного-размера*

Ограничение: максимальная размерность равна семи.

R513 *специф-явной-конфигурации* **есть** [*нижняя-граница* :] *верхняя-граница*

R514 *нижняя-граница* **есть** *выражение-спецификации*

R515 *верхняя-граница* **есть** *выражение-спецификации*

Ограничение: массив явной конфигурации, границы которого зависят от значений неконстантных выражений, должен быть либо формальным аргументом, либо результатом функции, либо массивом, автоматически размещаемым в процедуре.

R516 *специф-неопределенной-конфигурации* **есть** [*нижняя-граница*] :

R517 *специф-определяемой-конфигурации* **есть** :

R518 *специф-неопределенного-размера* **есть** [*список-специф-явной-* ♦
♦ *конфигурации*,] [*нижняя-граница* :] *

Ограничение: имя функции, значением которой является массив, не должно быть объявлено как массив неопределенного размера.

R519 *оператор-INTENT* **есть** INTENT(*специф-назначения*) [::] ♦
♦ *список-имен-формальных-аргументов*

Ограничение: *оператор-INTENT* может находиться только в *разделе-спецификаций* процедуры или тела интерфейса (12.3.2.1).

Ограничение: *имя-формального-аргумента* не должно быть именем формальной процедуры или формального аргумента-указателя.

R520 *оператор-OPTIONAL* **есть** OPTIONAL [::] ♦
♦ *список-имен-формальных-аргументов*

Ограничение: *оператор-OPTIONAL* может находиться только в контекстной области процедуры или тела интерфейса.

R521 *оператор-доступности* **есть** *специф-доступности* [[::] ♦
♦ *список-идентификаторов-доступности*]

R522 *идентификатор-доступности* **есть** *имя-для-USE*
или *обобщенный-специф*

Ограничение: *оператор-доступности* может находиться только в контекстной области модуля. Только один оператор доступности без *списка-идентификаторов-доступности* разрешен в контекстной области модуля.

Ограничение: каждое *имя-для-USE* должно быть именем переменной, процедуры, производного типа, группы NAMELIST или именованной константы.

Ограничение: модульная процедура, формальный аргумент или результат которой принадлежит к типу с атрибутом доступности PRIVATE, должна иметь атрибут доступности PRIVATE и не должна иметь обобщенный идентификатор с атрибутом доступности PUBLIC.

R523 *оператор-SAVE* **есть** SAVE [[::] *список-сохраняемых-объектов*]

R524 *сохраняемый-объект* **есть** *имя-объекта*
или /*имя-общего-блока* /

Ограничение: *имя-объекта* не должно быть именем формального аргумента, процедуры, результата функции, автоматически размещаемого объекта или именем объекта из общего блока.

Ограничение: если в контекстной области есть оператор SAVE без списка сохраняемых объектов, все другие вхождения оператора или атрибута SAVE в этой контекстной области запрещены.

R525 *оператор-DIMENSION* **есть** DIMENSION [::] *имя-массива* ♦
♦ (*специф-массива*) [,*имя-массива* (*специф-массива*)]...

R526 *оператор-ALLOCATABLE* **есть** ALLOCATABLE [::] ♦
♦ *имя-массива* [(*список-специф-определяемой-конфигурации*)] ♦
♦ [,*имя-массива*[(*список-специф-определяемой-конфигурации*)]]...

Ограничение: *имя-массива* не должно быть формальным аргументом или результатом функции.

Ограничение: если где-нибудь в контекстной области специфицируется атрибут DIMENSION для *имени-массива*, *спецификатор-массива* должен быть *списком-спецификаторов-определяемой-конфигурации*.

R527 *оператор-POINTER* **есть** POINTER [::] ♦
♦ *имя-объекта* [(*список-специф-определяемой-конфигурации*)] ♦
♦ [,*имя-объекта*[(*список-специф-определяемой-конфигурации*)]]...

Ограничение: для *имени-объекта* не должен быть специфицирован атрибут INTENT.

Ограничение: если для *имени-объекта* где-нибудь в контекстной области специфицируется атрибут DIMENSION, *спецификатор-массива* должен быть *списком-спецификаторов-определяемой-конфигурации*.

Ограничение: для *имени-объекта* не должен быть специфицирован атрибут PARAMETER.

R528 *оператор-TARGET* **есть** TARGET [::] ♦
♦ *имя-объекта* [(*специф-массива*)] ♦
♦ [,*имя-объекта* [(*специф-массива*)]]...

Ограничение: для *имени-объекта* не должен быть специфицирован атрибут PARAMETER.

R529 *оператор-DATA* **есть** DATA *множество-оператора-DATA* ♦
♦ [[,] *множество-оператора-DATA*]...

R530 *множество-оператора-DATA* **есть** *список-объектов-* ♦
♦ *оператора-DATA* /*список-значений-оператора-DATA*/

R531 *объект-оператора-DATA* **есть** *переменная*
или *неявный-цикл-DATA*

R532 *значение-оператора-DATA* **есть** [*повторитель-оператора-DATA**] ♦
♦ *константа-оператора-DATA*

R533 *константа-оператора-DATA* **есть** *скалярная-константа*
или *целая-литеральная-константа-со-знаком*
или *вещественная-литеральная-константа-со-знаком*
или *конструктор-структуры*
или *BOZ-литеральная-константа*

R534 *повторитель-оператора-DATA* **есть** *скалярная-целая-константа*

R535 *неявный-цикл-DATA* **есть** (*список-объектов-нцкл-DATA*, ♦
♦ *переменная-нцкл-DATA* = *скалярное-целое-выражение*, ♦
♦ *скалярное-целое-выражение* [,*скалярное-целое-выражение*])

R536 *объект-нцкл-DATA* **есть** *элемент-массива*
или *скалярный-компонент-структуры*
или *неявный-цикл-DATA*

Ограничение: *элемент-массива* не должен иметь в качестве основы константу.

Ограничение: *скалярный-компонент-структуры* не должен иметь в качестве основы константу.

R537 *переменная-нцкл-DATA* **есть** *скалярная-целая-переменная*

Ограничение: *переменная-нцкл-DATA* должна быть именованной переменной.

Ограничение: повторитель оператора DATA должен быть положительным или нулевым. Если повторитель - именованная константа, он должен быть предварительно объявлен в этой контекстной области или быть доступным через связь по объемлющему контексту или использованию.

Ограничение: если *константа-оператора-DATA* является *конструктором-структуры*, каждый его компонент должен быть выражением инициализации.

Ограничение: в переменной, которая является *объектом-оператора-DATA*, все индексы, индексы секции, начальные и конечные точки подстроки должны быть выражениями инициализации.

Ограничение: переменная, имя или обозначение которой включены в *список-объектов-оператора-DATA* или *список-объектов-нцкл-DATA*, не должна быть формальным аргументом, объектом, доступ к которому реализован через связь по использованию или по объемлющему контексту, объектом из именованного общего блока (если только оператор DATA не находится в блоке данных), объектом из неименованного общего блока, именем функции, именем результата функции, автоматически размещаемым объектом, указателем или динамически размещаемым массивом.

Ограничение: все индексы в *элементе-массива* или *скалярном-компоненте-структуры*, которые являются *объектами-нцкл-DATA*, должны быть выражениями, в которых все первичные - только константы или переменные объемлющих неявных циклов, а все операции - встроенные.

Ограничение: *скалярное-целое-выражение* в неявном цикле должно содержать в качестве первичных только константы или переменные объемлющих *неявных-циклов-DATA*, а каждая операция должна быть встроенной.

R538 *оператор-PARAMETER* **есть** PARAMETER ♦
♦ (*список-описаний-именованных-констант*)

R539 *описание-именованной-константы* **есть** *именованная-константа* ♦
♦ = *выражение-инициализации*

R540 *оператор-IMPLICIT* **есть** IMPLICIT *список-специф-IMPLICIT*
или IMPLICIT NONE

R541 *специф-IMPLICIT* **есть** *специф-типа* (*список-специф-буквы*)

R542 *специф-буквы* **есть** *буква* [- *буква*]

Ограничение: если в контекстной области есть оператор IMPLICIT NONE, он должен предшествовать всем операторам PARAMETER, которые находятся в этой же контекстной области, и в этой контекстной области не должно быть других операторов IMPLICIT.

Ограничение: если есть минус и вторая буква, вторая буква должна следовать за первой в алфавитном порядке.

R543 *оператор-NAMELIST* **есть** NAMELIST ♦
♦ /*имя-группы-NAMELIST*/ *список-объектов-группы* ♦
♦ [[,]/*имя-группы-NAMELIST*/ *список-объектов-группы*]...

R544 *объект-группы* **есть** *имя-переменной*

Ограничение: *объект-группы* не должен быть формальным аргументом-массивом с неконстантной границей, переменной с неконстантной длиной (для символьных), автоматически размещаемым объектом, указателем, переменной, тип которой имеет указатель в качестве конечного компонента или динамически размещаемым массивом.

Ограничение: если *имя-группы-NAMELIST* имеет атрибут PUBLIC, ни один из элементов *списка-объектов-группы* не может иметь атрибут PRIVATE.

R545 *оператор-EQUIVALENCE* **есть** EQUIVALENCE ♦
♦ *список-множеств-эквивалентности*

R546 *множество-эквивалентности* **есть** (*эквивалентный-объект*, ♦
♦ *список-эквивалентных-объектов*)

R547 *эквивалентный-объект* **есть** *имя-переменной*
или *элемент-массива*
или *подстрока*

Ограничение: эквивалентный объект не должен быть формальным аргументом, указателем, динамически размещаемым массивом, объектом непоследовательного производного типа или последовательного производного типа, содержащего указатель на любом уровне выбора компонентов, автоматически размещаемым объектом, именем функции, именем входа или результата, именованной константой, компонентом структуры, либо же подобъектом любого из перечисленных выше объектов.

Ограничение: каждое индексное выражение или выражение диапазона подстроки в эквивалентном объекте должно быть целым выражением инициализации (7.1.6.1).

Ограничение: если *эквивалентный-объект* относится к стандартному целому, стандартному вещественному, вещественному двойной точности, стандартному комплексному, стандартному логическому или к числовому последовательному типу, все объекты в множестве эквивалентности должны относиться к тем же типам.

Ограничение: если *эквивалентный-объект* относится к стандартному символьному или к символьному последовательному типу, все объекты в множестве эквивалентности должны принадлежать к тем же типам.

Ограничение: если *эквивалентный-объект* принадлежит к производному типу, который не является ни числовым последовательным, ни символьным последовательным, все объекты в множестве эквивалентности должны относиться к этому же типу.

Ограничение: если *эквивалентный-объект* относится к встроенному типу, который не является ни стандартным целым, ни стандартным вещественным, ни вещественным двойной точности, ни стандартным комплексным, ни

стандартным логическим, ни стандартным символьным, все объекты в множестве эквивалентности должны быть того же типа и с теми же значениями параметров вида.

R548 *оператор-COMMON* **есть** COMMON [/[*имя-общего-блока*]/] ♦
♦ *список-объектов-общего-блока* ♦
♦ [[,]/[*имя-общего-блока*] /*список-объектов-общего-блока*]...

R549 *объект-общего-блока* **есть** *имя-переменной* ♦
♦ [(*список-специф-явной-конфигурации*)]

Ограничение: только одно вхождение *имени-переменной* разрешено во все *списки-объектов-общего-блока* в контекстной области.

Ограничение: *объект-общего-блока* не должен быть формальным аргументом, автоматически размещаемым объектом, динамически размещаемым массивом, именем функции, именем входа или именем результата.

Ограничение: каждая граница в *спецификаторе-явной-конфигурации* должна быть константным выражением спецификации (7.1.6.2).

Ограничение: если *объект-общего-блока* - производного типа, тип должен быть последовательным (4.4.1).

Ограничение: если *имя-переменной* сопровождается *списком-спецификаторов-явной-конфигурации*, оно не должно иметь атрибута POINTER.

D.1.6. Использование объектов данных

R601 *переменная* **есть** *имя-скалярной-переменной*
или *имя-переменной-массива*
или *подобъект*

Ограничение: *имя-переменной-массива* должно быть именем объекта данных, являющегося массивом.

Ограничение: *имя-переменной-массива* не должно иметь атрибута PARAMETER.

Ограничение: *имя-скалярной-переменной* не должно иметь атрибута PARAMETER.

Ограничение: *подобъект* не должен быть обозначением подобъекта (например, подстроки), основой которого является константа.

R602 *подобъект* **есть** *элемент-массива*
или *секция-массива*
или *компонент-структуры*
или *подстрока*

R603 *логическая-переменная* **есть** *переменная*

Ограничение: *логическая-переменная* должна быть логического типа.

R604 *переменная-станд-логич-типа* **есть** *переменная*

Ограничение: *переменная-станд-логич-типа* должна быть переменной стандартного логического типа.

R605 *символьная-переменная* **есть** *переменная*

Ограничение: *символьная-переменная* должна быть символьного типа.

R606 *переменная-станд-симв-типа* **есть** *переменная*

Ограничение: *переменная-станд-симв-типа* должна быть переменной стандартного символьного типа.

R607 *целая-переменная* **есть** *переменная*

Ограничение: *целая-переменная* должна быть целого типа.

R608 *переменная-станд-целого-типа* **есть** *переменная*

Ограничение: *переменная-станд-целого-типа* должна быть переменной стандартного целого типа.

R609 *подстрока* **есть** *строка-основа* (*диапазон-подстроки*)

R610 *строка-основа* **есть** *имя-скалярной-переменной*
или *элемент-массива*
или *скалярный-компонент-структуры*
или *скалярная-константа*

R611 *диапазон-подстроки* **есть** [*скалярное-целое-выражение*] : ♦
♦ [*скалярное-целое-выражение*]

Ограничение: *строка-основа* должна быть символьного типа.

R612 *ссылка-на-данное* **есть** *ссылка-на-часть* [% *ссылка-на-часть*]...

R613 *ссылка-на-часть* **есть** *имя-части* [(*список-индексов-секции*)]

Ограничение: в *ссылке-на-данное* каждое *имя-части*, кроме самого правого, должно быть производного типа.

Ограничение: в *ссылке-на-данное* каждое *имя-части*, кроме самого левого, должно быть именем компонента из описания производного типа для предшествующего *имени-части*.

Ограничение: в *ссылке-на-часть*, содержащей *список-индексов-секции*, число *индексов-секции* должно быть равно размерности *имени-части*.

Ограничение: в *ссылке-на-данное* должно быть не более одной *ссылки-на-часть* с ненулевой размерностью. *Имя-части*, стоящее справа в *ссылке-на-часть* с ненулевой размерностью, не должно иметь атрибута POINTER.

R614 *компонент-структуры* **есть** *ссылка-на-данное*

Ограничение: в *компоненте-структуры* должно быть более одной *ссылки-на-часть* и самая правая из них должна иметь вид: *имя-части*.

R615 *элемент-массива* **есть** *ссылка-на-данное*

Ограничение: в *элементе-массива* каждая *ссылка-на-часть* должна иметь нулевую размерность, а последняя *ссылка-на-часть* должна содержать *список-индексов*.

R616 *секция-массива* **есть** *ссылка-на-данное* [(*диапазон-подстроки*)]

Ограничение: в *секции-массива* ровно одна *ссылка-на-часть* должна иметь ненулевую размерность и либо последняя *ссылка-на-часть* имеет *список-индексов-секции* с ненулевой размерностью, либо другая *ссылка-на-часть* имеет ненулевую размерность.

Ограничение: в *секции-массива* с *диапазоном-подстроки* самое правое *имя-части* должно быть символьного типа.

R617 *индекс* **есть** *скалярное-целое-выражение*

R618 *индекс-секции* **есть** *индекс*
или *индексный-триплет*
или *векторный-индекс*

R619 *индексный-триплет* **есть** [*индекс*]:[*индекс*] [:*шаг*]

R620 *шаг* **есть** *скалярное-целое-выражение*

R621 *векторный-индекс* **есть** *целое-выражение*

Ограничение: *векторный-индекс* должен быть выражением-массивом размерности 1.

Ограничение: нельзя опускать второй *индекс* в *индексном-триплете* последнего измерения массива неопределенного размера.

R622 *оператор-ALLOCATE* **есть** ALLOCATE (*список-размещений* ♦
♦ [,STAT=*переменная-состояния*])

R623 *переменная-состояния* **есть** *скалярная-целая-переменная*

R624 *размещение* **есть** *размещаемый-объект* ♦
♦ [(*список-спецификаций-размещения*)]

R625 *размещаемый-объект* **есть** *имя-переменной*
или *компонент-структуры*

R626 *спецификация-размещения* **есть** [*нижняя-граница-размещения* :] ♦
♦ *верхняя-граница-размещения*

R627 *нижняя-граница-размещения* **есть** *скалярное-целое-выражение*

R628 *верхняя-граница-размещения* **есть** *скалярное-целое-выражение*

Ограничение: каждый *размещаемый-объект* должен быть указателем или динамически размещаемым массивом.

Ограничение: число *спецификаций-размещения* в *списке-спецификаций-размещения* должно совпадать с размерностью указателя или динамически размещаемого массива.

R629 *оператор-NULLIFY* **есть** NULLIFY (*список-объектов-указателей*)

R630 *объект-указатель* **есть** *имя-переменной*
или *компонент-структуры*

Ограничение: каждый *объект-указатель* должен иметь атрибут POINTER

R631 *оператор-DEALLOCATE* **есть** DEALLOCATE ♦
♦ (*список-размещаемых-объектов* [,STAT=*переменная-состояния*])

Ограничение: каждый *размещаемый-объект* должен быть указателем или динамически размещаемым массивом.

D.1.7. Выражения и присваивание

R701 *первичное* **есть** *константа*
или *константа-подобъект*
или *переменная*
или *конструктор-массива*
или *конструктор-структуры*
или *ссылка-на-функцию*
или *(выражение)*

R702 *константа-подобъект* **есть** *подобъект*

Ограничение: *подобъект* должен быть обозначением подобъекта, основой которого является константа.

Ограничение: *переменная* в качестве *первичного* не должна быть массивом неопределенного размера.

R703 *выражение-1-уровня* **есть** ♦
♦ [*знак-определяемой-унарной-операции*] *первичное*

R704 *знак-определяемой-унарной-операции* **есть** .*буква*[*буква*]... .

Ограничение: *знак-определяемой-унарной-операции* должен содержать не более 31 буквы и не должен совпадать с каким-либо из *знаков-встроенных-операций* или с одной из *логических-литеральных-констант*.

R705 *множитель* **есть** *выражение-1-уровня* [*оп-степени множитель*]

R706 *слагаемое* **есть** [*слагаемое оп-умножения*] *множитель*

R707 *выражение-2-уровня* **есть** [[*выражение-2-уровня*] *оп-сложения*] *слагаемое*

R708 *оп-степени* **есть** **

R709 *оп-умножения* **есть** *
или /

R710 *оп-сложения* **есть** +
или -

R711 *выражение-3-уровня* **есть** [*выражение-3-уровня оп-конкатенации*] ♦
♦ *выражение-2-уровня*

R712 *оп-конкатенации* **есть** //

R713 *выражение-4-уровня* **есть** [*выражение-3-уровня оп-отношения*] ♦
♦ *выражение-3-уровня*

R714 *оп-отношения* **есть** .EQ. **или** ==
или .NE. **или** /=
или .LT. **или** <

или .LE. **или** <=
или .GT. **или** >
или .GE. **или** >=

R715 *логический-множитель* **есть** [*оп-отрицания*] *выражение-4-уровня*

R716 *логическое-слагаемое* **есть** [*логическое-слагаемое* ♦ ♦ *оп-конъюнкции*] *логический-множитель*

R717 *операнд-эквивалентности* **есть** [*операнд-эквивалентности* ♦ ♦ *оп-дизъюнкции*] *логическое-слагаемое*

R718 *выражение-5-уровня* **есть** [*выражение-5-уровня* ♦ ♦ *оп-эквивалентности*] *операнд-эквивалентности*

R719 *оп-отрицания* **есть** .NOT.

R720 *оп-конъюнкции* **есть** .AND.

R721 *оп-дизъюнкции* **есть** .OR.

R722 *оп-эквивалентности* **есть** .EQV.
или .NEQV.

R723 *выражение* **есть** [*выражение* ♦ ♦ *знак-определяемой-бинарной-операции*] *выражение-5-уровня*

R724 *знак-определяемой-бинарной-операции* **есть** .*буква*[*буква*]... .

Ограничение: *знак-определяемой-бинарной-операции* должен содержать не более 31 буквы и не должен совпадать с каким-либо из *знаков-встроенной-операции* или с одной из *логических-литеральных-констант*.

R725 *логическое-выражение* **есть** *выражение*

Ограничение: *логическое-выражение* должно быть логического типа.

R726 *символьное-выражение* **есть** *выражение*

Ограничение: *символьное-выражение* должно быть символьного типа.

R727 *выражение-станд-симв-типа* **есть** *выражение*

Ограничение: *выражение-станд-симв-типа* должно быть выражением стандартного символьного типа.

R728 *целое-выражение* **есть** *выражение*

Ограничение: *целое-выражение* должно быть целого типа.

R729 *числовое-выражение* **есть** *выражение*

Ограничение: *числовое-выражение* должно быть целого, вещественного или комплексного типа.

R730 *выражение-инициализации* **есть** *выражение*

Ограничение: *выражение-инициализации* должно быть выражением инициализации.

R731 *символьное-выражение-инициализации* **есть** *символьное-выражение*

Ограничение: *символьное-выражение-инициализации* должно быть выражением инициализации.

R732 *целое-выражение-инициализации* **есть** *целое-выражение*

Ограничение: *целое-выражение-инициализации* должно быть выражением инициализации.

R733 *логическое-выражение-инициализации* **есть** *логическое-выражение*

Ограничение: *логическое-выражение-инициализации* должно быть выражением инициализации.

R734 *выражение-спецификации* **есть** *скалярное-целое-выражение*

Ограничение: *скалярное-целое-выражение* должно быть ограниченным выражением.

R735 *оператор-присваивания* **есть** *переменная* = *выражение*

Ограничение: *переменная* в *операторе-присваивания* не должна быть массивом неопределенного размера.

R736 *оператор-присваивания-указателю* **есть** *объект-указатель* => *адресат*

R737 *адресат* **есть** *переменная*
или *выражение*

Ограничение: *объект-указатель* должен иметь атрибут POINTER.

Ограничение: *переменная* должна иметь атрибут TARGET или быть подобъектом объекта, имеющего атрибут TARGET, либо она должна иметь атрибут POINTER.

Ограничение: *адресат* должен иметь тот же тип, те же параметры типа и размерность, что указатель.

Ограничение: *адресат* не должен быть секцией массива с векторным индексом.

Ограничение: *выражение* должно вырабатывать ссылку на результат.

R738 *оператор-WHERE* **есть** WHERE (*выражение-маска*) ♦
♦ *оператор-присваивания*

R739 *конструкция-WHERE* **есть** *оператор-конструкции-WHERE*
[*оператор-присваивания*]...
[*оператор-ELSEWHERE*
[*оператор-присваивания*]...]
оператор-ENDWHERE

R740 *оператор-конструкции-WHERE* **есть** WHERE (*выражение-маска*)

R741 *выражение-маска* **есть** *логическое-выражение*

R742 *оператор-ELSEWHERE* **есть** ELSEWHERE

R743 *оператор-END-WHERE* **есть** END WHERE

Ограничение: *выражение-маска* и определяемая переменная в каждом *операторе-присваивания* должны быть массивами одной конфигурации.

Ограничение: *оператор-присваивания* не должен быть определяемым присваиванием.

D.1.8. Управление выполнением

R801 *блок* **есть** [*конструкция-раздела-выполняемых-операторов*]...

R802 *конструкция-IF* **есть** *оператор-IF-THEN*
блок
[*оператор-ELSE-IF*
блок]...
[*оператор-ELSE*
блок]
оператор-END-IF

R803 *оператор-IF-THEN* **есть** [*имя-конструкции-IF* :] ♦
♦ IF (*скалярное-логическое-выражение*) THEN

R804 *оператор-ELSE-IF* **есть** ELSE IF ♦
♦ (*скалярное-логическое-выражение*) THEN [*имя-конструкции-IF*]

R805 *оператор-ELSE* **есть** ELSE [*имя-конструкции-IF*]

R806 *оператор-END-IF* **есть** END IF[*имя-конструкции-IF*]

Ограничение: если *оператор-IF-THEN* конструкции IF помечен *именем-конструкции-IF*, соответствующий *оператор-END-IF* должен специфицировать это же *имя-конструкции-IF*. Если же *оператор-IF-THEN* конструкции IF не имеет *имени-конструкции-IF*, в соответствующем *операторе-END-IF* не должно содержаться *имя-конструкции-IF*. Если *оператор-ELSE-IF* или *оператор-ELSE* содержат *имя-конструкции-IF*, соответствующий *оператор-IF-THEN* должен содержать это же *имя-конструкции-IF*.

R807 *оператор-IF* **есть** IF (*скалярное-логическое-выражение*) *оператор-действия*

Ограничение: *оператор-действия* не должен быть оператором IF, END PROGRAM, END FUNCTION или END SUBROUTINE.

R808 *конструкция-CASE* **есть** *оператор-SELECT-CASE*
[*оператор-CASE*

блок]...
оператор-END-SELECT

R809 *оператор-SELECT-CASE* **есть** [*имя-конструкции-CASE* :] ♦
♦ SELECT CASE (*выражение-выбора*)

R810 *оператор-CASE* **есть** CASE *селектор-CASE* [*имя-конструкции-CASE*]

R811 *оператор-END-SELECT* **есть** END SELECT [*имя-конструкции-CASE*]

Ограничение: если *оператор-SELECT-CASE конструкции-CASE* помечен *именем-конструкции-CASE*, в соответствующем *операторе-END-SELECT* должно быть задано это же *имя-конструкции-CASE*. Если же *оператор-SELECT-CASE конструкции-CASE* не содержит *имени-конструкции-CASE*, соответствующий *оператор-END-SELECT* не должен содержать *имени-конструкции-CASE*. Если *оператор-CASE* помечен *именем-конструкции-CASE*, соответствующий *оператор-SELECT-CASE* должен содержать это же *имя-конструкции-CASE*.

R812 *выражение-выбора* **есть** *скалярное-целое-выражение*
или *скалярное-символьное-выражение*
или *скалярное-логическое-выражение*

R813 *селектор-CASE* **есть** (*список-диапазонов-значений-CASE*)
или DEFAULT

Ограничение: не более чем один *селектор-CASE* формы DEFAULT может находиться в одной *конструкции-CASE*.

R814 *диапазон-значений-CASE* **есть** *значение-CASE*
или *значение-CASE* :
или : *значение-CASE*
или *значение-CASE* : *значение-CASE*

R815 *значение-CASE* **есть** *скалярное-целое-выражение-инициализации*
или *скалярное-символьное-выражение-инициализации*
или *скалярное-логическое-выражение-инициализации*

Ограничение: для одной конструкции CASE каждое *значение-CASE* должно быть того же типа, что *выражение-выбора*. Для символьного типа допускаются различия в длине, но параметры вида должны быть одинаковыми.

Ограничение: если *выражение-выбора* логического типа, нельзя использовать *диапазон-значений-CASE* с двоеточием.

Ограничение: для одной *конструкции-CASE* недопустимо перекрытие *диапазонов-значений-CASE*, т.е. не должно быть такого возможного значения *выражения-выбора*, которое соответствовало бы более чем одному *диапазону-значений-CASE*.

R816 *конструкция-DO* **есть** *блочная-конструкция-DO*
или *неблочная-конструкция-DO*

R817 *блочная-конструкция-DO* **есть** *оператор-DO*
блок-DO
конец-цикла

R818 *оператор-DO* **есть** *оператор-DO-с-меткой*
или *оператор-DO-без-метки*

R819 *оператор-DO-с-меткой* **есть** [*имя-конструкции-DO* :] DO *метка* ♦
♦ [*управление-циклом*]

R820 *оператор-DO-без-метки* **есть** [*имя-конструкции-DO*:] DO ♦
♦ [*управление-циклом*]

R821 *управление-циклом* **есть** [,] *DO-переменная* = ♦
♦ *скалярное-числовое-выражение*, ♦
♦ *скалярное-числовое-выражение* ♦
♦ [, *скалярное-числовое-выражение*]
или [,] WHILE(*скалярное-логическое-выражение*)

R822 *DO-переменная* **есть** *скалярная-переменная*

Ограничение: *DO-переменная* должна быть скалярной переменной целого, стандартного вещественного или вещественного двойной точности типа.

Ограничение: каждое *скалярное-числовое-выражение* в *управлении-циклом* должно иметь тип целый, стандартный вещественный или вещественный двойной точности.

R823 *блок-DO* **есть** *блок*

R824 *конец-цикла* **есть** *оператор-END-DO*
или *оператор-CONTINUE*

R825 *оператор-END-DO* **есть** END DO [*имя-конструкции-DO*]

Ограничение: если в *операторе-DO* есть *имя-конструкции-DO*, *концом-цикла* такой конструкции должен быть *оператор-END-DO* с тем же *именем-конструкции-DO*. Если же в *операторе-DO* нет *имени-конструкции-DO*, его не должно быть и в соответствующем *конце-цикла*.

Ограничение: если *оператор-DO* - это *оператор-DO-без-метки*, соответствующим *концом-цикла* должен быть *оператор-END-DO*.

Ограничение: если *оператор-DO* - это *оператор-DO-с-меткой*, *конец-цикла* должен быть помечен этой меткой.

R826 *неблочная-конструкция-DO* **есть** *конструкция-DO-оканчивающаяся-оператором-действия*
или *внешняя-конструкция-DO-с-общим-окончанием*

R827 *конструкция-DO-оканчивающаяся-оператором-действия* **есть** *оператор-DO-с-меткой*
тело-цикла
оператор-действия-оканчивающий-цикл

R828 *тело-цикла* **есть** [*конструкция-раздела-выполняемых-операторов*]...

R829 *оператор-действия-оканчивающий-цикл* **есть** *оператор-действия*

Ограничение: *оператор-действия-оканчивающий-цикл* не должен быть оператором CONTINUE, GOTO, RETURN, STOP, EXIT, CYCLE, END FUNCTION, END SUBROUTINE, END PROGRAM, арифметический IF или GOTO по предписанию.

Ограничение: *оператор-действия-оканчивающий-цикл* должен быть помечен меткой, и на нее должен ссылаться соответствующий *оператор-DO-с-меткой*.

R830 *внешняя-конструкция-DO-с-общим-окончанием* **есть** *оператор-DO-с-меткой*
тело-цикла
конструкция-DO-с-общим-окончанием

R831 *конструкция-DO-с-общим-окончанием* **есть** *внешняя-конструкция-DO-с-общим-окончанием*
или *внутренняя-конструкция-DO-с-общим-окончанием*

R832 *внутренняя-конструкция-DO-с-общим-окончанием* **есть** *оператор-DO-с-меткой*
тело-цикла
общий-оператор-окончания-цикла

R833 *общий-оператор-окончания-цикла* **есть** *оператор-действия*

Ограничение: *общий-оператор-окончания-цикла* не должен быть оператором GO TO, RETURN, STOP, EXIT, CYCLE, END FUNCTION, END SUBROUTINE, END PROGRAM, арифметический IF или GO TO по предписанию.

Ограничение: *общий-оператор-окончания-цикла* должен быть помечен меткой, и все *операторы-DO-с-меткой конструкции-DO-с-общим-окончанием* должны ссылаться на эту метку.

R834 *оператор-CYCLE* **есть** CYCLE [*имя-конструкции-DO*]

Ограничение: если *оператор-CYCLE* ссылается на *имя-конструкции-DO*, он должен находиться внутри этой *конструкции-DO*, иначе он должен находиться внутри хотя бы одной какой-нибудь *конструкции-DO*.

R835 *оператор-EXIT* **есть** EXIT [*имя-конструкции-DO*]

Ограничение: если *оператор-EXIT* ссылается на *имя-конструкции-DO*, он должен находиться внутри этой конструкции, иначе он должен входить в тело хотя бы одной из конструкций DO.

R836 *оператор-GOTO* **есть** GO TO *метка*

Ограничение: *метка* должна быть меткой оператора-адресата, находящегося в той же контекстной области, что и *оператор-GOTO*.

R837 *вычисляемый-оператор-GOTO* **есть** GO TO (*список-меток*) ♦
♦ [,] *скалярное-целое-выражение*

Ограничение: каждая метка в *списке-меток* должна быть меткой оператора-адресата, находящегося в той же контекстной области, что и *вычисляемый-оператор-GOTO*.

R838 *оператор-ASSIGN* **есть** ASSIGN *метка* TO *скалярная-целая-переменная*

Ограничение: *метка* должна быть меткой оператора-адресата для оператора перехода или оператора FORMAT, который находится в той же контекстной области, что оператор ASSIGN.

Ограничение: *скалярная-целая-переменная* должна быть именованной переменной стандартного целого типа.

R839 *оператор-GOTO-по-предписанию* **есть** GO TO *скалярная-целая-переменная* [[,] (*список-меток*)]

Ограничение: каждая *метка* из *списка-меток* должна быть меткой оператора-адресата в той же контекстной области, что и оператор ASSIGN.

Ограничение: *скалярная-целая-переменная* должна быть именованной переменной стандартного целого типа.

R840 *арифметический-оператор-IF* **есть** IF (*скалярное-числовое-выражение*) *метка,метка,метка*

Ограничение: каждая *метка* должна быть меткой оператора-адресата, который находится в той же контекстной области, что и *арифметический-оператор-IF*.

Ограничение: *скалярное-числовое-выражение* не может быть комплексным.

R841 *оператор-CONTINUE* **есть** CONTINUE

R842 *оператор-STOP* **есть** STOP [*стоп-код*]

R843 *стоп-код* **есть** *скалярная-символьная-константа*
или *цифра* [*цифра* [*цифра* [*цифра*[*цифра*]]]]

Ограничение: *скалярная-целая-константа* должна иметь стандартный целый тип.

R844 *оператор-PAUSE* **есть** PAUSE [*стоп-код*]

D.1.9. Операторы ввода/вывода

R901 *устройство-вв* **есть** *устройство-внешнего-файла*
или *
или *устройство-внутреннего-файла*

R902 *устройство-внешнего-файла* **есть** *скалярное-целое-выражение*

R903 *устройство-внутреннего-файла* **есть** *переменная-станд-символьного-типа*

Ограничение: *переменная-станд-символьного-типа* не должна быть секцией массива с векторным индексом.

R904 *оператор-OPEN* **есть** OPEN (*список-специф-подсоединения*)

R905 *специф-подсоединения* ♦
♦ **есть** [UNIT=] *устройство-внешнего-файла*
или IOSTAT= *скалярная-целая-переменная*
или ERR= *метка*
или FILE= *выражение-имени-файла*
или STATUS= *скалярное-выражение-станд-символьного-типа*
или ACCESS= *скалярное-выражение-станд-символьного-типа*
или FORM= *скалярное-выражение-станд-символьного-типа*
или RECL= *скалярное-целое-выражение*
или BLANK= *скалярное-выражение-станд-символьного-типа*
или POSITION= *скалярное-выражение-станд-символьного-типа*
или ACTION= *скалярное-выражение-станд-символьного-типа*
или DELIM= *скалярное-выражение-станд-символьного-типа*
или PAD= *скалярное-выражение-станд-символьного-типа*

R906 *выражение-имени-файла* **есть** *скалярное-символьное-выражение*

Ограничение: если необязательные символы UNIT= в спецификаторе устройства опущены, он должен быть первым элементом в *списке-специф-подсоединения*.

Ограничение: каждый из спецификаторов не должен появляться в одном *операторе-OPEN* более одного раза; *устройство-внешнего-файла* должно быть задано обязательно.

Ограничение: метка, используемая в спецификаторе ERR=, должна быть меткой адресата оператора ветвления, находящегося в той же контекстной области, что и оператор OPEN.

R907 *оператор-CLOSE* **есть** CLOSE (*список-специф-закрытия*)

R908 *специф-закрытия* **есть** [UNIT=] *устройство-внешнего-файла*
или IOSTAT= *скалярная-целая-переменная*
или ERR = *метка*
или STATUS=*скалярное-выражение-станд-символьного-типа*

Ограничение: если в спецификаторе устройства опущены необязательные символы UNIT=, спецификатор устройстства должен быть первым элементом в *списке-специф-закрытия*.

Ограничение: ни один из спецификаторов не должен входить в один *оператор-CLOSE* более одного раза; спецификатор устройства должен присутствовать всегда.

Ограничение: *метка*, используемая в спецификаторе ERR= , должна быть меткой адресата оператора ветвления, находящегося в той же контекстной области.

R909 *оператор-READ* **есть** READ (*список-специф-управления-вв*) ♦
♦ [*список-элементов-ввода*]
или READ *формат*[,*список-элементов-ввода*]

R910 *оператор-WRITE* **есть** WRITE (*список-специф-управления-вв*) ♦
♦ [*список-элементов-вывода*]

R911 *оператор-PRINT* **есть** PRINT *формат*[,*список-элементов-вывода*]

R912 *специф-управления-вв* ♦
♦ **есть** [UNIT=] *устройство-вв*
или [FMT=] *формат*
или [NML=] *имя-группы-NAMELIST*
или REC= *скалярное-целое-выражение*
или IOSTAT= *скалярная-целая-переменная*
или ERR= *метка*
или END= *метка*
или ADVANCE=*скалярное-выражение-станд-символьного-типа*
или SIZE= *скалярное-целое-выражение*
или EOR= *метка*

Ограничение: *список-специф-управления-вв* должен содержать ровно одно *устройство-вв* и может содержать не более чем по одному все остальные спецификаторы.

Ограничение: спецификаторы END= , EOR= и SIZE= не должны входить в *оператор-WRITE*.

Ограничение: *метка*, используемая в спецификаторах ERR= , EOR= и END= , должна быть меткой оператора-адресата для операторов ветвления в той же контекстной области, где находится оператор передачи данных.

Ограничение: если в операторе передачи данных присутствует *список-элементов-ввода* или *список-элементов-вывода*, в нем не должно быть *имени-группы-NAMELIST*.

Ограничение: *список-специф-управления-вв* не должен содержать одновременно и *формат*, и *имя-группы-NAMELIST*.

Ограничение: если в спецификаторе устройства опущены необязательные символы UNIT= , спецификатор устройства должен быть первым элементом в списке управляющей информации.

Ограничение: если в спецификаторе формата опущены необязательные символы FMT= , спецификатор формата должен быть вторым элементом в списке управляющей информации, а первым элементом в нем должен быть спецификатор устройства без необязательных символов UNIT= .

Ограничение: если в спецификаторе NAMELIST опущены необязательные символы NML= , этот спецификатор должен быть вторым элементом в списке управляющей информации, а первым должен быть спецификатор устройства без необязательных символов UNIT= .

Ограничение: если спецификатор устройства специфицирует внутренний файл, в *списке-специф-управления-вв* не должно быть спецификатора REC= или *имени-группы-NAMELIST*.

Ограничение: если есть спецификатор REC=, спецификатор END= и *имя-группы-NAMELIST* не должны появляться, а формат (если он есть) не должен быть звездочкой, задающей ввод/вывод, управляемый списком.

Ограничение: спецификатор ADVANCE= может присутствовать только в форматных последовательных операторах ввода/вывода с явной спецификацией формата (10.1), список управляющей информации которой не содержит спецификатора внутреннего файла.

Ограничение: если присутствует спецификатор EOR= , должен присутствовать и спецификатор ADVANCE= .

R913 *формат* **есть** *выражение-станд-символьного-типа*
или *метка*
или *
или *скалярная-целая-переменная*

Ограничение: *метка* должна быть меткой оператора FORMAT, расположенного в той же контекстной области, что оператор, содержащий спецификацию формата.

R914 *элемент-ввода* **есть** *переменная*
или *неявный-цикл-вв*

R915 *элемент-вывода* **есть** *выражение*
или *неявный-цикл-вв*

R916 *неявный-цикл-вв* **есть** (*список-объектов-нцкл-вв, управление-нцкл-вв*)

R917 *объект-нцкл-вв* **есть** *элемент-ввода*
или *элемент-вывода*

R918 *управление-нцкл-вв* **есть** *переменная-цикла=*
♦ *скалярное-числовое-выражение,*
♦ *скалярное-числовое-выражение*
♦ [,*скалярное-числовое-выражение*]

Ограничение: переменная, которая является *элементом-ввода*, не должна быть массивом неопределенного размера.

Ограничение: *переменная-цикла* должна быть скаляром целого, стандартного вещественного или вещественного двойной точности типа.

Ограничение: все *скалярные-числовые-выражения* в *управлении-нцкл-вв* должны быть целого, стандартного вещественного или вещественного двойной точности типа.

Ограничение: в *списке-элементов-ввода объект-нцкл-вв* должен быть *элементом-ввода*. В *списке-элементов-вывода объект-нцкл-вв* должен быть *элементом-вывода*.

R919 *оператор-BACKSPACE* **есть** BACKSPACE *устройство-внешнего-файла*
или BACKSPACE (*список-специф-позиции*)

R920 *оператор-ENDFILE* **есть** ENDFILE *устройство-внешнего-файла*
или ENDFILE (*список-специф-позиции*)

R921 *оператор-REWIND* **есть** REWIND *устройство-внешнего-файла*
или REWIND (*список-специф-позиции*)

R922 *специф-позиции* **есть** [UNIT=] *устройство-внешнего-файла*
или IOSTAT= *скалярная-целая-переменная*
или ERR= *метка*

Ограничение: *метка* в спецификаторе ERR= должна быть меткой адресата оператора ветвления, находящегося в той же контекстной области, что оператор установки позиции файла.

Ограничение: если в спецификаторе устройства опущены необязательные символы UNIT=, спецификатор устройства должен быть первым элементом *списка-специф-позиции*.

Ограничение: *список-специф-позиции* должен содержать ровно одно *устройство-внешнего-файла* и может содержать не более чем по одному остальные спецификаторы.

R923 *оператор-INQUIRE* **есть** INQUIRE (*список-специф-запроса*)
или INQUIRE [IOLENGTH= ♦
♦ *скалярная-переменная-станд-целого-типа*)*список-элементов-вывода*

R924 *специф-запроса* ♦
♦ **есть** [UNIT=] *устройство-внешнего-файла*
или FILE = *выражение-имени-файла*
или IOSTAT = *скалярная-переменная-станд-целого-типа*
или ERR = *метка*
или EXIST = *скалярная-переменная-станд-логического-типа*
или OPENED = *скалярная-переменная-станд-логического-типа*
или NUMBER = *скалярная-переменная-станд-целого-типа*
или NAMED = *скалярная-переменная-станд-логического-типа*
или NAME = *скалярная-переменная-станд-символьного-типа*
или ACCESS = *скалярная-переменная-станд-символьного-типа*
или SEQUENTIAL= *скалярная-переменная-станд-символьного-типа*
или DIRECT = *скалярная-переменная-станд-символьного-типа*
или FORM = *скалярная-переменная-станд-символьного-типа*
или FORMATTED= *скалярная-переменная-станд-символьного-типа*
или UNFORMATTED=*скалярная-переменная-станд-символьного-типа*
или RECL = *скалярная-переменная-станд-целого-типа*
или NEXTREC = *скалярная-переменная-станд-целого-типа*
или BLANK = *скалярная-переменная-станд-символьного-типа*
или POSITION = *скалярная-переменная-станд-символьного-типа*
или ACTION = *скалярная-переменная-станд-символьного-типа*
или READ = *скалярная-переменная-станд-символьного-типа*
или WRITE = *скалярная-переменная-станд-символьного-типа*
или READWRITE= *скалярная-переменная-станд-символьного-типа*
или DELIM = *скалярная-переменная-станд-символьного-типа*
или PAD = *скалярная-переменная-станд-символьного-типа*

Ограничение: *список-специф-запроса* должен содержать либо один спецификатор FILE=, либо один спецификатор UNIT= (но не оба одновременно) и не более чем по одному остальные спецификаторы.

Ограничение: если в операторе INQUIRE, заданном в форме запроса об устройстве, опущены необязательные символы UNIT= в спецификаторе устройства, он должен быть первым элементом в *списке-специф-запроса*.

D.1.10. Редактирование ввода/вывода

R1001 *оператор-FORMAT* **есть** FORMAT *спецификация-формата*
R1002 *спецификация-формата* **есть** ([*список-элементов-формата*])

Ограничение: *оператор-FORMAT* должен быть помечен.

Ограничение: запятая, используемая для разделения элементов формата в списке, может быть опущена в следующих случаях:
- между описателем редактирования P и следующими за ним описателями редактирования F, E, EN, ES, D или G (10.6.5);
- до или после описателя редактирования "/", когда отсутствует необязательная спецификация повторения (10.6.2);
- до или после описателя редактирования ":" (10.6.3).

R1003 *элемент-формата* **есть** [*r*] *описатель-редактирования-данных*
или *управляющий-описатель-редактирования*
или *описатель-редактирования-симв-строка*
или [*r*] (*список-элементов-формата*)

R1004 *r* **есть** *целая-литеральная-константа*

Ограничение: *r* должно быть положительным.

Ограничение: *r* должно быть задано без параметра вида.

R1005 *описатель-редактирования-данных* **есть** I*w*[.*m*]
или B*w*[.*m*]
или O*w*[.*m*]
или Z*w*[.*m*]
или F*w*.*d*
или E*w*.*d*[E*e*]
или E*w*.*d*[E*e*]
или E*w*.*d*[E*e*]
или G*w*.*d*[E*e*]
или L*w*
или A[*w*]
или D*w*.*d*

R1006 *w* **есть** *целая-литеральная-константа*

R1007 *m* **есть** *целая-литеральная-константа*

R1008 *d* **есть** *целая-литеральная-константа*

R1009 *e* **есть** *целая-литеральная-константа*

Ограничение: *w* и *e* должны быть положительными.

Ограничение: *w*, *m*, *d* и *e* должны быть специфицированы без параметров вида.

R1010 *управляющий-описатель-редактирования* ♦
♦ **есть** *описатель-редактирования-позиции*
или [*r*] /
или :
или *описатель-редактирования-знака*
или *k*P
или *описатель-редактирования-интерпр-пробелов*

R1011 *k* **есть** *целая-литеральная-константа-со-знаком*

Ограничение: *k* должно быть специфицировано без параметра вида.

R1012 *описатель-редактирования-позиции* **есть** T*n*
или TL*n*
или TR*n*
или *n*X

R1013 *n* **есть** *целая-литеральная-константа*

Ограничение: *n* должно быть положительным.

Ограничение: *n* должно быть специфицировано без параметра вида.

R1014 *описатель-редактирования-знака* **есть** S
или SP
или SS

R1015 *описатель-редактирования-интерпр-пробелов* **есть** BN
или BZ

R1016 *описатель-редактирования-симв-строка* ♦
♦ **есть** *символьная-литеральная-константа*
или *cH представимый-символ* [*представимый-символ*]...

R1017 *c* **есть** *целая-литеральная-константа*

Ограничение: *c* должно быть положительным.

Ограничение: *c* должно быть специфицировано без параметра вида.

Ограничение: *представимый-символ* в форме *cH* должен относиться к стандартному символьному типу.

Ограничение: *символьная-литеральная-константа* должна быть без параметра вида.

D.1.11. Программные единицы

R1101 *главная-программная-единица* **есть** [*оператор-PROGRAM*]
[*раздел-спецификаций*]
[*раздел-выполняемых-конструкций*]
[*раздел-внутренних-процедур*]
оператор-конца-программы

R1102 *оператор-PROGRAM* **есть** PROGRAM *имя-программы*

R1103 *оператор-конца-программы* **есть** END[PROGRAM[*имя-программы*]]

Ограничение: раздел-выполняемых-конструкций главной-программной-единицы не должен включать операторы RETURN и ENTRY.

Ограничение: имя-программы может быть включено в оператор-конца-программы, только если используется необязательный оператор-PROGRAM, и оно должно быть идентично имени-программы, специфицированному в этом операторе-PROGRAM.

Ограничение: в разделе-спецификаций (R204) главной программной единицы не должно быть автоматически размещаемых объектов.

R1104 *модуль* **есть** *оператор-MODULE*
[*раздел-спецификаций*]
[*раздел-модульных-процедур*]
оператор-конца-модуля

R1105 *оператор-MODULE* **есть** MODULE *имя-модуля*

R1106 *оператор-конца-модуля* **есть** END [MODULE [*имя-модуля*]]

Ограничение: если в операторе-конца-модуля задано имя-модуля, оно должно совпадать с именем в операторе-MODULE.

Ограничение: раздел-спецификаций модуля не должен содержать описаний-операторных-функций и операторов ENTRY и FORMAT.

Ограничение: в разделе-спецификаций (R204) модуля не должны находиться автоматически размещаемые объекты.

R1107 *оператор-USE* **есть** USE *имя-модуля* [,*список-переименований*]
или USE *имя-модуля*, ONLY:[*список-объектов-ONLY*]

R1108 *переименование* **есть** *локальное-имя* => *используемое-имя*

R1109 *объект-ONLY* **есть** *идентификатор-доступа*
или [*локальное-имя* =>] *используемое-имя*

Ограничение: каждый *идентификатор-доступа* должен быть общедоступным объектом в модуле.

Ограничение: каждое *используемое-имя* должно быть общедоступным объектом в модуле.

R1110 *блок-данных* **есть** *оператор-BLOCK-DATA*
[*раздел-спецификаций*]
оператор-конца-блока-данных

R1111 *оператор-BLOCK-DATA* **есть** BLOCK DATA [*имя-блока-данных*]

R1112 *оператор-конца-блока-данных* **есть** END [BLOCK DATA [*имя-блока-данных*]]

Ограничение: *имя-блока-данных* может быть включено в *оператор-конца-блока-данных* только в том случае, если оно есть в *операторе-BLOCK-DATA*, и тогда эти имена должны быть идентичны.

Ограничение: *раздел-спецификаций блока-данных* может содержать только операторы USE, операторы объявления типа, IMPLICIT, PARAMETER, описания производного типа и следующие операторы спецификации: COMMON, DATA, DIMENSION, EQUIVALENCE, INTRINSIC, POINTER, SAVE и TARGET.

Ограничение: операторы объявления типа в *разделе-спецификаций блока-данных* не должны содержать спецификаторов атрибутов ALLOCATABLE, EXTERNAL, INTENT, OPTIONAL, PRIVATE или PUBLIC.

D.1.12. Процедуры

R1201 *блок-интерфейса* **есть** *оператор-INTERFACE*
[*тело-интерфейса*]...
[*оператор-модульных-процедур*]...
оператор-конца-интерфейса

R1202 *оператор-INTERFACE* **есть** INTERFACE[*обобщенная-спецификация*]

R1203 *оператор-конца-интерфейса* **есть** END INTERFACE

R1204 *тело-интерфейса* **есть** *оператор-FUNCTION*
[*раздел-спецификаций*]
оператор-конца-функции
или *оператор-SUBROUTINE*
[*раздел-спецификаций*]
оператор-конца-подпрограммы

R1205 *оператор-модульных-процедур* **есть** MODULE PROCEDURE *список-имен-процедур*

R1206 *обобщенная-спецификация* **есть** *обобщенное-имя*
или OPERATOR (*знак-определяемой-операции*)
или ASSIGNMENT (=)

Ограничение: *тело-интерфейса* не должно содержать операторов ENTRY, FORMAT, DATA и описаний операторной функции.

Ограничение: спецификация MODULE PROCEDURE разрешена только если *блок-интерфейса* имеет *обобщенную-спецификацию* и объемлющий контекст его является модулем или имеет доступ к модулю через связь по использованию. Каждое *имя-процедуры* должно быть именем модульной процедуры, доступной в объемлющей области.

Ограничение: *блок-интерфейса* не может находиться в программной единице - блоке данных.

Ограничение: *блок-интерфейса* не может включать *тело-интерфейса* для процедуры, описываемой данной программной единицей.

R1207 *оператор-EXTERNAL* **есть** EXTERNAL *список-внешних-имен*

R1208 *оператор-INTRINSIC* **есть** INTRINSIC *список-имен-встроенных-процедур*

Ограничение: каждое *имя-встроенной-процедуры* должно быть именем встроенной процедуры.

R1209 *ссылка-на-функцию* **есть** *имя-функции* ♦
♦ ([*список-специф-фактических-аргументов*])

Ограничение: список-специф-фактических-аргументов в ссылке на функцию не должен включать специф-альтерн-возврата.

R1210 *оператор-CALL* **есть** CALL *имя-подпрограммы* ♦
♦ ([[*список-специф-фактических-аргументов*]])

R1211 *специф-фактического-аргумента* **есть** [*ключевое-слово* =] *фактический-аргумент*

R1212 *ключевое-слово* **есть** *имя-формального-аргумента*

R1213 *фактический-аргумент* **есть** *выражение*
или *переменная*
или *имя-процедуры*
или *специф-альтерн-возврата*

R1214 *специф-альтерн-возврата* **есть** * *метка*

Ограничение: если интерфейс процедуры в данной контекстной области неявный, конструкция "*ключевое-слово* =" не разрешена.

Ограничение: "*ключевое-слово* =" может быть опущено из *специф-фактического-аргумента* только тогда, когда опущено "*ключевое-слово* =" из каждой предшествующей *специф-фактического-аргумента* в списке аргументов.

Ограничение: каждое *ключевое-слово* должно быть именем формального аргумента в явном интерфейсе процедуры.

Ограничение: *имя-процедуры* в качестве *фактического-аргумента* не может быть именем внутренней процедуры или операторной функции и не может быть обобщенным именем процедуры (12.3.2.1, 13.1).

Ограничение: метка в *специф-альтерн-возврата* должна быть меткой оператора-адресата для оператора ветвления, находящегося в одной контекстной области с оператором CALL.

R1215 *процедура-функция* **есть** *оператор-FUNCTION*
[*раздел-спецификаций*]
[*раздел-выполняемых-конструкций*]
[*раздел-внутренних-процедур*]
оператор-конца-функции

R1216 *оператор-FUNCTION* **есть** [*префикс*] FUNCTION *имя-функции* ♦
♦ ([*список-имен-формальных-аргументов*]) ♦
♦ [RESULT (*имя-результата*)]

Ограничение: если задано ключевое слово RESULT, *имя-функции* не должно входить ни в один оператор спецификации в контекстной области функции.

R1217 *префикс* **есть** *специф-типа* [RECURSIVE]
или RECURSIVE [*специф-типа*]

R1218 *оператор-конца-функции* **есть** END [FUNCTION[*имя-функции*]]

Ограничение: если задано ключевое слово RESULT, *имя-результата* не должно совпадать с *именем-функции*.

Ограничение: ключевое слово FUNCTION обязательно для *оператора-конца-функции* в модульной или внутренней функции.

Ограничение: внутренняя функция не должна содержать операторов ENTRY.

Ограничение: во внутренней функции не должно быть *раздела-внутренних-процедур*.

Ограничение: если в *операторе-конца-функции* есть *имя-функции*, оно должно соответствовать *имени-функции* в *операторе-FUNCTION*.

R1219 *процедура-подпрограмма* **есть** *оператор-SUBROUTINE*
[*раздел-спецификаций*]
[*раздел-выполняемых-конструкций*]
[*раздел-внутренних-процедур*]
оператор-конца-подпрограммы

R1220 *оператор-SUBROUTINE* **есть** [RECURSIVE] SUBROUTINE ♦
♦ *имя-подпрограммы* [([*список-формальных-аргументов*])]

R1221 *формальный-аргумент* **есть** *имя-формального-аргумента*
или *

R1222 *оператор-конца-подпрограммы* **есть** END [SUBROUTINE [*имя-подпрограммы*]]

Ограничение: в *операторе-конца-подпрограммы* внутренней или модульной подпрограммы слово SUBROUTINE должно присутствовать.

Ограничение: внутренняя подпрограмма не должна содержать операторов ENTRY.

Ограничение: во внутренней подпрограмме не должно быть *раздела-внутренних-процедур*.

Ограничение: если в *операторе-конца-подпрограммы* есть *имя-подпрограммы*, оно должно совпадать с *именем-подпрограммы* в *операторе-SUBROUTINE*.

R1223 *оператор-ENTRY* **есть** ENTRY *имя-входа* ♦
♦ [([*список-формальных-аргументов*]) ♦
♦ [RESULT (имя-результата)]]

Ограничение: если задано ключевое слово RESULT, *имя-входа* не должно входить ни в один оператор спецификации в контекстной области процедуры-функции.

Ограничение: *оператор-ENTRY* может находиться только во *внешней-процедуре* или в *модуле*. *Оператор-ENTRY* не должен находиться внутри *выполняемой-конструкции*.

Ограничение: RESULT может присутствовать только в операторе ENTRY, расположенном в процедуре-функции.

Ограничение: в процедуре, содержащей оператор ENTRY, *имя-входа* не должно находиться в качестве формального аргумента в операторах FUNCTION или SUBROUTINE и в других операторах ENTRY и не может входить в оператор EXTERNAL или INTRINSIC.

Ограничение: *формальный-аргумент* может быть указателем альтернативного возврата только в случае, если оператор ENTRY находится в подпрограмме.

Ограничение: если задано RESULT, *имя-результата* не должно совпадать с *именем-входа*

R1224 *оператор-RETURN* **есть** RETURN [*целое-скалярное-выражение*]

Ограничение: *оператор-RETURN* должен находиться в контекстной области процедуры-функции или процедуры-подпрограммы.

Ограничение: *целое-скалярное-выражение* разрешено только в контекстной области подпрограммы.

R1225 *оператор-CONTAINS* **есть** CONTAINS

R1226 *описание-операторной-функции* **есть** *имя-функции* ♦
♦ ([*список-имен-формальных-аргументов*]) = *скалярное-выражение*

Ограничение: *скалярное-выражение* может быть образовано только из констант (литеральных и именованных), ссылок на скалярные переменные и элементы массива, ссылок на функции и формальные процедуры-функции и из встроенных операций. Если в *скалярном-выражении* появилась ссылка на другую операторную функцию, она должна быть ранее описана в контекстной области и при этом не должна быть именем описываемой операторной функции.

Ограничение: именованные константы в *скалярном-выражении* должны быть предварительно объявлены в этой контекстной области, либо должны быть доступны через связь по объемлющему контексту или по использованию. Если в *скалярном-выражении* есть элементы массива, массив-основа должен быть предварительно объявлен как массив в этой контекстной области, либо должен быть доступен через связь по объемлющему контексту или по использованию. Если скалярная переменная, элемент массива, ссылка на функцию или формальную функцию типизируются по правилам неявной типизации, все их последующие вхождения в операторы объявления типа должны согласовываться с неявно заданными типом и значениями его параметров.

Ограничение: *имя-функции* и все *имена-формальных-аргументов* должны быть специфицированы явно или неявно как скалярные объекты данных.

Ограничение: одно и то же *имя-формального-аргумента* может появиться в *списке-имен-формальных-аргументов* только один раз.

Ограничение: каждая ссылка на скалярную переменную в *скалярном-выражении* должна быть ссылкой либо на формальный аргумент операторной

функции, либо на локальную переменную из той же контекстной области, где находится описание операторной функции.

D.1.13. Встроенные процедуры

D.1.14. Область действия, связь и определенность

D.2. Перекрестные ссылки

В приведенных ниже перекрестных ссылках все вхождения скалярного-*xyz* и списка-*xyz* трактуются как вхождения *xyz*.

D.2.1. Нетерминальные символы, которые определены

Символ	Определен в	Ссылка в
адресат	R737	R736
арифметический-оператор-IF	R840	R216
блок	R801	R802, R808, R823
блок-данных	R1110	R202
блок-интерфейса	R1201	R207
блок-DO	R823	R817
блочная-конструкция-DO	R817	R816
буква-экспоненты	R415	R413
буквенно-цифровой-символ	R302	R301, R304
векторный-индекс	R621	R618
верхняя-граница	R515	R513
верхняя-граница-размещения	R628	R626
вещественная-литеральная-константа	R413	R306, R412
вещественная-литеральная-константа-со-знаком	R412	R418, R419, R533
внешняя-конструкция-DO-с-общим-окончанием	R830	R826, R831
внешняя-процедура	R203	R202
внутренняя-конструкция-DO-с-общим-окончанием	R832	R831
внутренняя-процедура	R211	R210
восьмеричная-константа	R409	R407
выполняемая-конструкция	R215	R208, R209
выполняемая-программа	R201	
выражение	R723	R430, R432, R701, R723, R725, R726, R727, R728, R729, R730, R735, R735, R737, R915, R1213, R1226
выражение-1-уровня	R703	R705
выражение-2-уровня	R707	R707, R711
выражение-3-уровня	R711	R711, R713
выражение-4-уровня	R713	R715
выражение-5-уровня	R718	R718, R723

Символ	Определен в	Ссылка в
выражение-выбора	R812	R809
выражение-имени-файла	R906	R905, R924
выражение-инициализации	R730	R504 R539
выражение-маска	R741	R738, R740
выражение-спецификации	R734	R509, R514, R515
выражение-станд-симв-типа	R727	R905, R906, R908, R912, R913
вычисляемый-оператор-GO-TO	R837	R216
главная-программная-единица	R1101	R202
двоичная-константа	R408	R407
действит-часть	R418	R417
диапазон-значений-CASE	R814	R813
диапазон-подстроки	R611	R609, R616
длина-символьного	R508	R429, R504, R507
знак	R406	R401, R403, R412
знак-встроенной-операции	R310	R312
знак-определяемой-операции	R311	R1206
знак-определяемой-бинарной-операции	R724	R311, R723
знак-определяемой-унарной-операции	R704	R311, R703
знак-расширенной-встроенной-операции	R312	R311
значащая-часть	R414	R413
значение-констр-массива	R432	R431, R433
значение-оператора-DATA	R532	R530
значение-параметра-типа	R509	R506, R507, R508
значение-CASE	R815	R814
идентификатор-доступности	R522	R521, R1109
именованная-константа	R307	R305, R539
имя	R304	R307
индекс	R617	R618, R619
индекс-секции	R618	R613
индексный-триплет	R619	R618
ключевое-слово	R1212	R1211
комплексная-литеральная-константа	R417	R306
компонент-структуры	R614	R536, R602, R610, R625, R630
конец-цикла	R824	R817
константа	R305	R308, R309, R533, R610, R701
константа-оператора-DATA	R533	R532
константа-подобъект	R702	R701
конструктор-массива	R431	R701
конструктор-структуры	R430	R533, R701
конструкция-объявления	R207	R204
конструкция-раздела-выполняемых-операторов	R209	R208, R801, R828

Символ	Определен в	Ссылка в
конструкция-CASE	R808	R215
конструкция-DO	R816	R215
конструкция-DO-оканчивающаяся-оператором-действия	R827	R826
конструкция-DO--с-общим-окончанием	R831	R830
конструкция-IF	R802	R215
конструкция-WHERE	R739	R215
литеральная-константа	R306	R305
логическая-литеральная-константа	R421	R306
логическая-переменная	R603	
логический-множитель	R715	R716
логическое-выражение	R725	R733, R741, R803, R804 R807, R812, R821
логическое-выражение-инициализации	R733	R815
логическое-слагаемое	R716	R716, R717
метка	R313	R819, R836, R837, R838, R839, R840, R905, R908, R912, R913, R922, R924, R1214
мнимая-часть	R419	R417
множество-оператора-DATA	R530	R529
множество-эквивалентности	R546	R545
множитель	R705	R705, R706
модуль	R1104	R202
модульная-процедура	R213	R212
неблочная-конструкция-DO	R826	R816
неявный-цикл-вв	R916	R914, R915
неявный-цикл-констр-массива	R433	R432
неявный-цикл-DATA	R535	R531, R536
нижняя-граница	R514	R513, R516, R518
нижняя-граница-размещения	R627	R626
обобщенная-спецификация	R1206	R522, R1202
объект-группы-NAMELIST	R544	R543
объект-нцкл-DATA	R536	R535
объект-нцкл-вв	R917	R916
объект-оператора-DATA	R531	R530
объект-общего-блока	R549	R548
объект-указатель	R630	R629, R736
объявление-объекта	R504	R501
общий-оператор-окончания-цикла	R833	R832
оп-дизъюнкции	R721	R310, R717
оп-конкатенации	R712	R310, R711
оп-конъюнкции	R720	R310, R716
оп-отношения	R714	R310, R713
оп-отрицания	R719	R310, R715

Символ	Определен в	Ссылка в
оп-сложения	R710	R310, R707
оп-степени	R708	R310, R705
оп-умножения	R709	R310, R706
оп-эквивалентности	R722	R310, R718
операнд-эквивалентности	R717	R717, R718
оператор-действия	R216	R215, R807, R829,R833
оператор-действия-оканчивающий-цикл	R829	R827
оператор-доступности	R521	R214
оператор-заголовка-типа	R424	R422
оператор-конца-блока-данных	R1112	R1110
оператор-конца-интерфейса	R1203	R1201
оператор-конца-модуля	R1106	R1104
оператор-конца-подпрограммы	R1222	R216, R1204,R1219
оператор-конца-программы	R1103	R216, R1101
оператор-конца-типа	R425	R422
оператор-конца-функции	R1218	R216, R1204,R1215
оператор-конструкции-WHERE	R740	R739
оператор-модульных-процедур	R1205	R1201
оператор-объявления-типа	R501	R207
оператор-описания-компонента	R426	R422
оператор-присваивания	R735	R216, R738, R739
оператор-присваивания-указателю	R736	R216
оператор-раздела-IMPLICIT	R206	R205
оператор-спецификации	R214	R207
оператор-ALLOCATABLE	R526	R214
оператор-ALLOCATE	R622	R216
оператор-ASSIGN	R838	R216
оператор-BLOCK-DATA	R1111	R1110
оператор-BACKSPACE	R919	R216
оператор-CALL	R1210	R216
оператор-CASE	R810	R808
оператор-CLOSE	R907	R216
оператор-COMMON	R548	R214
оператор-CONTAINS	R1225	R210, R212
оператор-CONTINUE	R841	R216, R824
оператор-CYCLE	R834	R216
оператор-DATA	R529	R209, R214
оператор-DEALLOCATE	R631	R216
оператор-DIMENSION	R525	R214
оператор-DO	R818	R817
оператор-DO-с-меткой	R819	R818, R827, R830,R832
оператор-DO-без-метки	R820	R818
оператор-ELSE	R805	R802
оператор-ELSE-IF	R804	R802

Символ	Определен в	Ссылка в
оператор-ELSEWHERE	R742	R739
оператор-ENDFILE	R920	R216
оператор-END-DO	R825	R824
оператор-END-IF	R806	R802
оператор-END-SELECT	R811	R808
оператор-END-WHERE	R743	R739
оператор-ENTRY	R1223	R206, R207, R209
оператор-EQUIVALENCE	R545	R214
оператор-EXIT	R835	R216
оператор-EXTERNAL	R1207	R214
оператор-FORMAT	R1001	R206, R207, R209
оператор-FUNCTION	R1216	R1204, R1215
оператор-GOTO	R836	R216
оператор-GOTO-по-предписанию	R839	R216
оператор-IF-THEN	R803	R802
оператор-IF	R807	R216
оператор-IMPLICIT	R540	R205, R206
оператор-INQUIRE	R923	R216
оператор-INTENT	R519	R214
оператор-INTERFACE	R1202	R1201
оператор-INTRINSIC	R1208	R214
оператор-MODULE	R1105	R1104
оператор-NAMELIST	R543	R214
оператор-NULLIFY	R629	R216
оператор-OPEN	R904	R216
оператор-OPTIONAL	R520	R214
оператор-PARAMETER	R538	R206, R207
оператор-PAUSE	R844	R216
оператор-POINTER	R527	R214
оператор-PRINT	R911	R216
оператор-PRIVATE-SEQUENCE	R423	R422
оператор-PROGRAM	R1102	R1101
оператор-READ	R909	R216
оператор-RETURN	R1224	R216
оператор-REWIND	R921	R216
оператор-SAVE	R523	R214
оператор-SELECT-CASE	R809	R808
оператор-STOP	R842	R216
оператор-SUBROUTINE	R1220	R1204, R1219
оператор-TARGET	R528	R214
оператор-USE	R1107	R204
оператор-WHERE	R738	R216
оператор-WRITE	R910	R216
описатель-редактирования-данных	R1005	R1003

Символ	Определен в	Ссылка в
описатель-редактирования-знака	R1014	R1010
описатель-редактирования-интерпр-пробелов	R1015	R1010
описатель-редактирования позиции	R1012	R1010
описатель-редактирования-симв-строка	R1016	R1003
описание-именованной-константы	R539	R538
описание-компонента	R429	R426
описание-операторной-функции	R1226	R207
описание-производного-типа	R422	R207
параметр-вида	R405	R404, R413, R420,R421
первичное	R701	R703
переименование	R1108	R1107
переменная	R601	R531, R603, R604,R605, R606, R607, R608,R701, R735, R737, R822,R914, R1213
переменная-нцкл-констр-массива	R435	R434
переменная-нцкл-DATA	R537	R535
переменная-состояния	R623	R622, R631
переменная-станд-логич-типа	R604	R924
переменная-станд-симв-типа	R606	R903, R924
переменная-станд-целого-типа	R608	R905, R908, R912,R913, R922, R923, R924
повторитель-оператора-DATA	R534	R532
подобъект	R602	R601, R702
подпрограмма	R1219	R203, R211, R213
подстрока	R609	R547, R602
подчеркивание	R303	R302
префикс	R1217	R1216
программная-единица	R202	R201
процедура-функция	R1215	R203, R211, R213
раздел-внутренних-процедур	R210	R1101, R1215, R1219
раздел-выполняемых-операторов	R208	R1101, R1215, R1219
раздел-модульных-процедур	R212	R1104
раздел-спецификаций	R204	R1101, R1104, R1110, R1204, R1215, R1219
раздел-IMPLICIT	R205	R204
размещаемый-объект	R625	R624, R631
размещение	R624	R622
секция-массива	R616	R602
селектор-вида	R505	R502
селектор-длины	R507	R506
селектор-символьного	R506	R502
селектор-CASE	R813	R810
символ	R301	

Символ	Определен в	Ссылка в
символьная-константа	R309	R843
символьная-литеральная-константа	R420	R306, R1016
символьная-переменная	R605	
символьное-выражение	R726	R731, R812
символьное-выражение-инициализации	R731	R815
слагаемое	R706	R706, R707
сохраняемый-объект	R524	R523
спецификация-размещения	R626	R624
спецификация-формата	R1002	R1001
специф-альтерн-возврата	R1214	R1213
специф-атрибута	R503	R501
специф-атрибутов-компонента	R427	R426
специф-буквы	R542	R541
специф-доступности	R510	R424, R503, R521
специф-закрытия	R908	R907
специф-запроса	R924	R923
специф-компонента-массива	R428	R427, R429
специф-массива	R512	R503, R504, R525, R528
специф-назначения	R511	R503, R519
специф-неопределенного-размера	R518	R512
специф-неопределенной-конфигурации	R516	R512
специф-определяемой-конфигурации	R517	R428, R512, R526, R527
специф-подсоединения	R905	R904
специф-позиции	R922	R919, R920, R921
специф-типа	R502	R426, R501, R541, R1217
специф-управления-вв	R912	R909, R910
специф-фактического-аргумента	R1211	R1209, R1210
специф-явной-конфигурации	R513	R428, R512, R518, R549
специф-IMPLICIT	R541	R540
ссылка-на-данное	R612	R614, R615, R616
ссылка-на-функцию	R1209	R701
ссылка-на-часть	R613	R612
стоп-код	R843	R842, R844
строка-основа	R610	R609
строка-цифр	R402	R401, R404, R405, R413, R414
строка-цифр-со-знаком	R401	R416
тело-интерфейса	R1204	R1201
тело-цикла	R828	R827, R830, R832
управление-нцкл-вв	R918	R916
управление-нцкл-констр-массива	R434	R433
управление-циклом	R821	R819, R820

Символ	Определен в	Ссылка в
управляющий-описатель-редактирования	R1010	R1003
устройство-вв	R901	R912
устройство-внешнего-файла	R902	R901, R905, R908, R919, R920
устройство-внутреннего-файла	R903	R901
фактический-аргумент	R1213	R1211
формальный-аргумент	R1221	R1220, R1223
формат	R913	R909, R911, R912
целая-константа	R308	R534
целая-литеральная-константа-со-знаком	R403	R418, R419, R533, R1011
целая-литеральная-константа	R404	R306, R403, R508, R1004, R1006, R1007, R1108, R1009, R1013, R1017
целая-переменная	R607	R435, R537, R623, R838, R839
целое-выражение	R728	R434, R535, R611, R617, R620, R621, R627, R628, R732, R734, R812, R837, R902, R905, R912, R1224
целое-выражение-инициализации	R732	R505, R506, R815
числовое-выражение	R729	R821, R840, R918
шаг	R620	R619
шестнадцатеричная-константа	R410	R407
шестнадцатеричная-цифра	R411	R410
эквивалентный-объект	R547	R546
экспонента	R416	R413
элемент-ввода	R914	R909, R917
элемент-вывода	R915	R910, R911, R917, R923
элемент-массива	R615	R536, R547, R602, R610
элемент-формата	R1003	R1002, R1003
BOZ-литеральная-константа	R407	R306, R533
c	R1017	R1016
d	R1008	R1005
DO-переменная	R822	R821, R918
e	R1009	R1005
k	R1011	R1010
m	R1007	R1005
n	R1013	R1012
r	R1004	R1003, R1010
w	R1006	R1005

D.2.2. Нетерминальные символы, которые не определены

Символ	Ссылка в
буква	R302, R304, R542, R704, R724
внешнее-имя	R1207
имя-блока-данных	R1111, R1112
имя-встроенной-процедуры	R1208
имя-входа	R1223
имя-группы-NAMELIST	R543, R912
имя-для-USE	R522, R1108, R1109
имя-компонента	R429
имя-конструкции-CASE	R809, R810, R811
имя-конструкции-DO	R819, R820, R825, R834, R835,
имя-конструкции-IF	R803, R804, R805, R806,
имя-массива	R525, R526
имя-модуля	R1105, R1106, R1107
имя-общего-блока	R524, R548
имя-объекта	R504, R524, R527, R528
имя-переменной	R544, R547, R549, R601, R610, R625, R630
имя-переменной-массива	R601
имя-подпрограммы	R1210, R1220, R1222
имя-программы	R1102, R1103
имя-процедуры	R1205, R1213
имя-результата	R1216, R1223
имя-типа	R424, R425, R430, R502
имя-формального-аргумента	R519, R520, R1212, R1216, R1221, R1226
имя-функции	R504, R1209, R1216, R1218, R1226
имя-целой-константы	R405
имя-части	R613
локальное-имя	R1108, R1109
обобщенное-имя	R1206
представимый-символ	R420, R1106
специальный-символ	R301
цифра	R302, R313, R402, R408, R409, R411, R843

D.2.3. Терминальные символы

Символ	Ссылка в
%	R612
.	R408, R409, R410, R420
(	R417, R427, R429, R430, R433, R502, R503, R504, R505, R506, R507, R508, R519, R525, R526, R527, R528, R535, R538, R541, R546, R549, R609, R613, R616, R622, R624, R629, R631, R701, R738, R740, R803, R804, R807, R809, R813, R821, R837, R839, R840, R904, R907, R909, R910, R916, R919, R920, R921, R923, R1002, R1003, R1206, R1209, R1210, R1216, R1220, R1223, R1226,
(/	R431

Символ	Ссылка в
)	R417, R427, R429, R430, R433, R502, R503, R504, R505, R506, R507, R508, R519, R525, R526, R527, R528, R535, R538, R541, R546, R549, R609, R613, R616, R622, R624, R629, R631, R701, R738, R740, R803, R804, R807, R809, R813, R821, R837, R839, R840, R904, R907, R909, R910, R916, R919, R920, R921, R923, R1002, R1003, R1206, R1209, R1210, R1216, R1220, R1223, R1226
*	R429, R504, R507, R509, R518, R532, R709, R901, R913 R1214, R1221
**	R708
+	R406, R710
,	R417, R424, R426, R433, R434, R501, R506, R507, R518, R525, R526, R527, R528, R529, R535, R543, R546, R548, R622, R631, R821, R837, R839, R840, R909, R911, R916, R918, R1107,
-	R406, R542, R710
.	R414, R704, R724, R1005
.AND.	R720
.EQ.	R714
.EQV.	R722
.FALSE.	R421
.GE.	R714
.GT.	R714
.LE.	R714
.LT.	R714
.NE.	R714
.NEQV.	R722
.NOT.	R719
.OR.	R721
.TRUE.	R421
/	R524, R530, R543, R548, R709, R1010
/)	R431
//	R712
/=	R714
:	R513, R516, R517, R518, R611, R619, R626, R803, R809, R814, R819, R820, R1010, R1107
::	R424, R426, R501, R519, R520, R521, R523, R525, R526, R527, R528
<	R714
<=	R714
=	R434, R504, R505, R506, R507, R535, R539, R622, R631, R735, R821, R905, R908, R912, R918, R922, R923, R924, R1206, R1211, R1226
==	R714
=>	R736, R1108, R1109
>	R714
>=	R714
A	R411, R1005
ACCESS	R905, R924
ACTION	R905, R924

Символ	Ссылка в
ADVANCE	R912
ALLOCATABLE	R503, R526
ALLOCATE	R622
ASSIGN	R838
ASSIGNMENT	R1206
B	R408, R411, R1005
BACKSPACE	R919
BLANK	R905, R924
BLOCK	R1111, R1112
BN	R1015
BZ	R1015
C	R411
CALL	R1210
CASE	R809, R810
CHARACTER	R502
CLOSE	R907
COMMON	R548
COMPLEX	R502
CONTAINS	R1225
CONTINUE	R841
CYCLE	R834
D	R411, R415, R1005
DATA	R529, R1111, R1112
DEALLOCATE	R631
DEFAULT	R813
DELIM	R905, R924
DIMENSION	R427, R503, R525
DIRECT	R924
DO	R819, R820, R825
DOUBLE	R502
E	R411, R415, R1005
ELSE	R804, R805
ELSEWHERE	R742
EN	R1005
END	R425, R743, R806, R811, R825, R912, R1103, R1106, R1112 R1203, R1218, R1222
ENDFILE	R920
ENTRY	R1223
EOR	R912
EQUIVALENCE	R545
ERR	R905, R908, R912, R922, R924
ES	R1005
EXIST	R924
EXIT	R835
EXTERNAL	R503, R1207
F	R411, R1005

Символ	Ссылка в
FILE	R905, R924
FMT	R912
FORM	R905, R924
FORMAT	R1001
FORMATTED	R924
FUNCTION	R1216, R1218
G	R1005
GO	R836, R837, R839
H	R1016
I	R1005
IF	R803, R804, R806, R807, R840
IMPLICIT	R540
IN	R511
INOUT	R511
INQUIRE	R923
INTEGER	R502
INTENT	R503, R519
INTERFACE	R1202, R1203
INTRINSIC	R503, R1208
IOLENGTH	R923
IOSTAT	R905, R908, R912, R922, R924
KIND	R505, R506
L	R1005
LEN	R506, R507
LOGICAL	R502
MODULE	R1105, R1106, R1205
NAME	R924
NAMED	R924
NAMELIST	R543
NEXTREC	R924
NML	R912
NONE	R540
NULLIFY	R629
NUMBER	R924
O	R409, R1005
ONLY	R1107
OPEN	R904
OPENED	R924
OPERATOR	R1206
OPTIONAL	R503, R520
OUT	R511
P	R1010
PAD	R905, R924
PARAMETER	R503, R538
PAUSE	R844

POINTER R427, R503, R527
POSITION R905, R924
PRECISION R502
PRINT R911
PRIVATE R423, R510
PROCEDURE R1205
PROGRAM R1102, R1103
PUBLIC R510
READ R909, R924
READWRITE R924
REAL R502
REC R912
RECL R905, R924
RECURSIVE R1217, R1220
RESULT R1216, R1223
RETURN R1224
REWIND R921
S R1014
SAVE R503, R523
SELECT R809, R811
SEQUENCE R423
SEQUENTIAL R924
SIZE R912
SP R1014
SS R1014
STAT R622, R631
STATUS R905, R908
STOP R842
SUBROUTINE R1220, R1222
T R1012
TARGET R503, R528
THEN R803, R804
TL R1012
TO R836, R837, R838, R839
TR R1012
TYPE R424, R425, R502
UNFORMATTED R924
UNIT R905, R908, R912, R922, R924
USE R1107
WHERE R738, R740, R743
WHILE R821
WRITE R910, R924
X R1012
Z R410, R1005
- R303, R404, R413, R420, R421

Приложение Е
(информационное)

Переупорядоченный индекс для заголовков

Это приложение содержит оглавление документа, переупорядоченное в соответствии с латинским алфавитом. Авторы перевода не сочли целесообразным приводить это приложение, т.к. в переводе есть оглавление и индекс терминов, упорядоченных в соответствии с русским алфавитом.

ПРИЛОЖЕНИЕ F
(информационное)

ИНДЕКС

автоматически размещаемый массив	50	
автоматически размещаемый объект	44	
адресат	100	R737
активная	110	
арифметический-оператор-IF	116	R840
атрибут доступности	48	
атрибут ALLOCATABLE	53	
атрибут DIMENSION	49	
атрибут EXTERNAL	53	
атрибут INTENT	48	
атрибут INTRINSIC	53	
атрибут OPTIONAL	52	
атрибут PARAMETER	48	
атрибут POINTER	52	
атрибут SAVE	52	
атрибут TARGET	53	
атрибуты	43	
блок	103	
блок	103	R801
блок-данных	7, 173	R1110
блок-интерфейса	179	R1201
блок интерфейса процедуры	10	
блок-DO	109	R823
блочная-конструкция-DO	108	R817
буква-экспоненты	31	R415
буквенно-цифровой-символ	19	R302
буквы	19	
векторный индекс	72	
векторный-индекс	70	R621
верхняя-граница	49	R515
верхняя-граница-размещения	73	R628
вещественная-литеральная-константа	31	R413
вещественная-литеральная-константа-со-знаком	31	R412
вещественный тип	30	
вещественный двойной точности	30	
вид	29, 30, 31, 32, 34	
внешнее устройство	122	
внешний файл	118	
внешняя-конструкция-DO-с-общим-окончанием	109	R830
внешняя процедура	9, 10, 175	
внешняя-процедура	7	R203
внутренняя-конструкция-DO-с-общим-окончанием	109	R832
внутренняя процедура	9, 10, 175	
внутренняя-процедура	8	R211
внутреннее устройство	122	
восьмеричная-константа	30	R409
встроенная бинарная операция	80	
встроенная операция	80	
встроенная операция отношения	81	
встроенная процедура	175	
встроенная функция	197	
встроенная унарная операция	80	
встроенный	17	

встроенный оператор логического присваивания	97	
встроенный оператор присваивания	97	
встроенный оператор присваивания для производного типа	97	
встроенный оператор присваивания массиву	97	
встроенный оператор символьного присваивания	97	
встроенный оператор числового присваивания	97	
встроенный тип	13	
выполнение цикла	111	
выполняемая-конструкция	8	R215
выполняемая программа	9	
выполняемая-программа	7	R201
выполняемый оператор	11	
выражение	15, 77	
выражение	80	R723
выражение-1-уровня	78	R703
выражение-2-уровня	78	R707
выражение-3-уровня	79	R711
выражение-4-уровня	79	R713
выражение-5-уровня	79	R718
выражение-выбора	106	R812
выражение-имени-файла	124	R906
выражение инициализации	84	
выражение-инициализации	85	R730
выражение-маска	101	R741
выражение спецификации	86	
выражение-спецификации	86	R734
выражение-станд-симв-типа	82	R727
вычисляемый-оператор-GO-TO	115	R837
главная программная единица	167	
главная-программная-единица	167	
глобальный объект	255	
графические символы	19	
данные	14	
двоичная-константа	30	R408
действительнаячасть	32	
действит-часть	32	R418
диапазон-значений-CASE	106	R814
диапазон-подстроки	68	R611
динамически размещаемый массив	50	
длина	32	
длина-символьного	46	R508
запись	117	
запись "конец файла"	117	
запрос об устройстве	139	
запрос о списке вывода	139	
запрос о файле	139	
знак	29	R406
знак-встроенной-операции	21	R310
знак встроенной операции отношения	81	
знак логической встроенной операции	81	
знак операции	17	
знак операции расширения	82	
знак-определяемой-бинарной-операции	22, 80	R724
знак-определяемой-операции	22	R311
знак-определяемой-унарной-операции	22, 78	R704
знак-расширенной-встроенной-операции	22	R312
знак символьной встроенной операции	81	
знак числовой встроенной операции	81	

значащая-часть	31	R414
значение-констр-массива	40	R432
значение-оператора-DATA	56	R532
значение-параметра-типа	46	R509
значение-CASE	106	R815
идентификатор-доступности	54	R522
имена	21	
именованная константа	14	
именованная-константа	21	R307
именованный общий блок	63	
именованный файл	118	
имеют общее окончание	109	
имя	16	
имя	21	R304
имя программы	167	
индекс	70	R617
индекс-секции	70	R618
индекс-CASE	106	
индексный-триплет	70	R619
интерфейс	178	
исключенные средства	5	
ключевое слово	16	
ключевое-слово	184	R1212
ключевое слово аргумента	16	
ключевые слова операторов	16	
комментарий	23	
комплексная-литеральная-константа	31	R417
комплексный тип	31	
компонент-структуры	68	
компонент-структуры	69	R614
компоненты	27	
конечная точка	68	
конечная точка файла	119	
конечные компоненты	34	
конец-цикла	109	R824
константное выражение	84	
константа	14	
константа	21	R305
константа-оператора-DATA	57	R533
константа-подобъект	77	R702
конструктор массива	40	
конструктор-массива	40	R431
конструктор структуры	39	
конструктор-структуры	40	R430
конструкция-объявления	7	R207
конструкция-раздела-выполняемых-операторов	8	R209
конструкция CASE	105	
конструкция-CASE	105	R808
конструкция-DO	108	R816
конструкция-DO-оканчивающаяся-оператором-действия	109	R827
конструкция-DO-с-общим-окончанием	109	R831
конструкция IF	104	
конструкция-IF	104	R802
конструкция-WHERE	101	R739
контекстная область	9	
конфигурация	15	
левая граница табуляции	156	
лексические элементы	20	

литеральная константа 14
литеральная-константа 21 R306
литеральные 33
логическая встроенная операция 81
логическая-литеральная-константа 34 R421
логическая-переменная 67 R603
логический-множитель 79 R715
логический тип 34
логическое-выражение 82 R725
логическое-выражение-инициализации 85 R733
логическое константное выражение 84
логическое-слагаемое 79 R716
массив 15,69
массив неопределенного размера 51
массив неопределенной конфигурации 50
массив определяемой конфигурации 50
массив явной конфигурации 49
масштабный множитель 147
метка 22 R313
метка оператора 22, 115
мнимая часть 31
мнимая-часть 32 R419
многозначная секция 72
множество-оператора-DATA 56 R530
множество разрешенных действий 118
множество разрешенных длин записей 118
множество разрешенных способов доступа 118
множество разрешенных форм 118
множество-эквивалентности 61 R546
множитель 78 R705
модуль 10, 168
модуль 7, 168 R1104
модульная процедура 9, 10, 175
модульная-процедура 8 R213
набор символов Фортрана 19
начальной точкой 68
начальная точка файла 119
неактивная 110
неблочная-конструкция-DO 109 R826
невыполняемый оператор 11
неименованный общий блок 63
не определена 17
неопределенный 261
неопределенный-параметр-типа 46
не связанный 16, 261
нечисловые типы 32
неформатная запись 117
неформатный оператор ввода/вывода 129
неявный интерфейс 178
неявный-цикл-вв 132 R916
неявный-цикл-констр-массива 41 R433
неявный-цикл-DATA 57 R535
нижняя-граница 49 R514
нижняя-граница-размещения 73 R627
номер записи 119
объект данных 14
объект-группы 61 R544
объект-нцкл-DATA 57 R536

объект-нцкл-вв	132	R917
объект-основа	69	
объект-оператора-DATA	56	R531
объект-общего-блока	63	R549
объект-указатель	74	R630
объект-ONLY	169	R1109
объемлющая контекстная область	9	
объемлющий контекст	10, 168	
объявление	16	
объявление-компонента	35	R429
объявление-объекта	43	R504
область	110	
область действия	255	
обобщенная-спецификация	179	R1206
обобщенное имя	181	
обобщенный идентификатор	180	
обобщенный интерфейс	180	
обозначение подобъекта	16	
общие блоки	63	
общий-оператор-окончания-цикла	109	R833
ограниченное выражение	86	
ограничители	22	
окончание DO	109	
оп-дизъюнкции	22, 80	R721
оп-конкатенации	21, 79	R712
оп-конъюнкции	22, 80	R720
оп-отношения	21, 79	R714
оп-отрицания	22, 79	R719
оп-сложения	21, 78	R710
оп-степени	21, 78	R708
оп-умножения	21, 78	R709
оп-эквивалентности	22, 80	R722
операнд-эквивалентности	79	R717
оператор	22	
оператор-адресат	115	
оператор ввода	117	
оператор ввода данных	117	
оператор ввода/вывода без продвижения	120	
оператор ввода/вывода последовательного доступа	130	
оператор ввода/вывода по NAMELIST	130	
оператор ввода/вывода прямого доступа	130	
оператор ввода/вывода с продвижением	120	
оператор ввода/вывода, управляемого списком.	130	
оператор вывода	117	
оператор вывода данных	117	
оператор-действия	8	R216
оператор-действия-оканчивающий-цикл	109	R829
оператор-доступности	54	R521
оператор-заголовка-типа	35	R424
оператор запроса о файле	117	
оператор-конструкции-WHERE	101	R740
оператор-конца-блока-данных	173	R1112
оператор-конца-интерфейса	179	R1203
оператор-конца-модуля	168	R1106
оператор-конца-программы	167	R1103
оператор-конца-подпрограммы	190	R1222
оператор-конца-типа	35	R425
оператор-конца-функции	188	R1218

оператор-модульных-процедур 179 R1205
оператор объявления типа 43
оператор-объявления-типа 43 R501
оператор-описания-компонента 35 R426
оператор перехода 115
оператор подсоединения файла 117
оператор-присваивания 97 R735
оператор-присваивания-указателю 100 R736
оператор-раздела-IMPLICIT 7 R206
оператор-спецификации 8 R214
оператор установления позиции файла 117
оператор-ALLOCATABLE 55 R526
оператор ALLOCATE 73
оператор-ALLOCATE 73 R622
оператор-ASSIGN 116 R838
оператор-BACKSPACE 138 R919
оператор-BLOCK-DATA 173 R1111
оператор-CALL 184 R1210
оператор-CASE 106 R810
оператор CLOSE 126
оператор-CLOSE 127 R907
оператор COMMON 63
оператор-COMMON 63 R548
оператор CONTAINS 192
оператор-CONTAINS 192 R1225
оператор-CONTINUE 116 R841
оператор-CYCLE 111 R834
оператор DATA 56
оператор-DATA 56 R529
оператор DEALLOCATE 74
оператор-DEALLOCATE 74 R631
оператор-DIMENSION 55 R525
оператор-DO 108 R818
оператор-DO-без-метки 108 R820
оператор-DO-с-меткой 108 R819
оператор-ELSEWHERE 101 R742
оператор-ELSE-IF 104 R804
оператор-ELSE 104 R805
оператор END 12
оператор-END-DO 109 R825
оператор-ENDFILE 138 R920
оператор-END-IF 104 R806
оператор-END-SELECT 106 R811
оператор-END-WHERE 101 R743
оператор ENTRY 190
оператор-ENTRY 190 R1223
оператор EQUIVALENCE 61
оператор-EQUIVALENCE 61 R545
оператор-EXIT 111 R835
оператор EXTERNAL 183
оператор-EXTERNAL 183 R1207
оператор-FORMAT 145 R1001
оператор-FUNCTION 188 R1216
оператор-GO-TO 115 R836
оператор-GOTO-по-предписанию 116 R839
оператор IF 104, 105
оператор-IF 105 R807
оператор-IF-THEN 104 R803

оператор IMPLICIT 59
оператор-IMPLICIT 59 R540
оператор-INQUIRE 139 R923
оператор-INTENT 53 R519
оператор-INTERFACE 179 R1202
оператор INTRINSIC 183
оператор-INTRINSIC 183 R1208
оператор-MODULE 168 R1105
оператор NAMELIST 61
оператор-NAMELIST 61 R543
оператор NULLIFY 74
оператор-NULLIFY 74 R629
оператор OPEN 123
оператор-OPEN 123 R904
оператор-OPTIONAL 54 R520
оператор PARAMETER 58
оператор-PARAMETER 58 R538
оператор-PAUSE 116 R844
оператор-POINTER 56 R527
оператор PRINT 127
оператор-PRINT 128 R911
оператор-PRIVATE-SEQUENCE 35 R423
оператор-PROGRAM 167 R1102
оператор READ 127
оператор-READ 127 R909
оператор RETURN 192
оператор-RETURN 192 R1224
оператор-REWIND 138 R921
оператор-SAVE 54 R523
оператор-SELECT-CASE 105 R809
оператор-STOP 116 R842
оператор-SUBROUTINE 190 R1220
оператор-TARGET 56 R528
оператор USE 169
оператор-USE 169 R1107
оператор-WHERE 101 R738
оператор WRITE 127
оператор-WRITE 127 R910
операторная функция 177
операторный объект 255
операция конкатенации 33
операция расширения 82
описание 17
описание-именованной-константы 58 R539
описание-операторной-функции 195 R1226
описание-производного-типа 35 R422
описатель редактирования данных 146
описатель-редактирования-данных 147 R1005
описатель-редактирования-знака 147 R1014
описатель-редактирования-интерпр-пробелов 147 R1015
описатель-редактирования-позиции 146 R1012
описатель редактирования "символьная строка" 146
описатель-редактирования-симв-строка 147 R1016
определение 17
определенный 17
определяемая бинарная операция 82
определяемая операция 81
определяемая унарная операция 81

определяемый оператор присваивания	98	
процессор	1	
процессорно-зависимый	3	
параметр вида	13	
параметр-вида	29	R405
параметр длины	13	
параметр типа	13	
первичное	77	R701
переименование	169	R1108
переменная	14	
переменная	67	R601
переменная-нцкл-констр-массива	41	R435
переменная-нцкл-DATA	57	R537
переменная-состояния	73	R623
переменная-станд-логич-типа	67	R604
переменная-станд-симв-типа	67	R606
переменная-станд-целого-типа	67	R608
печать	137	
повторитель-оператора-DATA	57	R534
подобъект	14	
подобъект	67	R602
подобъект константы	14	
подпоследовательность имя-значение	162	
подпрограмма	10	
подсоединенный	122	
подстрока	68	
подстрока	68	R609
подчеркивание	19	R303
позиция	118	
поле	147	
полный массив	69	
полный массив-именованная константа	69	
полный массив-переменная	69	
последовательная структура	48	
последовательность	18	
последовательность памяти общего блока	64	
последовательность элементов	186	
последовательный доступ	119	
последовательный тип	35	
поэлементная	175	
поэлементная подпрограмма	197	
поэлементная ссылка	187	
поэлементная функция	197	
предварительное подсоединение	123	
предшествующая запись	120	
префикс	188	R1217
принадлежит	103	
присваивание массиву по маске	101	
присутствует	192	
программная единица	9	
программная-единица	7	R202
программная единица - блок данных	173	
производный тип	13	
протяженность	15	
процедура	10	
процедура-подпрограмма	7, 190	R1219
процедура-функция	7, 188	R1215
прямой доступ	119	

раздел-спецификаций	7	R204
раздел-IMPLICIT	7	R205
раздел-выполняемых-операторов	8	R208
раздел-внутренних-процедур	8	R210
раздел-модульных-процедур	8	R212
разделитель значений	159	
размер	15	
размерность	15	
размер общего блока	64	
размер последовательности памяти	262	
размещаемый-объект	73	R625
размещение	73	R624
размещенный в данный момент	74	
результирующая переменная	10	
свободная форма исходного текста	23	
связан	16	
связаны по использованию	169	
связаны полностью	263	
связаны по памяти	262	
связаны частично	263	
связь	17	
связь по объемлющему контексту	176	
секция массива	15, 71	
секция-массива	70	R616
селектор-вида	43	R505
селектор-длины	46	R507
селектор-символьного	46	R506
селектор-CASE	106	R813
символ	19	R301
символьная встроенная операция	81	
символьная встроенная операция отношения	81	
символьная-константа	21	R309
символьная литеральная константа	32	
символьная-литеральная-константа	32	R420
символьная-переменная	67	R605
символьная последовательная структура	47	
символьная строка	32	
символьное-выражение	82	R726
символьное-выражение-инициализации	85	R731
символьное константное выражение	84	
символьный контекст	23	
символьный последовательный тип	36	
символьный тип	32	
синтаксис нижнего уровня	20	
синтаксические правила	4	
ситуация "конец записи"	133	
ситуация "конец файла"	133	
скаляр	15, 68	
следующая запись	120	
следующий действующий элемент	135	
слагаемое	78	R706
согласованы	15	
согласованы по конфигурации	83	
создать файл	118	
соответствующая стандарту программа	2	
сохраняемый объект	52	
сохраняемый-объект	54	R524
специальный символ	20	

спецификатор типа	45	
спецификация повторения	146	
спецификация-размещения	73	R626
спецификация-формата	145	R1002
специфическийинтерфейс	180	
специфические имена	197	
специф-альтерн-возврата	184	R1214
специф-атрибута	43	R503
специф-атрибутов-компонента	35	R427
специф-буквы	59	R542
специф-доступности	48	R510
специф-закрытия	127	R908
специф-запроса	139	R924
специф-компонента-массива	35	R428
специф-массива	49	R512
специф-назначения	48	R511
специф-неопределенного-размера	51	R518
специф-неопределенной-конфигурации	50	R516
специф-определяемой-конфигурации	51	R517
специф-подсоединения	123	R905
специф-позиции	138	R922
специф-типа	43	R502
специф-управления-вв	128	R912
специф-фактического-аргумента	184	R1211
специф-явной-конфигурации	49	R513
специф-IMPLICIT	59	R541
список управляющей информации	128	
способы аппроксимации	30	
способы представления	29, 30, 32, 34	
ссылка-на-данное	68	R612
ссылка на модуль	17, 168	
ссылка на объект данных	17	
ссылка на процедуру	17	
ссылка на функцию	15	
ссылка-на-функцию	184	R1209
ссылка-на-часть	68	R613
стандартный вещественный тип	30	
стандартный комплексный тип	31	
стандартный логический тип	34	
стандартный модуль	6	
стандартный символьный тип	32	
стандартный целый тип	29	
стоп-код	116	R843
строка	22	
строка-основа	68	R610
строка-цифр	29	R402
строка-цифр-со-знаком	29	R401
строка INCLUDE	25	
структура	13, 28, 47	
существуют	118	
счетчик числа итераций	110	
текущая запись	120	
тело интерфейса	180	
тело-интерфейса	179	R1204
тело-цикла	109	R828
тип данных	13, 27	
удалить файл	118	
указатель	16	

указатель-на-массив	50	
упорядоченность элементов массива	72	
упорядочивающая последовательность	33	
упорядочивающая последовательность ASCII	34	
управление-нцкл-вв	132	R918
управление-нцкл-констр-массива	41	R434
управление форматом	147	
управление-циклом	108	R821
управляющие символы	19	
управляющий описатель редактирования	146	
управляющий-описатель-редактирования	146	R1010
устаревшие средства	5	
устройство	121	
устройство-вв	121	R901
устройство-внешнего-файла	122	R902
устройство-внутреннего-файла	122	R903
файл	118	
фактический-аргумент	184	R1213
фиксированная форма исходного текста	24	
форма исходного текста	23	
формальная процедура	177	
формальный-аргумент	190	R1221
формат	129	R913
форматная запись	117	
форматный оператор ввода/вывода	129	
функция	10	
функции преобразования	197	
характеристики процедуры	178	
целая-константа	21	R308
целая-литеральная-константа	29	R404
целая-литеральная-константа-со-знаком	29	R403
целая-переменная	67	R607
целое-выражение	82	R728
целое-выражение-инициализации	85	R732
целое константное выражение	84	
целый тип	29	
цикл	108	
цифры	19	
числовая встроенная операция	81	
числовая встроенная операция отношения	81	
числовая единица памяти	262	
числовое-выражение	82	R729
числовое константное выражение	84	
числовой последовательный тип	36	
числовые типы	29	
чтение	117	
шаг	72	
шаг	70	R620
шестнадцатеричная-константа	30	R410
шестнадцатеричная-цифра	30	R411
ширина поля	147	
эквивалентный-объект	62	R547
экземпляр	190	
экспонента	31	R416
элемент-ввода	132	R914
элемент-вывода	132	R915
элемент массива	15	
элемент-массива	69	R615

элемент-формата	146	R1003
элементы массива	69	
явный	178	
BOZ-*литеральная-константа*	30	R407
DO-переменная	109	R822
c	147	R1017
d	146	R1008
e	146	R1009
k	146	R1011
m	146	R1007
n	146	R1013
PRIVATE	54	
PUBLIC	54	
r	146	R1004
RECURSIVE	189	
w	146	R1006

ОГЛАВЛЕНИЕ

Предисловие редактора перевода XXIII
Предисловие к русскому переводу XXVII
Предисловие XXVIII

ВВЕДЕНИЕ XXIX
Стандартный язык программирования Фортран XXIX

ОБЩИЕ СВЕДЕНИЯ XXIX

Операции над массивами XXIX
Числовые вычисления XXX
Параметризованный символьный тип данных XXX
Производные типы XXX
Модульные определения XXX
Указатели XXX
Эволюция языка XXXI

ОРГАНИЗАЦИЯ ДАННОГО МЕЖДУНАРОДНОГО СТАНДАРТА XXXI

Понятия верхнего/нижнего уровней XXXI
Концепция данных XXXI
Вычисления XXXII
Управление исполнением XXXII
Ввод/вывод XXXII
Программные единицы XXXIII
Область действия и правила связывания XXXIII

Глава 1. **ОБЩИЕ СВЕДЕНИЯ** 1

1.1. Цель 1
1.2. Процессор 1
1.3. Что включено в стандарт и что не включено 1
1.3.1. Что включено 1
1.3.2. Что не включено 1
1.4. Соответствие стандарту 2
1.4.1. Совместимость с ФОРТРАНом 77 3
1.5. Условные обозначения 3
1.5.1. Синтаксические правила 4
1.5.2. Подразумеваемые синтаксические правила 5
1.5.3. Синтаксические соглашения и характеристики 5
1.5.4. Текстовые обозначения 5
1.6. Исключенные и устаревшие средства 5
1.6.1. Природа исключенных средств 5
1.6.2. Природа устаревших средств 6
1.7. Модули 6
1.8. Нормативные ссылки 6

Глава 2. **ТЕРМИНЫ И ПОНЯТИЯ ФОРТРАНА** 7

2.1. Синтаксис верхнего уровня 7
2.2. Понятия программной единицы 9
2.2.1. Выполняемая программа 9
2.2.2. Главная программная единица 9
2.2.3. Процедура 10
2.2.3.1. Внешняя процедура 10
2.2.3.2. Модульная процедура 10
2.2.3.3. Внутренняя процедура 10
2.2.3.4. Блок интерфейса процедуры 10
2.2.4. Модуль 10
2.3. Концепции выполнения 11
2.3.1. Выполняемые/невыполняемые операторы 11
2.3.2. Порядок операторов 11
2.3.3. Оператор END 12
2.3.4. Последовательность выполнения 12
2.4. Концепции данных 13
2.4.1. Тип данных 13
2.4.1.1. Встроенный тип 13
2.4.1.2. Производный тип 13
2.4.2. Значения данных 14
2.4.3. Данное 14
2.4.3.1. Объект данных 14
2.4.3.2. Выражение 15
2.4.3.3. Ссылка на функцию 15
2.4.4. Скаляр 15
2.4.5. Массив 15
2.4.6. Указатель 16
2.4.7. Память 16
2.5. Основные термины 16
2.5.1. Имя и обозначение 16
2.5.2. Ключевое слово 16
2.5.3. Объявление 16
2.5.4. Определение и описание 17
2.5.5. Ссылка 17
2.5.6. Связь 17
2.5.7. Встроенный 17
2.5.8. Знак операции 17
2.5.9. Последовательность 18

Глава 3. **СИМВОЛЫ, ЛЕКСИЧЕСКИЕ ЭЛЕМЕНТЫ И ФОРМА ИСХОДНОГО ТЕКСТА** 19

3.1. Набор символов процессора 19
3.1.1. Буквы 19
3.1.2. Цифры 19
3.1.3. Символ подчеркивания 19
3.1.4. Специальные символы 20
3.1.5. Другие символы 20

Глава 3. СИМВОЛЫ, ЛЕКСИЧЕСКИЕ ЭЛЕМЕНТЫ И ФОРМА ИСХОДНОГО ТЕКСТА

3.2. Синтаксис нижнего уровня 20
3.2.1. Ключевые слова 20
3.2.2. Имена 21
3.2.3. Константы 21
3.2.4. Знаки операций 21
3.2.5. Метки операторов 22
3.2.6. Ограничители 22
3.3. Форма исходного текста 22
3.3.1. Свободная форма 23
3.3.1.1. Комментарий в свободной форме 23
3.3.1.2. Разделение операторов в свободной форме..... 24
3.3.1.3. Продолжение оператора в свободной форме ... 24
3.3.1.4. Операторы в свободной форме 24
3.3.2. Фиксированная форма 24
3.3.2.1. Комментарии в фиксированной форме 25
3.3.2.2. Разделение операторов в фиксированной форме 25
3.3.2.3. Продолжение оператора в фиксированной форме 25
3.3.2.4. Операторы в фиксированной форме 25
3.4. Включение исходного текста 25

Глава 4. **ВСТРОЕННЫЕ И ПРОИЗВОДНЫЕ ТИПЫ ДАННЫХ** 27

4.1. Концепция типа данных 27
4.1.1. Множество значений 27
4.1.2. Константы 28
4.1.3. Операции 28
4.2. Взаимосвязь типов и значений объектов 28
4.3. Встроенные типы данных 29
4.3.1. Числовые типы 29
4.3.1.1. Тип целый 29
4.3.1.2. Вещественный тип 30
4.3.1.3. Комплексный тип 31
4.3.2. Нечисловые типы 32
4.3.2.1. Символьный тип 32
4.3.2.2. Логический тип 34
4.4. Производные типы 35
4.4.1. Описание производного типа 35
4.4.2. Определение производных типов 38
4.4.3. Значения производного типа 40
4.4.4. Конструирование значений производного типа 40
4.4.5. Операции и присваивание для производного типа 41
4.5. Конструирование значений массива 41

Глава 5. **ОБЪЯВЛЕНИЯ И СПЕЦИФИКАЦИИ ОБЪЕКТОВ ДАННЫХ**. 43

5.1. Операторы объявления типа 43
5.1.1. Спецификаторы типа..... 45
5.1.1.1. INTEGER 45
5.1.1.2. REAL 45

Глава 5. ОБЪЯВЛЕНИЯ И СПЕЦИФИКАЦИИ ОБЪЕКТОВ ДАННЫХ
5.1.1.3. DOUBLE PRECISION ... 45
5.1.1.4. COMPLEX ... 46
5.1.1.5. CHARACTER ... 46
5.1.1.6. LOGICAL ... 47
5.1.1.7. Производный тип ... 47
5.1.2. Атрибуты ... 48
5.1.2.1. Атрибут PARAMETER ... 48
5.1.2.2. Атрибут доступности ... 48
5.1.2.3. Атрибут INTENT ... 48
5.1.2.4. Атрибут DIMENSION ... 49
5.1.2.5. Атрибут SAVE ... 52
5.1.2.6. Атрибут OPTIONAL ... 52
5.1.2.7. Атрибут POINTER ... 52
5.1.2.8. Атрибут TARGET ... 53
5.1.2.9. Атрибут ALLOCATABLE ... 53
5.1.2.10. Атрибут EXTERNAL ... 53
5.1.2.11. Атрибут INTRINSIC ... 53
5.2. Операторы спецификации атрибутов ... 53
5.2.1. Оператор INTENT ... 53
5.2.2. Оператор OPTIONAL ... 54
5.2.3. Операторы PRIVATE и PUBLIC ... 54
5.2.4. Оператор SAVE ... 55
5.2.5. Оператор DIMENSION ... 55
5.2.6. Оператор ALLOCATABLE ... 55
5.2.7. Оператор POINTER ... 56
5.2.8. Оператор TARGET ... 56
5.2.9. Оператор DATA ... 56
5.2.10. Оператор PARAMETER ... 58
5.3. Оператор IMPLICIT ... 59
5.4. Оператор NAMELIST ... 61
5.5. Связь объектов данных по памяти ... 61
5.5.1. Оператор EQUIVALENCE ... 61
5.5.1.1. Связь по эквивалентности ... 62
5.5.1.2. Эквивалентность объектов стандартного символьного типа ... 62
5.5.1.3. Имена массивов и обозначение элементов массива ... 63
5.5.1.4. Ограничения на оператор EQUIVALENCE ... 63
5.5.2. Оператор COMMON ... 63
5.5.2.1. Последовательность памяти общего блока ... 64
5.5.2.2. Размер общего блока ... 65
5.5.2.3. Связь по общему блоку ... 65
5.5.2.4. Разница между именованным и неименованным общими блоками ... 65
5.5.2.5. Ограничения на COMMON и EQUIVALENCE ... 66

Глава 6. **ИСПОЛЬЗОВАНИЕ ОБЪЕКТОВ ДАННЫХ** ... 67

6.1. Скаляры ... 68
6.1.1. Подстроки ... 68
6.1.2. Компоненты структур ... 68

Глава 6. ИСПОЛЬЗОВАНИЕ ОБЪЕКТОВ ДАННЫХ

6.2. Массивы ... 69
6.2.1 Полный массив ... 69
6.2.2. Элементы и секции массива ... 70
6.2.2.1. Элементы массива ... 70
6.2.2.2. Последовательность элементов массива ... 70
6.2.2.3. Секции массива ... 71
6.3. Динамическая связь ... 73
6.3.1 Оператор ALLOCATE 73
6.3.1.1. Размещение динамически размещаемых массивов ... 74
6.3.1.2. Размещение адресатов указателей ... 74
6.3.2. Оператор NULLIFY ... 74
6.3.3. Оператор DEALLOCATE ... 74
6.3.3.1. Удаление из памяти динамически размещаемых массивов ... 75
6.3.3.2. Удаление из памяти адресатов указателей ... 75

Глава 7. **ВЫРАЖЕНИЯ И ПРИСВАИВАНИЕ** ... 77

7.1 Выражения ... 77
7.1.1 Форма выражения ... 77
7.1.1.1. Первичное ... 77
7.1.1.2. Выражения уровня-1 ... 78
7.1.1.3. Выражения уровня-2 ... 78
7.1.1.4. Выражения уровня-3 ... 79
7.1.1.5. Выражения уровня-4 ... 79
7.1.1.6. Выражения уровня-5 ... 79
7.1.1.7. Общая форма выражения ... 80
7.1.2. Встроенные операции ... 80
7.1.3. Определяемые операции ... 81
7.1.4. Тип, параметры типа и конфигурация выражения ... 82
7.1.4.1. Тип, параметры типа и конфигурация первичного ... 82
7.1.4.2. Тип, параметры типа и конфигурация результата операции ... 83
7.1.5. Правила согласования для встроенных операций ... 83
7.1.6. Выражения-скаляры и выражения-массивы ... 84
7.1.6.1. Константное выражение ... 84
7.1.6.2. Выражение спецификации ... 85
7.1.7. Выполнение операций ... 87
7.1.7.1. Вычисление операндов ... 88
7.1.7.2. Целостность скобок ... 88
7.1.7.3. Выполнение числовых встроенных операций ... 88
7.1.7.4. Выполнение встроенной символьной операции 89
7.1.7.5. Выполнение встроенных операций отношения 90
7.1.7.6. Выполнение логических встроенных операций 90
7.1.7.7. Выполнение определяемых операций ... 90
7.2. Интерпретация встроенных операций ... 90
7.2.1. Числовые встроенные операции ... 90
7.2.1.1. Целочисленное деление ... 91
7.2.1.2. Комплексное возведение в степень ... 91

Глава 7. ВЫРАЖЕНИЯ И ПРИСВАИВАНИЕ

7.2.2. Символьная встроенная операция ... 91
7.2.3. Встроенные операции отношения ... 92
7.2.4. Логические встроенные операции ... 94
7.3. Интерпретация определяемых операций ... 94
7.3.1. Унарная определяемая операция ... 94
7.3.2. Бинарная определяемая операция ... 95
7.4. Старшинство операций ... 95
7.5. Присваивание ... 97
7.5.1. Оператор присваивания ... 97
7.5.1.1. Общая форма ... 97
7.5.1.2. Встроенный оператор присваивания ... 97
7.5.1.3. Определяемый оператор присваивания ... 98
7.5.1.4. Правила согласования для встроенного присваивания ... 98
7.5.1.5. Интерпретация встроенного присваивания 99
7.5.1.6. Интерпретация определяемых операторов присваивания ... 100
7.5.2. Оператор присваивания указателю ... 100
7.5.3. Присваивание массивам по маске - WHERE ... 101
7.5.3.1. Общая форма присваивания массивам по маске ... 101
7.5.3.2. Интерпретация присваивания массиву по маске 102

Глава 8. **УПРАВЛЕНИЕ ВЫПОЛНЕНИЕМ** ... 103

8.1. Выполняемые конструкции ... 103
8.1.1. Правила управления блоками ... 103
8.1.1.1. Выполняемые конструкции в блоках ... 103
8.1.1.2. Передача управления в блоках ... 103
8.1.1.3. Выполнение блока ... 104
8.1.2. Конструкция IF ... 104
8.1.2.1. Форма конструкции IF ... 104
8.1.2.2. Выполнение конструкции IF ... 104
8.1.2.3. Примеры конструкции IF ... 105
8.1.2.4. Оператор IF ... 105
8.1.3. Конструкция CASE ... 105
8.1.3.1. Форма конструкции CASE ... 105
8.1.3.2. Выполнение конструкции CASE ... 106
8.1.3.3. Примеры конструкции CASE ... 107
8.1.4. Конструкция DO ... 108
8.1.4.1. Формы конструкции DO ... 108
8.1.4.2. Область конструкции DO ... 110
8.1.4.3. Активные и неактивные конструкции DO ... 110
8.1.4.4. Выполнение конструкции DO ... 110
8.1.4.5. Примеры конструкции DO ... 112
8.2. Операторы перехода (операторы ветвления) ... 115
8.2.1. Метки операторов ... 115
8.2.2. Оператор GO TO ... 115
8.2.3. Вычисляемый оператор GO TO ... 115
8.2.4. Оператор ASSIGN и оператор GO TO по предписанию 115
8.2.5. Арифметический оператор IF ... 116

Глава 8. УПРАВЛЕНИЕ ВЫПОЛНЕНИЕМ

8.3. Оператор CONTINUE 116
8.4. Оператор STOP 116
8.5. Оператор PAUSE 116

Глава 9. **ОПЕРАТОРЫ ВВОДА/ВЫВОДА** 117

9.1. Записи 117
9.1.1. Форматная запись 117
9.1.2. Неформатная запись 117
9.1.3. Запись "конец файла" 117
9.2. Файлы 118
9.2.1. Внешние файлы 118
9.2.1.1. Существование файла 118
9.2.1.2. Доступ к файлу 118
9.2.1.3. Позиция файла 119
9.2.2. Внутренние файлы 121
9.2.2.1. Свойства внутренних файлов 121
9.2.2.2. Ограничения на внутренние файлы 121
9.3. Подсоединение файла 121
9.3.1. Существование устройства 122
9.3.2. Подсоединение файла к устройству 122
9.3.4. Предварительное подсоединение 123
9.3.4. Оператор OPEN 123
9.3.4.1. Спецификатор FILE= в операторе OPEN 124
9.3.4.2. Спецификатор STATUS= в операторе OPEN 124
9.3.4.3. Спецификатор ACCESS= в операторе OPEN 125
9.3.4.4. Спецификатор FORM= в операторе OPEN 125
9.3.4.5. Спецификатор RECL= в операторе OPEN 125
9.3.4.6. Спецификатор BLANK= в операторе OPEN 125
9.3.4.7. Спецификатор POSITION= в операторе OPEN 125
9.3.4.8. Спецификатор ACTION= в операторе OPEN ... 126
9.3.4.9. Спецификатор DELIM= в операторе OPEN 126
9.3.4.10. Спецификатор PAD= в операторе OPEN 126
9.3.5. Оператор CLOSE 126
9.3.5.1. Спецификатор STATUS= в операторе CLOSE 127
9.4. Операторы передачи данных 127
9.4.1. Список управляющей информации 128
9.4.1.1. Спецификатор формата 129
9.4.1.2. Спецификатор NAMELIST 130
9.4.1.3. Номер записи 130
9.4.1.4. Статус ввода/вывода 130
9.4.1.5. Ветвь ошибки 131
9.4.1.6. Ветвь "конец файла" 131
9.4.1.7. Ветвь "конец записи" 131
9.4.1.8. Спецификатор продвижения 132
9.4.1.9. Число символов 132
9.4.2. Список ввода/вывода для передачи данных 132
9.4.3. Ситуации ошибки, "конец записи" и "конец файла" 133
9.4.4. Выполнение операторов ввода/вывода для передачи данных 134

Глава 9. ОПЕРАТОРЫ ВВОДА/ВЫВОДА

9.4.4.1. Направление передачи данных 134
9.4.4.2. Идентификация устройства 134
9.4.4.3. Установление формата 135
9.4.4.4. Передача данных 135
9.4.4.5. Форматирование, управляемое списком 137
9.4.4.6. Форматирование по NAMELIST 137
9.4.5. Печать форматных записей 137
9.4.6. Завершение операторов передачи данных 137
9.5. Операторы установки позиции файла 138
9.5.1. Оператор BACKSPACE 138
9.5.2. Оператор ENDFILE 138
9.5.3. Оператор REWIND 139
9.6. Запрос о файле 139
9.6.1. Спецификаторы запроса 139
9.6.1.1. Спецификатор FILE= в операторе INQUIRE 140
9.6.1.2. Спецификатор EXIST= в операторе INQUIRE 140
9.6.1.3. Спецификатор OPENED= в операторе INQUIRE 140
9.6.1.4. Спецификатор NUMBER= в операторе INQUIRE 141
9.6.1.5. Спецификатор NAMED= в операторе INQUIRE 141
9.6.1.6. Спецификатор NAME= в операторе INQUIRE 141
9.6.1.7. Спецификатор ACCESS= в операторе INQUIRE 141
9.6.1.8. Спецификатор SEQUENTIAL= в операторе INQUIRE 141
9.6.1.9. Спецификатор DIRECT= в операторе INQUIRE 141
9.6.1.10. Спецификатор FORM= в операторе INQUIRE 141
9.6.1.11. Спецификатор FORMATTED= в операторе INQUIRE 141
9.6.1.12. Спецификатор UNFORMATTED= в операторе INQUIRE 142
9.6.1.13. Спецификатор RECL= в операторе INQUIRE 142
9.6.1.14. Спецификатор NEXTREC= в операторе INQUIRE 142
9.6.1.15. Спецификатор BLANK= в операторе INQUIRE 142
9.6.1.16. Спецификатор POSITION в операторе INQUIRE 142
9.6.1.17. Спецификатор ACTION= в операторе INQUIRE 142
9.6.1.18. Спецификатор READ= в операторе INQUIRE 143
9.6.1.19. Спецификатор WRITE= в операторе INQUIRE 143
9.6.1.20. Спецификатор READWRITE= в операторе INQUIRE 143
9.6.1.21. Спецификатор DELIM= в операторе INQUIRE 143
9.6.1.22. Спецификатор PAD= в операторе INQUIRE 143
9.6.2. Ограничения на спецификаторы запроса 143
9.6.3. Запрос о списке вывода 143
9.7. Ограничения для ссылок на функции и для элементов списка 144
9.8. Ограничения на операторы ввода/вывода 144

Глава 10. **РЕДАКТИРОВАНИЕ ВВОДА/ВЫВОДА** 145
10.1. Методы явной спецификации формата 145
10.1.1. Оператор FORMAT 145
10.1.2. Символьная спецификация формата 145
10.2. Форма списка элементов формата 146
10.2.1. Описатели редактирования 146
10.2.2. Поля 147
10.3. Взаимодействие между списком ввода/вывода и форматом 147
10.4. Установка позиции файла при управлении форматом 148
10.5. Описатели редактирования данных 149
10.5.1. Описатели редактирования чисел 149
10.5.1.1. Редактирование целых данных 149
10.5.1.2. Описатели редактирования вещественных и комплексных данных 150
10.5.2. Редактирование логических данных 153
10.5.3. Редактирование символьных данных 153
10.5.4. Обобщенные описатели редактирования 154
10.5.4.1. Обобщенные описатели редактирования чисел 154
10.5.4.2. Обобщенные описатели редактирования данных логического типа 155
10.5.4.3. Обобщенные описатели редактирования данных символьного типа 155
10.6. Управляющие описатели редактирования 155
10.6.1. Описатели редактирования позиции 155
10.6.1.1. Описатели редактирования T, TL и TR 156
10.6.1.2. Описатель редактирования X 156
10.6.2. Описатель редактирования "дробная черта" 156
10.6.3. Описатель редактирования "двоеточие" 157
10.6.4. Описатели редактирования S, SP и SS 157
10.6.5. Описатель редактирования P 157
10.6.5.1. Масштабный множитель 157
10.6.6. Описатели редактирования BN и BZ 158
10.7. Описатели редактирования "символьная строка" 158
10.7.1. Описатель редактирования "символьная константа" 158
10.7.2. Описатель редактирования H 158
10.8. Форматирование, управляемое списком 158
10.8.1. Ввод, управляемый списком 159
10.8.1.1. Пустые значения 160
10.8.1.2. Пример ввода, управляемого списком 160
10.8.2. Вывод, управляемый списком 161
10.9. Форматирование по NAMELIST 162
10.9.1. Ввод по NAMELIST 162
10.9.1.1. Имена объектов группы NAMELIST 162
10.9.1.2. Значения данных, вводимых по NAMELIST 163
10.9.1.3. Элементы списка объектов группы NAMELIST 163
10.9.1.4. Пустые значения 164
10.9.1.5. Пробелы 164
10.9.1.6. Пример ввода по NAMELIST 165

Глава 10. РЕДАКТИРОВАНИЕ ВВОДА/ВЫВОДА
10.9.2. Вывод по NAMELIST ... 165
10.9.2.1. Редактирование вывода по NAMELIST ... 165
10.9.2.2. Записи вывода по NAMELIST ... 166

Глава 11. **ПРОГРАММНЫЕ ЕДИНИЦЫ** ... 167
11.1. Главная программная единица ... 167
11.1.1. Спецификации главной программной единицы ... 167
11.1.2. Раздел выполняемых конструкций главной программной единицы ... 167
11.1.3. Внутренние процедуры главной программной единицы ... 168
11.2. Внешние процедуры ... 168
11.3. Модули ... 168
11.3.1. Ссылка на модуль ... 168
11.3.2. Оператор USE и связь по использованию ... 169
11.3.3. Примеры использования модулей ... 170
11.3.3.1. Идентичные общие блоки ... 170
11.3.3.2. Глобальные данные ... 170
11.3.3.3. Производные типы ... 171
11.3.3.4. Глобальные динамически размещаемые массивы ... 171
11.3.3.5. Библиотеки процедур ... 171
11.3.3.6. Расширение семантики знаков операций ... 172
11.3.3.7. Абстракция данных ... 172
11.3.3.8. Переименованные общедоступные объекты 172
11.4. Программные единицы - блоки данных ... 173

Глава 12. **ПРОЦЕДУРЫ** ... 175
12.1. Классификации процедур ... 175
12.1.1. Классификация процедур по ссылке ... 175
12.1.2. Классификация процедур по способу описания ... 175
12.1.2.1. Встроенные процедуры ... 175
12.1.2.2. Внешние, внутренние и модульные процедуры ... 175
12.1.2.3. Формальные процедуры ... 177
12.1.2.4. Операторные функции ... 177
12.2. Характеристики процедур ... 178
12.2.1. Характеристики формальных аргументов ... 178
12.2.1.1. Характеристики формальных объектов данных ... 178
12.2.1.2. Характеристики формальных процедур ... 178
12.2.1.3. Характеристики формального аргумента - звездочки ... 178
12.2.2. Характеристики результатов функции ... 178
12.3. Интерфейс процедуры ... 178
12.3.1. Явный и неявный интерфейсы ... 178
12.3.1.1. Явный интерфейс ... 179
12.3.2. Спецификация интерфейса процедуры ... 179
12.3.2.1. Блок интерфейса процедуры ... 179
12.3.2.2. Оператор EXTERNAL ... 183
12.3.2.3. Оператор INTRINSIC ... 183
12.3.2.4. Спецификация неявного интерфейса ... 183

Глава 12. ПРОЦЕДУРЫ
12.4. Ссылка на процедуру ... 184
12.4.1. Список фактических аргументов ... 184
12.4.1.1. Аргументы, связанные с формальными объектами данных ... 185
12.4.1.2. Аргументы, связанные с формальными процедурами ... 186
12.4.1.3. Аргументы, связанные с указателями альтернативного возврата ... 186
12.4.1.4. Связь по последовательности ... 186
12.4.2. Ссылка на функцию ... 187
12.4.3. Поэлементная ссылка на встроенную функцию ... 187
12.4.4. Ссылка на подпрограмму ... 187
12.4.5. Поэлементная ссылка на встроенную подпрограмму 187
12.5. Определение процедуры ... 188
12.5.1. Определение встроенной процедуры ... 188
12.5.2. Процедуры, определяемые средствами Фортрана ... 188
12.5.2.1. Действие атрибута INTENT в процедурах ... 188
12.5.2.2. Процедура-функция ... 188
12.5.2.3. Подпрограмма ... 190
12.5.2.4. Экземпляры процедуры ... 190
12.5.2.5. Оператор ENTRY ... 190
12.5.2.6. Оператор RETURN ... 192
12.5.2.7. Оператор CONTAINS ... 192
12.5.2.8. Ограничения на отсутствующие формальные аргументы ... 192
12.5.2.9. Ограничения на объекты, связанные с формальными аргументами ... 193
12.5.3. Определение процедур способом, отличным от Фортрана ... 194
12.5.4. Операторная функция ... 195

Глава 13. **ВСТРОЕННЫЕ ПРОЦЕДУРЫ** ... 197
13.1. Встроенные функции ... 197
13.2. Поэлементные встроенные процедуры ... 197
13.2.1. Аргументы и результаты поэлементной встроенной функции ... 197
13.2.2. Аргументы встроенной поэлементной подпрограммы 197
13.3. Позиционные аргументы и аргументы с ключевыми словами ... 198
13.4. Функция запроса о наличии аргументов ... 198
13.5. Числовые, математические, символьные, логические, битовые процедуры и процедуры для параметра вида ... 198
13.5.1. Числовые функции ... 198
13.5.2. Математические функции ... 198
13.5.3. Символьные функции ... 198
13.5.4. Символьная справочная функция ... 198
13.5.5. Функции для параметра вида ... 198
13.5.6. Логическая функция ... 198

Глава 13. ВСТРОЕННЫЕ ПРОЦЕДУРЫ
13.5.7. Процедуры обработки и справочные функции для битовых данных ... 199
13.6. Функция передачи типа ... 199
13.7. Функции обработки и справочные функции для числовых данных ... 199
13.7.1. Модели для целых и вещественных данных ... 199
13.7.2. Числовые справочные функции ... 200
13.7.3. Функции обработки данных с плавающей запятой ... 200
13.8. Встроенные функции для массивов ... 200
13.8.1. Конфигурация массивов-аргументов ... 200
13.8.2. Аргументы-маски ... 201
17.8.3. Функции векторного и матричного умножения ... 201
13.8.4. Функции редукции ... 201
13.8.5. Справочные функции для массивов ... 201
13.8.6. Функции конструирования массивов ... 201
13.8.7. Функция изменения конфигурации массива ... 202
13.8.8. Функции обработки массивов ... 202
13.8.9. Функции определения расположения в массиве ... 202
13.8.10. Справочная функция для запроса о статусе связанности указателя ... 202
13.9. Встроенные подпрограммы ... 202
13.9.1. Подпрограммы даты и времени ... 202
13.9.2. Псевдослучайные числа ... 202
13.9.3. Подпрограмма копирования битов ... 202
13.10. Обобщенные встроенные функции ... 203
13.10.1. Справочная функция для запроса о наличии аргументов ... 203
13.10.2. Числовые функции ... 203
13.10.3. Математические функции ... 204
13.10.4. Символьные функции ... 204
13.10.5. Символьная справочная функция ... 204
13.10.6. Справочные функции вида ... 204
13.10.7. Логическая функция ... 205
13.10.8. Числовые справочные функции ... 205
13.10.9. Битовая справочная функция ... 205
13.10.10. Функции для операций с битами ... 205
13.10.11. Функция передачи типа ... 205
13.10.12. Функции для операций с плавающей точкой ... 205
13.10.13. Функции векторного и матричного умножения ... 206
13.10.14. Функции редукции ... 206
13.10.15. Функции запроса о массиве ... 206
13.10.16. Функции конструирования массива ... 206
13.10.17. Функция изменения конфигурации массива ... 206
13.10.18. Функции манипуляции с массивами ... 206
13.10.19. Функции определения положения в массиве ... 207
13.10.20. Функция запроса о статусе связанности указателя 207
13.11. Встроенные подпрограммы ... 207
13.12. Специфические имена для встроенных функций ... 207

Глава 13. ВСТРОЕННЫЕ ПРОЦЕДУРЫ

13.13. Спецификации встроенных процедур 209
13.13.1. ABS (A) 209
13.13.2. ACHAR (I) 209
13.13.3. ACOS (X) 210
13.13.4. ADJUSTL (STRING) 210
13.13.5. ADJUSTR (STRING) 210
13.13.6. AIMAG (Z) 210
13.13.7. AINT(A, KIND) 211
13.13.8. ALL (MASK, DIM) 211
13.13.9. ALLOCATED (ARRAY) 212
13.13.10. ANINT(A, KIND) 212
13.13.11. ANY (MASK, DIM) 212
13.13.12. ASIN(X) 213
13.13.13. ASSOCIATED (POINTER, TARGET) 213
13.13.14. ATAN(X) 214
13.13.15. ATAN2(Y, X) 214
13.13.16. BIT_SIZE (I) 214
13.13.17. BTEST(I, POS) 215
13.13.18. CEILING (A) 215
13.13.19. CHAR (I, KIND) 215
13.13.20. CMPLX (X, Y, KIND) 216
13.13.21. CONJG(Z) 216
13.13.22. COS (X) 216
13.13.23. COSH(X) 216
13.13.24. COUNT (MASK, DIM) 217
13.13.25. CSHIFT (ARRAY, SHIFT, DIM) 217
13.13.26. DATE_AND_TIME (DATE, TIME, ZONE, VALUES) ... 218
13.13.27. DBLE(A) 219
13.13.28. DIGITS (X) 219
13.13.29. DIM (X, Y) 220
13.13.30. DOT_PRODUCT (VECTOR_A, VECTOR_B) 220
13.13.31. DPROD(X,Y) 221
13.13.32. EOSHIFT (ARRAY, SHIFT, BOUNDARY, DIM)..... 221
13.13.33. EPSILON(X) 222
13.13.34. EXP (X) 222
13.13.35. EXPONENT (X) 223
13.13.36. FLOOR (A) 223
13.13.37. FRACTION (X) 223
13.13.38. HUGE (X) 223
13.13.39. IACHAR (C) 224
13.13.40. IAND (I, J) 224
13.13.41. IBCLR (I, POS) 224
13.13.42. IBITS (I, POS, LEN) 225
13.13.43. IBSET(I, POS) 225
13.13.44. ICHAR(C) 225
13.13.45. IEOR (I,J) 226
13.13.46. INDEX (STRING, SUBSTRING, BACK) 226
13.13.47. INT (A, KIND) 226
13.13.48. IOR (I, J) 227

Глава 13. ВСТРОЕННЫЕ ПРОЦЕДУРЫ

13.13.49. ISHFT (I, SHIFT) 227
13.13.50. ISHFTC (I, SHIFT, SIZE) 228
13.13.51. KIND (X) 228
13.13.52. LBOUND (ARRAY, DIM) 228
13.13.53. LEN (STRING) 229
13.13.54. LEN_TRIM (STRING) 229
13.13.55. LGE(STRING_A, STRING_B) 229
13.13.56. LGT (STRING_A, STRING_B) 230
13.13.57. LLE (STRING_A, STRING_B) 230
13.13.58. LLT(STRING_A, STRING_B) 230
13.13.59. LOG (X) 231
13.13.60. LOG10 (X) 231
13.13.61. LOGICAL (L, KIND) 231
13.13.62. MATMUL (MATRIX_A, MATRIX_B) 231
13.13.63. MAX (A1, A2, A3,...) 232
13.13.64. MAXEXPONENT(X) 233
13.13.65. MAXLOC (ARRAY, MASK) 233
13.13.66. MAXVAL(ARRAY, DIM, MASK) 234
13.13.67. MERGE (TSOURCE, FSOURCE, MASK) 235
13.13.68. MIN(A1, A2, A3,...) 235
13.13.69. MINEXPONENT(X) 235
13.13.70. MINLOC(ARRAY, MASK) 236
13.13.71. MINVAL (ARRAY, DIM, MASK) 236
13.13.72. MOD (A, P) 237
13.13.73. MODULO (A, P) 238
13.13.74. MVBITS (FROM, FROMPOS, LEN, TO, TOPOS) 238
13.13.75. NEAREST (X, S) 238
13.13.76. NINT(A, KIND) 239
13.13.77. NOT (I) 239
13.13.78. PACK (ARRAY, MASK, VECTOR) 239
13.13.79. PRECISION (X) 240
13.13.80. PRESENT (X) 240
13.13.81. PRODUCT (ARRAY, DIM, MASK) 240
13.13.82. RADIX (X) 241
13.13.83. RANDOM_NUMBER (HARVEST) 242
13.13.84. RANDOM_SEED (SIZE, PUT, GET) 242
13.13.85. RANGE (X) 242
13.13.86. REAL (A, KIND) 243
13.13.87. REPEAT (STRING, NCOPIES) 243
13.13.88. RESHAPE (SOURCE, SHAPE, PAD, ORDER) 243
13.13.89. RRSPACING (X) 244
13.13.90. SCALE (X, I) 245
13.13.91. SCAN (STRING, SET, BACK) 245
13.13.92. SELECTED_INT_KIND (R) 245
13.13.93. SELECTED_REAL_KIND (P, R) 246
13.13.94. SET_EXPONENT (X, I) 246
13.13.95. SHAPE (SOURCE) 247
13.13.96. SIGN (A, B) 247
13.13.97. SIN (X) 247

Глава 13. ВСТРОЕННЫЕ ПРОЦЕДУРЫ

13.13.98. SINH (X) ... 247
13.13.99. SIZE (ARRAY, DIM) ... 248
13.13.100. SPACING (X) ... 248
13.13.101. SPREAD (SOURCE, DIM, NCOPIES) ... 248
13.13.102. SQRT (X) ... 249
13.13.103. SUM (ARRAY, DIM, MASK) ... 249
13.13.104. SYSTEM_CLOCK (COUNT, COUNT_RATE, COUNT_MAX) ... 250
13.13.105. TAN(X) ... 250
13.13.106. TANH(X) ... 251
13.13.107. TINY(X) ... 251
13.13.108. TRANSFER (SOURCE, MOLD, SIZE) ... 251
13.13.109. TRANSPOSE (MATRIX) ... 252
13.13.110. TRIM (STRING) ... 252
13.13.111. UBOUND (ARRAY, DIM) ... 252
13.13.112. UNPACK (VECTOR, MASK, FIELD) ... 253
13.13.113. VERIFY (STRING, SET, BACK) ... 254

Глава 14. **ОБЛАСТЬ ДЕЙСТВИЯ, СВЯЗЬ И ОПРЕДЕЛЕННОСТЬ** ... 255

14.1. Область действия имен ... 255
14.1.1. Глобальные объекты ... 255
14.1.2. Локальные объекты ... 255
14.1.2.1. Общие блоки ... 256
14.1.2.2. Результаты функций ... 256
14.1.2.3. Однозначные обобщенные ссылки на процедуры ... 256
14.1.2.4. Разрешение ссылок на процедуры ... 257
14.1.2.5 Компоненты ... 259
14.1.2.6. Ключевые слова аргументов ... 259
14.1.3. Операторные объекты ... 259
14.2. Область действия меток ... 260
14.3. Область действия внешних устройств ввода/вывода ... 260
14.4. Область действия знаков операций ... 260
14.5. Область действия символа присваивания ... 260
14.6. Связывание ... 260
14.6.1. Связь по имени ... 260
14.6.1.1. Связь по аргументу ... 260
14.6.1.2. Связь по использованию и по объемлющему контексту ... 261
14.6.2. Связь по указателю ... 261
14.6.2.1. Статус связанности указателя ... 261
14.6.2.2. Статус определенности указателя ... 261
14.6.2.3. Взаимосвязь между статусом связанности и статусом определенности ... 261
14.6.3. Связь по памяти ... 262
14.6.3.1. Последовательность памяти ... 262
14.6.3.2. Связь последовательностей памяти ... 262
14.6.3.3. Связь скалярных объектов данных ... 263
14.7. Определенность и неопределенность переменных ... 264
14.7.1. Определение объектов и подобъектов ... 264

Глава 14. ОБЛАСТЬ ДЕЙСТВИЯ, СВЯЗЬ И ОПРЕДЕЛЕННОСТЬ
14.7.2. Переменные, которые всегда определены 264
14.7.3. Переменные, которые определены начально 264
14.7.4. Переменные, не определенные начально 265
14.7.5. События, которые приводят к определению объектов 265
14.7.6. События, приводящие к неопределенности 266
14.8. Статус размещенности 268

ПРИЛОЖЕНИЕ А. Словарь технических терминов..... 269

ПРИЛОЖЕНИЕ В. Устаревшие средства 281
В.1. Исключенные средства 281
В.2. Отживающие средства 281
В.2.1. Альтернативный возврат 281
В.2.2. Оператор PAUSE 282
В.2.3. Операторы ASSIGN и GO TO по предписанию 282
В.2.4. Присваиваемые спецификаторы формата 282
В.2.5. Описатель редактирования Н 282

ПРИЛОЖЕНИЕ С. Примечания к главам 283
С.1. Примечания к главе 1 283
С. 1.1. Соответствие стандарту (1.4) 283
С.2. Примечания к главе 2 283
С.2.1. Ключевые слова 283
С.3. Примечания к главе 3 283
С.3.1. Представимые символы (3.1.5.) 283
С.3.2. Строки комментария (3.3.1.1, 3.3.2.1) 283
С.3.3. Метки оператора (3.2.5) 284
С.3.4. Форма исходного текста (3.3) 284
С.4. Примечания к главе 4 284
С.4.1. Нуль (4.3.1) 284
С.4.2. Символы (4.2) 284
С.4.3. Встроенные и производные типы данных (4.3, 4.4) 285
С.4.4 Выбор способов аппроксимации 285
С.4.5. Память для производных типов (4.4.1) 286
С.4.6. Указатели 286
С.5. Примечания к главе 5 287
С.5.1. Операторы объявления типа (5.1) 287
С.5.2. Атрибут POINTER (5.1.2.7) 288
С.5.3. Атрибут TARGET (5.1.2.8) 288
С.5.4. Операторы PARAMETER и IMPLICIT NONE (5.2.10, 5.3) 289
С.5.5. Расширения оператора EQUIVALENCE (5.5.1) 289
С.5.6. Расширения оператора COMMON (5.5.2) 289
С.6. Примечания к главе 6 290
С.6.1. Подстроки (6.1.1) 290
С.6.2. Ссылки на элементы массива (6.2.2) 290
С.6.3. Компоненты структуры (6.1.2) 290
С.6.4. Размещение и связывание указателей 291
С.7. Примечания к главе 7 292
С.7.1. Символьное присваивание 292
С.7.2. Вычисление ссылок на функции 292
С.7.3. Указатели в выражениях 292
С.7.4. Указатели в левой части присваивания 292
С.8. Примечания к главе 8 293
С.8.1. Управление циклом 293
С.8.2. Конструкция CASE 293
С.8.3. Примеры неверных конструкций DO 293

C.9. Примечания к главе 9 294
C.9.1. Записи ввода/вывода (9.1) 294
C.9.2. Файлы (9.2) 294
C.9.2.1. Подсоединение файла (9.3) 294
C.9.2.2. Существование файла (9.2.1.1) 294
C.9.2.3. Имена файлов (9.3.4.1) 295
C.9.2.4. Доступ к файлу (9.2.1.2) 295
C.9.2.5. Ввод/вывод без продвижения (9.2.1.3.1) 295
C.9.3. Оператор OPEN (9.3.4) 296
C.9.4. Характеристики подсоединения (9.3.2) 298
C.9.5. Оператор CLOSE (9.3.5) 299
C.9.6. Оператор INQUIRE 299
C.9.7. Спецификаторы с ключевыми словами 299
C.9.8. Спецификации формата (9.4.1.1) 299
C.9.9. Неформатный ввод/вывод (9.4.4.4.1) 299
C.9.10. Ограничения на ввод/вывод 300
C.9.11. Указатели в списке ввода/вывода 300
C.9.12. Объекты производного типа в списке ввода/вывода (9.4.2) 301
C.10. Примечания к главе 10 301
C.10.1. Спецификация формата в виде символьной константы (10.1.2, 10.7.1) 301
C.10.2. Описатель редактирования T (10.6.1.1) 301
C.10.3. Длины форматных записей 301
C.10.4. Количество записей (10.3,10.4,10.6.2) 302
C.10.5. Ввод/вывод, управляемый списком (10.8) 302
C.10.6. Ввод, управляемый списком (10.8.1) 302
C.10.7. Элементы списка NAMELIST для ввода символьных данных (10.9.1.7) 303
C.10.8. Записи вывода по NAMELIST (10.9.2.2) 303
C.11. Примечания к главе 11 303
C.11.1. Главная программная единица и программная единица-блок данных (11.1, 11.4) 303
C.11.2. Зависимая компиляция (11.3) 303
C.11.2.1. Оператор USE и зависимая компиляция (11.3.2) 304
C.11.2.2. Атрибуты доступности (11.3.1) 305
C.11.3. Указатели в модулях 306
C.11.4. Пример модуля (11.3) 306
C.12. Примечания к главе 12 309
C.12.1. Примеры связи по объемлющему контексту (12.1.2.2.1) 309
C.12.2. Внешние процедуры (12.3.2.2) 310
C.12.3. Процедуры, определяемые способом, отличным от Фортрана (12.5.3) 310
C.12.4. Интерфейсы процедур (12.3) 310
C.12.5. Связь и вычисление аргументов (12.4.1) 311
C.12.6. Спецификация назначения аргументов (12.4.1.1) 312
C.12.7. Ограничения на формальные аргументы (12.5.2.9) 312
C.12.8. Указатели и адресаты в качестве аргументов 312
C.12.9. Функция ASSOCIATED (13.13.13) 313
C.12.10. Ограничения на внутренние процедуры 314
C.12.11. Результирующая переменная (12.5.2.2) 314

C.13. Примечания к главе 13 314
C.13.1. Резюме 314
C.13.1.1. Выражения и присваивания для массивов 314
C.13.1.2. Секции массива (2.4.5, 6.2.2.3) 314
C.13.1.3. Оператор WHERE (7.5.3) 315
C.13.1.4. Динамически и автоматически размещаемые массивы (5.1, 5.1.2.4.3) 315
C.13.1.5. Конструкторы массива (4.5) 315
C.13.1.6. Встроенные функции 315
C.13.2. Примеры 316
C.13.2.1. Безусловные вычисления массива 316
C.13.2.2. Условные вычисления массива 317
C.13.2.3. Простая программа: модель Изинга 318
C.13.2.3.1. Задачи, которые должны быть решены 318
C.13.2.3.2. Решения в Фортране 318
C.13.2.3.3. Полная подпрограмма на Фортране 319
C.13.2.3.4. Уменьшение памяти 320
C.13.3. Перевод формул и обработка массивов 320
C.13.3.1. Сумма произведений 320
C.13.3.2. Произведение сумм 321
C.13.3.3. Сложение выбираемых элементов 321
C.13.4. Сумма квадратичных отклонений 321
C.13.5. Нормы вектора: норма-бесконечность и норма-один 321
C.13.6. Нормы матрицы: норма-бесконечность и норма-один 322
C.13.7. Логические вопросы 322
C.13.8. Параллельные вычисления 322
C.13.9. Примеры поэлементного вычисления 323
C.13.10. Процедуры операций с битами 323
C.14. Примечания к главе 14 323
C.14.1. Связь по памяти для объектов нулевого размера 323

ПРИЛОЖЕНИЕ D. Синтаксические правила 325
D.1. Синтаксические правила и ограничения 325
D.1.1. Общие сведения 325
D.1.2. Термины и понятия Фортрана 325
D.1.3. Символы, лексические элементы и форма исходного текста 327
D.1.4. Встроенные и производные типы данных 329
D.1.5. Спецификации и описания объектов данных 331
D.1.6. Использование объектов данных 337
D.1.7. Выражения и присваивание 339
D.1.8. Управление выполнением 341
D.1.9. Операторы ввода/вывода 344
D.1.10. Редактирование ввода/вывода 347
D.1.11. Программные единицы 349
D.1.12. Процедуры 350
D.1.13. Встроенные процедуры 353
D.1.14. Область действия, связь и определенность 353
D.2. Перекрестные ссылки 353
D.2.1. Нетерминальные символы, которые определены 353
D.2.2. Нетерминальные символы, которые не определены ... 361
D.2.3 Терминальные символы 362

ПРИЛОЖЕНИЕ E. Переупорядоченный индекс для заголовков 366

ПРИЛОЖЕНИЕ F. ИНДЕКС 367