**Предисловие к русскому переводу**

Информация на русском языке о 8-разрядных компьютерах, операционных системах и компиляторах для них практически отсутствует. Утратив коммерческое значение они, тем не менее, представляют интерес с исторической точки зрения, отражая процесс развития программного обеспечения персональных компьютеров. Перевод руководства выполнен в 2016 году с целью сохранения компьютерного наследия, ознакомления с ранее довольно распространенным на больших ЭВМ языком программирования PL/I и как дань уважения программистам, реализовавшим его версию, работающую в очень ограниченном окружении на 8 разрядных микрокомпьютерах с 48 килобайтами ОЗУ.

В качестве исходного текста Справочного руководства PL/I для перевода на русский язык использован текст восстановленный 16.01.2001 Робином Энтони {Robin Vowels} с помощью программы распознавания текста из изображений исходных страниц, отсканированных Тимом Олмстедом {Tim Olmstead}. Некорректно восстановленные части текста исправлены и дополнены из других руководств Digital Research. Листинги программ восстановлены из файлов исходных текстов программ, включенных в дистрибутив компилятора. Структура Справочного руководства PL/I полностью соответствует оригиналу. В переведенной версии руководства нумерация приложений и листингов изменена с буквенной на цифровую.

Любые торговые марки, знаки и названия программ, служб и организаций, авторские и смежные права, которые упоминаются, используются или цитируются, принадлежат их законным владельцам и их использование здесь не дает вам право на любое другое использование.

Перевод данного руководства предназначен только для свободного ознакомления с возможностями языка программирования PL/I реализованного Digital Research для 8 и 16 разрядных компьютеров с исторической точки зрения и не подразумевает вознаграждения за перевод ни в каком виде. Распространение перевода данного руководства и информации приведенной в нем с целью получения прибыли недопустимо.

Перевод руководства может содержать технические неточности и опечатки. Вы можете использовать информацию из этого документа только на свой собственный страх и риск, в качестве альтернативы используя оригинальную версию.

Язык программирования PL/I первоначально разработанный IBM хотя и не стал распространенным, продолжает использоваться до настоящего времени на современных компьютерах. Версия PL/I Digital Research в настоящее время может использоваться в нескольких эмуляторах операционной системы CP/M, на сохранившихся 8-разрядных ретро компьютерах или микрокомпьютерах подходящей архитектуры до сих пор, собираемых и поддерживаемых энтузиастами (например, DX-Designs P112 или N8VEM).

**Язык**

**PL/I**

**Справочное руководство**

Авторское право © 1983

Digital Research

P.O. Box 579

160 Central Avenue

Pacific Grove, CA 93950

(408) 649-3896

TWX 910 360 5001

Все права защищены

**Авторское право**

Авторское право 1983 Digital Research. Все права защищены. Любая часть этой публикации не может быть воспроизведена, передана, записана, сохранена в поисковой системе или переведена на любой язык или компьютерный язык, в любой форме или каким-либо образом, электронным, механическим, магнитным, оптическим, химическим, ручным или иначе, без предварительного письменного разрешения Digital Research, Post Office Box 579, Pacific Grove, California, 93950.

**Исключение ответственности**

Digital Research не дает никаких заверений или гарантий в отношении содержания данного документа, включая любые гарантии товарности или пригодности для какой-либо конкретной цели. Кроме того, Digital Research сохраняет за собой право пересмотреть эту публикацию и время от времени вносить изменения в содержание настоящего документа без обязательства Digital Research уведомить любое лицо, о таком пересмотре или изменениях.

**Торговые марки**

CP/M - зарегистрированная торговая марка Digital Research. PL/I-80, и PL/I-86 являются торговыми марками Digital Research. SP/k - торговая марка университета Торонто. IBM - зарегистрированная торговая марка International Business Machines, Incorporated. DEC - зарегистрированная торговая марка Digital Equipment Corporation. Z80 - зарегистрированная торговая марка Zilog, Inc, Intel - зарегистрированная торговая марка Intel Corporation. MicroSoft - зарегистрированная торговая марка MicroSoft Incorporated.

Справочное руководство языка PL/I было подготовлено, используя средство форматирования текста TEX-80 Digital Research и распечатано в Соединенных Штатах Америки. [[1]](#footnote-1)

Первое издание: октябрь 1982

Второе издание: июнь 1983

**Предисловие**

Язык программирования PL/I Digital Research представляет собой полную систему разработки программного обеспечения для приложений и системного программирования. Digital Research реализовала PL/I для 8-разрядных и 16-разрядных микропроцессоров. На уровне исходного кода 16-разрядные версии совместимы снизу вверх с 8-разрядными версиями. *Приложение 1* содержит полный список отличий между различными версиями. Это руководство описывает язык PL/I, являющийся общим для всех версий.

Язык PL/I Digital Research работает в любой операционной системе Digital Research. Он также работает в дисковой операционной системе версии 1.1 на персональном компьютере IBM. Это руководство предполагает, что вы уже знакомы с операционной системой, и сводит к минимуму ссылки на конкретные системы.

Справочное руководство языка PL/I является формальной спецификацией языка программирования PL/I. Это руководство, прежде всего, является справочником и по своему содержанию не является учебником. Предполагается некоторый предыдущий опыт программирования на PL/I или на другом языке программирования.

Справочное руководство языка описывает общую структуру и организацию исходных программ PL/I в форме блоков и процедур. Также приведена спецификация набора символов языка, правил, управляющих формированием идентификаторов, констант, разделителей, операторов и комментариев.

Это руководство объясняет различные типы данных PL/I, включая массивы и структуры, правила преобразования типов данных и правила, регулирующие область действия объявлений данных. Также описаны присвоения и выражения, управление последовательностью выполнения, управление памятью во время выполнения и обработка ввода-вывода.

Приводится полное описание всех встроенных функций PL/I, включая арифметические, математические, строковые, преобразования, условий и прочих функций.

Наконец, в руководстве также присутствует описание внутреннего представления данных в различных версиях и соглашения для взаимодействия программ на языке PL/I с программами, написанными на языке ассемблера.

**Содержание**

Раздел 1 Ведение 11

1.1 Комплект документации 11

1.2 Обозначения 11

Раздел 2 Структура программы 12

2.1 Организация верхнего уровня 12

2.2 Блоки 13

2.3 Внешние и внутренние блоки 15

2.4 Область действия переменных 16

2.5 Процедуры 18

2.6 Оператор CALL 18

2.7 Оператор RETURN 18

2.8 Формальные и фактические параметры 19

2.9 Оператор PROCEDURE 21

2.10 Организация программы на нижнем уровне 23

2.11 Набор символов 23

2.12 Идентификаторы 24

2.13 Константы 25

2.14 Ограничители и разделители 25

2.14.1 Пустое пространство 25

2.14.2 Операции 25

2.14.3 Специальные символы 26

2.14.4 Комментарии 27

2.15 Операторы препроцессора 27

2.15.1 Оператор %INCLUDE 27

2.15.2 Оператор %REPLACE 28

Раздел 3 Типы и атрибуты данных 29

3.1 Арифмитические данные 29

3.1.1 Тип данных FIXED BINARY 29

3.1.2 Тип данных FLOAT BINARY 30

3.1.3 Тип данных FIXED DECIMAL 31

3.2 Строковые данные 32

3.2.1 Данные типа символьная строка 32

3.2.2 Данные типа битовая строка 32

3.3 Данные управления программой 33

3.3.1 Данные типа LABEL 33

3.3.2 Данные типа ENTRY 35

3.4 Данные типа POINTER 36

3.5 Данные типа FILE 36

3.6 Оператор DECLARE 37

3.7 Составной оператор DECLARE 37

3.8 Атрибуты по умолчанию 38

Раздел 4 Преобразования данных 39

4.1 Арифметические преобразования 40

4.2 Функции арифметических преобразований 42

4.2.1 Встроенная функция BINARY 42

4.2.2 Встроенная функция DECIMAL 43

4.2.3 Встроенная функция DIVIDE 43

4.2.4 Встроенная функция FIXED 43

4.2.5 Встроенная функция FLOAT 44

4.3 Преобразования строк 44

4.3.1 Преобразование арифметического типа в битовую строку 44

4.3.2 Преобразование арифметического типа в символьную строку 45

4.3.3 Преобразование битовой строки в арифметический тип 46

4.3.4 Преобразование битовой строки в символьную 46

4.3.5 Преобразование символьной строки в арифметический тип 46

4.3.6 Преобразование символьной строки в битовую 47

Раздел 5 Агрегаты данных 48

5.1 Объявление массива 48

5.2 Ссылки на массивы 49

5.3 Инициализация элементов массива 50

5.4 Массивы в операторах присваивания 52

5.5 Структуры 53

5.6 Смешанные агрегаты данных 55

5.7 Ссылки на смешанные агрегаты данных 56

Раздел 6 Присваивания и выражения 57

6.1 Оператор присваивания 57

6.2 Выражения 57

6.2.1 Префиксные выражения 57

6.2.2 Инфиксные выражения 57

6.3 Приоритет выполнения операций 58

6.4 Конкатенация (сцепление) 58

6.5 Операции сравнения 59

6.6 Операции для работы с битовыми строками 60

6.7 Возведение в степень 60

6.8 Псевдопеременные 60

6.8.1 Встроенная символьная функция SUBSTR 61

6.8.2 Встроенная битовая функция SUBSTR 61

6.8.3 Встроенная функция UNSPEC 62

Раздел 7 Управление памятью 63

7.1 Классы памяти 63

7.1.1 Класс памяти AUTOMATIC 63

7.1.2 Класс памяти BASED 63

7.1.3 Класс памяти PARAMETER 65

7.1.4 Класс памяти STATIC 65

7.2 Оператор ALLOCATE 66

7.3 Многократное выделение памяти 66

7.4 Оператор FREE 67

7.5 Встроенная функция NULL 67

7.6 Встроенная функция ADDR 68

7.7 Совместное использование памяти 68

7.8 Замечания по программированию указателей 69

Раздел 8 Управление последовательностью выполнения 71

8.1 Простой оператор DO 71

8.2 Управляемый оператор DO 71

8.2.1 Оператор DO WHILE 72

8.2.2 Оператор DO REPEAT 73

8.2.3 Оператор DO REPEAT WHILE 73

8.2.4 Оператор DO BY WHILE 74

8.3 Оператор IF 75

8.4 Оператор STOP 75

8.5 Оператор GOTO 75

8.6 Оператор нелокального GOTO 76

Раздел 9 Обработка состояний 77

9.1 Оператор ON 77

9.2 Оператор SIGNAL 78

9.3 Оператор REVERT 78

9.4 Состояние ERROR 79

9.5 Состояния арифметических ошибок 80

9.6 Встроенная функция ONCODE 80

9.7 ON-блоки по умолчанию 81

9.8 Состояния ввода-вывода 81

Раздел 10 Обработка ввода и вывода 82

10.1 Оператор OPEN 82

10.2 Назначение атрибутов файлов 85

10.3 Оператор CLOSE 87

10.4 Блок параметров файла 88

10.5 Состояния ввода-вывода 88

10.5.1 Состояние ENDFILE 88

10.5.2 Состояние UNDEFINEDFILE 89

10.5.3 Состояние KEY 89

10.5.4 Состояние ENDPAGE 89

10.5.5 ON-блоки ввода-вывода по умолчанию 89

10.6 Встроенные функции состояний ввода-вывола 89

10.6.1 Функция ONFILE 90

10.6.2 Функция ONKEY 90

10.6.3 Функция PAGENO 90

10.6.4 Функция LINENO 90

10.7 Предопределенные файлы SYSIN и SYSPRINT 90

10.8 Категории ввода-вывода 91

10.8.1 Потоковый (STREAM) ввод-вывод 91

10.8.2 Записеориентированный (RECORD) ввод-вывод 91

Раздел 11 Потоковый ввод-вывод 92

11.1 Ввод-вывод управляемый списком 93

11.1.1 Оператор GET LIST 93

11.1.2 Оператор PUT LIST 93

11.2 Строчно-ориентированный ввод-вывод 94

11.2.1 Оператор READ VARYING 94

11.2.2 Оператор WRITE VARYING 95

11.3 Ввод-вывод управляемый редактированием 95

11.3.1 Список форматов 95

11.3.2 Элементы формата данных 96

11.3.3 Элементы формата управления 97

11.3.4 Элементы удаленного формата 99

11.3.5 Оператор FORMAT 99

11.3.6 Элементы формата шаблон 99

11.3.7 Оператор GET EDIT 104

11.3.8 Оператор PUT EDIT 105

Раздел 12 Записеориентированный ввод-вывод 106

12.1 Оператор READ 106

12.2 Оператор READ с KEY 106

12.3 Оператор READ с KEYTO 106

12.4 Оператор WRITE 107

12.5 Оператор WRITE с KEYFROM 107

Раздел 13 Встроенные функции 108

13.1 Арифметические функции 108

13.2 Математические функции 108

13.3 Функции обработки строки 109

13.4 Функции преобразования строк 109

13.5 Функции обработки состояний 109

13.6 Вспомогательные функции 109

13.7 Список встроенных функций 110

Раздел 14 Операторы PL/I 126

14.1 Оператор ALLOCATE 126

14.2 Оператор присваивания 126

14.3 Оператор BEGIN 126

14.4 Оператор CALL 126

14.5 Оператор CLOSE 126

14.6 Оператор DECLARE (для скалярных переменных) 126

14.7 Оператор DECLARE (для массивов) 126

14.8 Оператор DECLARE (для структур) 127

14.9 Оператор DECLARE (для данных ENTRY) 127

14.10 Оператор DECLARE (для данных FILE) 127

14.11 Оператор DO 127

14.12 Оператор END 128

14.13 Оператор FORMAT 128

14.14 Оператор FREE 128

14.15 Оператор GET EDIT 128

14.16 Оператор GET LIST 128

14.17 Оператор GOTO 128

14.18 Оператор IF 128

14.19 Оператор %INCLUDE 129

14.20 Оператор NULL 129

14.21 Оператор ON 129

14.22 Оператор OPEN 129

14.23 Оператор PROCEDURE 129

14.24 Оператор PUT EDIT 129

14.25 Оператор PUT LIST 130

14.26 Оператор READ VARYING 130

14.27 Оператор READ (для файлов SEQUENTIAL RECORD) 130

14.28 Оператор READ с KEY 130

14.29 Оператор READ с KEYTO 130

14.30 Оператор %REPLACE 130

14.31 Оператор RETURN 130

14.32 Оператор REVERT 131

14.33 Оператор SIGNAL 131

14.34 Оператор STOP 131

14.35 Оператор WRITE VARYING (для файлов STREAM) 131

14.36 Оператор WRITE (для файлов SEQUENTIAL RECORD) 131

14.37 Оператор WRITE с KEYFROM 131

Раздел 15 Атрибуты данных 132

15.1 ALIGNED 132

15.2 AUTOMATIC | AUTO 132

15.3 BASED или BASED(p) или BASED(q()) 132

15.4 BINARY | BIN или BINARY(p) | BIN(p) 132

15.5 BIT(n) 132

15.6 BUILTIN 132

15.7 CHARACTER(n) | CHAR(n) 133

15.8 DECIMAL[(p[,q])] | DEC[(p[,q])] 133

15.9 ENTRY[(список\_параметров)] 133

15.10 ENVIRONMENT(параметры) | ENV(параметры) 133

15.11 EXTERNAL | EXT 133

15.12 FILE 133

15.13 FIXED[(p[,q])] 134

15.14 FLOAT[(p)] 134

15.15 INITIAL(список\_значений) | INIT(список\_значений) 134

15.16 LABEL 134

15.17 PARAMETER 134

15.18 POINTER | PTR 134

15.19 RETURNS(список\_атрибутов) 134

15.20 STATIC 135

15.21 VARIABLE 135

15.22 VARYING | VAR 135

Приложение 1 Замечания по реализации 136

Приложение 2 Внутреннее представление данных 140

Приложение 3 Интерфейсные соглашения 148

Приложение 4 Опции компилятора 158

Приложение 5 Сообщения об ошибках и коды состояний 159

Приложение 6 Коды ASCII 168

Приложение 7 Библиография PL/l 170

Приложение 8 Перечень терминов 173

Предметный указатель 179

**Таблицы,** рисунки **и листинги**

**Таблицы**

Таблица 2‑1 Символы PL/I 23

Таблица 2‑2 Знаки операций PL/I 26

Таблица 2‑3 Специальные символы ограничителей и разделителей 26

Таблица 3‑1 Числа PL/I FLOAT BINARY 30

Таблица 3‑2 Форматы константы битовой строки 33

Таблица 4‑1 Общие типы операндов в смешанных выражениях 40

Таблица 4‑2 Встроенные функции преобразования. 44

Таблица 4‑3 Преобразование символьной строки в арифметический тип 46

Таблица 6‑1 Приоритет операций PL/I 58

Таблица 6‑2 Операции PL/I для работы с битовыми строками 60

Таблица 9‑1 Коды состояний арифметических ошибок 80

Таблица 10‑1 Имена внешних устройств 84

Таблица 10‑2 Подразумеваемые атрибуты PL/I 85

Таблица 10‑3 Допустимые файловые атрибуты операторов IO 86

Таблица 10‑4 Допустимые файловые атрибуты 86

Таблица 11‑1 Соглашения о присвоении имен потокового ввода- вывода 92

Таблица 11‑2 Символы формата шаблона 99

Таблица 11‑3 Выходные символы шаблона 100

Таблица 11‑4 Вывод отредактированный шаблоном 102

Таблица 11‑5 Вывод отредактированный шаблоном 103

**Рисунки**

Рисунок 2‑1 Блоки BEGIN и PROCEDURE 14

Рисунок 2‑2 Внешние и внутренние блоки 15

Рисунок 2‑3 Вызов подпрограммы и функции 18

Рисунок 2‑4 Фактические и формальные параметры 19

Рисунок 5‑1 Двумерный массив 49

Рисунок 5‑2 Ссылки на элементы массива 49

Рисунок 5‑3 Инициализация массива 50

Рисунок 5‑4 Иерархия уровней структуры 54

Рисунок 5‑5 Иерархия уровней структуры 55

Рисунок 5‑6 Массив структур 55

Рисунок 5‑7 Массив структур и структура массивов 56

Рисунок 7‑1 Многократные выделения памяти базированной переменной 66

Рисунок 7‑2 Связный список 68

Рисунок 8‑1 Форма оператора DO 71

Рисунок 8‑2 Оператор DO WHILE 72

Рисунок 8‑3 Оператор DO REPEAT 73

Рисунок 8‑4 Оператор DO REPEAT WHILE 73

Рисунок 8‑5 Оператор DO BY WHILE 74

Рисунок 9‑1 Активация ON-блока 78

Рисунок 11‑1 Распознаватель спецификации шаблона 100

**Листинги**

Листинг 1 Процедура преобразования формата с плавающей точкой 143

Листинг 2 Программа DTEST 149

Листинг 3 DIV2.ASM Программа на языке ассемблера (8080) 150

Листинг 4 DIV2.A86 Программа на языке ассемблера (8086) 151

Листинг 5 Вывод DTEST (Сокращенный) 152

Листинг 6 Программа FDTEST 155

Листинг 7 FDIV2.ASM Программа на языке ассемблера (8080) 156

Листинг 8 FDIV2.A86 Программа на языке ассемблера (8086) 157

# Ведение

Язык PL/I Digital Research представляет собой версию PL/I для микрокомпьютеров, которые используют 8080, 8086, 8088 или аналогичные процессоры. Формально он основан на американском национальном стандарте X3.74 подмножестве PL/I общего назначения (подмножестве G). Подмножество G имеет формальную структуру полного языка PL/I, но в некотором смысле это новый язык и во многом улучшенный по сравнению с полной версией PL/I.

Язык PL/I подмножества G простой в освоении и использовании. Он является переносимым, так как его конструкция обычно обеспечивает независимость от оборудования. Кроме того, он более эффективный и экономичный. Программы, написанные на языке PL/I подмножества G, проще реализовывать, документировать и поддерживать.

## Комплект документации

*Справочное руководство языка PL/I* представляет собой подробное, но краткое описание языка программирования PL/I. Это не учебное пособие по программированию на PL/I, скорее, это функциональное описание языка, его синтаксиса и семантики. Это руководство является справочником, дополняющим *Руководство программиста языка PL/I* Digital Research.

*Руководство программиста языка PL/I* включает примеры программ, которые иллюстрируют многие функции PL/I, а также практические аспекты компиляции и компоновки программ. Если вы прежде не программировали на PL/I, сначала прочтите *Руководство программиста*, используя перекрестные ссылки на определенные темы справочного руководства. Если вы уже опытный программист PL/I, то можете прочитать только *Справочное руководство*.

Краткое описание команд языка PL/I перечисляет все ключевые слова PL/I и формы операторов, атрибуты данных и сообщения об ошибках. Оно также содержит краткую информацию о командах компилятора.

## Обозначения

В этом документе используются следующие условные обозначения:

* Слова, написанные заглавными буквами, являются ключевыми словами PL/I.
* Слова в нижнем регистре или в комбинации из строчных букв и цифр, разделенные подчеркиванием представляют переменную информацию выбираемую вами. Более подробно эти слова описаны или определены в тексте.
* Примеры операторов приводятся в нижнем регистре.
* Вертикальная черта | означает альтернативу.
* ⌴ представляет собой пробел.
* Квадратные скобки [ ] включают параметры.
* Многоточие (...) свидетельствуют о том, что непосредственно предшествующий ему элемент может встретиться один, или несколько раз подряд.
* За исключением специальных символов, перечисленных выше, все остальные знаки препинания и специальные символы представляют фактическое появление этих символов.
* В тексте символ CTRL представляет управляющий символ. Таким образом, CTRL-C означает Ctrl-C. В листинге исходной программы PL/I или любом листинге, который показывает пример взаимодействия с консолью, символ представляет собой управляющий символ.
* Ввод на клавиатуре и его отображение на экране выделено жирным шрифтом.

**Конец раздела 1**

# Структура программы

## Организация верхнего уровня

Следующие операторы содержатся в каждой программе PL/I:

* Структурные операторы;
* Операторы объявления;
* Выполняемые операторы.

*Структурные операторы* определяют отдельные логические блоки в пределах программы и, таким образом, определяют в целом организацию программы верхнего уровня. При выполнении программы управление всегда следует из одного из этих логических блоков в другой. Логические блоки могут содержать другие логические блоки. Они могут быть вложены. Структурные операторы также определяют иерархическую структуру программы, в которой некоторые логические блоки подчинены другим.

*Операторы объявления* определяют среду логического блока. Средой являются имена и атрибуты переменных, которые используются или активизируются в логическом блоке. Операторы объявления определяют переменные, которые могут правильно обрабатываться в логическом блоке.

*Выполняемыми* являются операторы, которые осуществляют некоторое действие. Структурные операторы и операторы объявления служат только для создания среды и контекста для выполняемых операторов. Все выполняемые операторы можно отнести к одной из следующих категорий:

* *Операторы присваивания*, которые присваивают переменной значение выражения или константы;
* *Операторы обработки состояния*, которые позволяют программе перехватывать и обрабатывать ошибки времени выполнения;
* *Операторы ввода-вывода*, которые управляют потоком данных в/из устройств ввода-вывода;
* *Операторы управления памятью*, которые распределяют память.
* Пустые операторы, которые не выполняют действий, но служат местом для заполнения;
* *Операторы препроцессора*, которые выполняются во время компиляции и управляют внешними исходными файлами;
* *Операторы передачи управления*, которые передают управления между логическими блоками.

Все операторы PL/I, кроме оператора присваивания, состоят из необязательной метки, за которой следует ключевое слово и тело оператора и завершаются точкой с запятой. Следующие разделы этого руководства подробно описывают каждый тип операторов. Для справки *Приложение 2* содержит полный список форматов операторов PL/I в алфавитном порядке.

## Блоки

PL/1 - язык с блочной структурой. Это означает, что каждая логическая часть программы, состоящая из одного или более операторов, является блоком. Блок - это набор операторов, в которых используются объявленные переменные. Внутри блока можно объявить переменные и для некоторых переменных, выделить, а затем освободить память. Можно вкладывать блоки друг в друга, но их границы не могут пересекаться.

Существуют два типа блоков: блоки BEGIN и блоки PROCEDURE. В блоке BEGIN последовательность операторов ограничивается операторами BEGIN и END. Операторы PROCEDURE и END ограничивают блок PROCEDURE.

Исходный текст вашей программы должен представлять собой один блок PROCEDURE (если это главный модуль) или, возможно, один BEGIN-блок с меткой (если это не главный модуль).

Блок BEGIN имеет следующий вид:

**[метка:]**

**BEGIN;**

**оператор\_1;**

**.**

**.**

**.**

**оператор\_n;**

**END [метка];**

где оператор\_1 ... оператор\_n - любые операторы PL/I, составляющие тело блока. Блоки BEGIN могут содержать вложенные блоки PROCEDURE, и вложенные блоки BEGIN. В PL/I необязательная метка в операторе END не приводит к автоматическому закрытию всех открытых блоков, как это происходит в некоторых полных версиях языка.

Блок PROCEDURE имеет следующий вид:

**имя\_процудуры:**

**PROCEDURE-оператор;**

**оператор\_1;**

**.**

**.**

**.**

**оператор\_n;**

**END [имя\_процудуры];**

где имя\_процедуры идентифицирует процедуру, и оператор\_1 ... оператор\_n являются любыми операторами PL/I, составляющими тело блока. [*Раздел 2.9*](#_Оператор_PROCEDURE) описывает оператор PROCEDURE.

**Примечание**: имя\_процедуры в операторе END необязательное, но если оно присутствует, то должно соответствовать метке имя\_процедуры в операторе PROCEDURE.

Существенное различие между блоком BEGIN и блоком PROCEDURE является то, как они получают управление, во время выполнения программы. Управление переходит в блок BEGIN обычным последовательным способом. В этот момент блок становится активным. Когда управление программой передается вне блока, или выполняется его соответствующий оператор END, блок завершается.

PL/I пропускает блоки PROCEDURE во время обычной последовательности выполнения, и они получают управления только при вызове (см. [*Раздел 2.5*](#_Процедуры)). *Рисунок 2-1* иллюстрирует концепцию блоков.

┌──────────────────────────────┐ ┌──────────────────────────────┐

│ **PROCEDURE-блок** │ │ **BEGIN-блок** │

│ │ │ │

│ ┌─ А: │ │ ┌─ begin; │

│ │ procedure options(main); │ │ │ . │

│ │ . │ │ │ . │

│ │ . │ │ │ . │

│ │ оператор(ы) │ │ │ оператор(ы) │

│ │ . │ │ │ . │

│ │ . │ │ │ . │

│ │ . │ │ │ . │

│ │ │ │ │ │

│ └─ end А; │ │ └─ end; │

└──────────────────────────────┘ └──────────────────────────────┘

┌─────────────────────────────────────┐

│ │

│ ┌─ А: │

│ │ procedure options(main); │

│ │ . │

│ │ . │

│ │ . │

│ │ ┌─ begin; │

│ │ │ . │

│ │ │ . │

│ │ │ . │

│ │ └─ end; │

│ │ │

│ │ │

│ │ ┌─ begin; │

│ │ │ . │

│ │ │ . │

│ │ │ . │

│ │ │ │

│ │ │ ┌─ begin; │

│ │ │ │ . │

│ │ │ │ . │

│ │ │ │ . │

│ │ │ └─ end; │

│ │ │ │

│ │ └─ end; │

│ │ │

│ │ │

│ │ │

│ └─ end А; │

│ │

└─────────────────────────────────────┘

Рисунок 2‑1 Блоки BEGIN и PROCEDURE

## Внешние и внутренние блоки

Каждый блок характеризуется как внутренний или внешний по отношению к другим блокам. Внутренняя процедура находится в теле блоков. Внешняя процедура расположена отдельно от других блоков. Процедура не входит (вложена) ни в один блок. Таким образом, *главная процедура* всегда является внешней.

Программа PL/I может состоять из одного или нескольких внешних процедур, которые содержат вложенные внутренние процедуры или блоки. Каждая внешняя процедура может быть скомпилирована индивидуально и скомпонована для формирования выполняемой программы. Одна из внешних процедур, формирующих программу, должна быть *главной процедурой*.

На *Рисунке 2-2 (а)* блоки P1, P2 и P3 внешние, а блок BEGIN внутренний по отношению к P3. На *Рисунке 2-2 (б)*, P1 - внешний блок, а P2, P3 и блок BEGIN внутренний.

┌────────────────────────────────┐ ┌────────────────────────────────┐

│ ┌─ P1: │ │ ┌─ P1: │

│ │ procedure options(main); │ │ │ procedure options(main); │

│ │ │ │ │ │

│ │ │ │ │ │

│ └─ end P1; │ │ │ │

│ │ │ │ │

│ ┌─ P2: │ │ │ ┌─ P2: │

│ │ procedure; │ │ │ │ procedure; │

│ │ │ │ │ │ │

│ │ │ │ │ │ │

│ └─ end P2; │ │ │ └─ end P2; │

│ │ │ │ │

│ ┌─ P3: │ │ │ ┌─ P3: │

│ │ procedure; │ │ │ │ procedure; │

│ │ │ │ │ │ │

│ │ ┌─ begin; │ │ │ │ ┌─ begin; │

│ │ │ │ │ │ │ │ │

│ │ │ │ │ │ │ │ │

│ │ │ │ │ │ │ └─ end; │

│ │ └─ end; │ │ │ │ │

│ │ │ │ │ └─ end P3; │

│ │ │ │ │ │

│ └─ end P3; │ │ └─ end P1; │

│ (а) │ │ (б) │

└────────────────────────────────┘ └────────────────────────────────┘

Рисунок 2‑2 Внешние и внутренние блоки

Главная процедура выглядит следующим образом:

**имя\_процедуры:**

**PROCEDURE OPTIONS(MAIN);**

**.**

**.**

**операторы или блоки**

**.**

**.**

**END [имя\_процедуры];**

*Руководство программиста языка PL/I* содержит некоторые примеры структурированных программ и инструкции как отдельно скомпилировать, скомпоновать и загрузить внешние процедуры.

## Область действия переменных

*Область действия переменной* определяется блоком в котором объявлена переменная. Переменные могут быть *локальными* или *внешними* по отношению к блоку, в котором они используются.

Когда вы объявляете переменную в блоке, вы можете сослаться на нее в этом блоке или любом блоке внутри этого блока. Переменная называется *локальной* для этого блока, так как вы не можете ссылаться на нее за пределами блока, в котором вы ее объявили. Во внутреннем блоке ссылку на переменную, объявленную в блоке содержащим внутренний, называют *ссылкой верхнего уровня*.

Следующий пример иллюстрирует понятие области действия:

**┌── P1:**

**│ procedure;**

**│ declare**

**│ (a,b) fixed binary(7);**

**│ a = 2; /\* a локальна в P1 \*/**

**│ b = 3; /\* b локальна в P1 \*/**

**│ ┌── P2:**

**│ │ procedure;**

**│ │ declare**

**│ │ b fixed binary(7);**

**│ │ b = 2; /\* b локальна в P2**

**│ │ a = a\*b; /\* b это b из P2, но не b из P1**

**│ └── end P2;**

**│**

**│ put list (a,b);**

**│**

**└── end P1;**

PL/I создает новую переменную b в блоке P2, потому что она является переменной, объявленной в этом блоке. Оператор PUT LIST расположен вне P2, поэтому, значение переменной b в P1 равняется 3. Поскольку объявление идентификатора в P2 отсутствует, действует ссылка верхнего уровня на переменную, объявленную в P1, и оператор присваивания в P2 изменяет свое значение. Таким образом, этот фрагмент программы выводит значения 4 и 3.

Любая переменная, объявленная как EXTERNAL (внешняя), известна всем блокам, в которых она объявлена EXTERNAL и во всех внутренних блоках, если не объявлена повторно без атрибута EXTERNAL. Два объявления переменной с одинаковым именем означает разные области памяти для этих переменных, если оба объявления не содержат атрибут EXTERNAL.

**┌── Pl:**

**│ procedure;**

**│ declare**

**│ z fixed binary external;**

**│ .**

**│ .**

**│ ┌── P2:**

**│ │ procedure;**

**│ │ declare**

**│ │ z fixed binary external;**

**│ │ .**

**│ │ .**

**│ │ ┌── P3:**

**│ │ │ ┌── begin;**

**│ │ │ │ declare**

**│ │ │ │ z float binary; /\* не внешняя \*/**

**│ │ │ │ .**

**│ │ │ │ .**

**│ │ │ └── end;**

**│ │ └── end P3;**

**│ └── end P2;**

**└── end P1;**

В этой последовательности кода переменная z в P1 и P2 является одной и той же внешней переменной, а переменная z в P3 является локальной переменной и отличается от внешней переменной z.

**┌── P1:**

**│ procedure options(main);**

**│ declare x float binary;**

**│ .**

**│ .**

**│ ┌── begin;**

**│ │ declare x fixed;**

**│ │ .**

**│ │ .**

**│ └── end;**

**│ .**

**│ .**

**│ ┌── P2:**

**│ │ procedure;**

**│ │ declare x character(10) varying;**

**│ │ .**

**│ │ .**

**│ │ .**

**│ └── end P2;**

**│**

**└── end P1;**

В этой последовательности кода область действия переменной x ограничена каждым блоком, в котором она объявлена. Несмотря на то, что в каждом объявлении имя идентично, компилятор обрабатывает каждую как абсолютно другую переменную с ее собственным типом данных и выделяет для них различные области памяти.

## Процедуры

В PL/I имеются два типа процедур: *подпрограммы* и *функции*. Оба типа выполняют определенную задачу и логически отделены от остальной части программы. Оба типа могут выполнить ту же последовательность кода один или несколько раз, при этом код присутствует в программе только один раз.

Вы вызываете подпрограмму и, при необходимости, передаете ей элементы данных в списке аргументов. Затем подпрограмма обрабатывает данные и, при необходимости, возвращает их процедуре вызова. После окончания работы процедуры, управление передается оператору, расположенному непосредственно после оператора вызова процедуры.

Функция является процедурой, которая обрабатывает элементы данных и затем возвращает единственное значение. Вы вызываете функцию, ссылаясь на нее по имени функции со списком аргументов в выражении. Управление передается функции, которая выполняет свою задачу и затем возвращает единственное значение, которое заменяет ссылку функции и выражение продолжает вычисляться.

## Оператор CALL

Общий вид оператора вызова процедуры CALL:

**CALL Имя\_процедуры((инд\_1,...,инд\_n)] [(Список\_аргументов)];**

где инд\_1 … инд\_n необязательные индексы, которые необходимы только когда имя процедуры является индексированной переменной ([*Раздел 3.3.2*](#_Данные_типа_FILE)), и Список аргументов - аргументы, передаваемые процедуре. На *Рисунке 2-3* показаны вызовы процедуры и функции.

**Вызов процедуры: Вызов функции:**

**┌─** Имя подпрограммы **┌─** Имя функции

**│ ┌─** Параметры **│ ┌─** Параметры

V V V V

**CALL Имя Список\_аргументов Имя Список\_аргументов**

Рисунок 2‑3 Вызов подпрограммы и функции

Примеры:

**call print\_header; point = 3.14/sin(A);**

**call compute(base\_pay,overtime); put list (Sum(X, Y));**

## Оператор RETURN

Оператор RETURN возвращает управление в точку в блоке вызова непосредственно следующую за вызовом процедуры. Он также возвращает значение, если находится в процедуре-функции.

Оператор RETURN имеет следующий вид:

**RETURN [(возвращаемое\_значение)];**

где возвращаемое\_значение - значение функции, возвращаемое подпрограммой в точку вызова. Когда необходимо, PL/I преобразует атрибуты возвращаемого значения, чтобы соответствовать атрибутам, определенным RETURNS в операторе PROCEDURE. (См. [*Раздел 4.1*](#_Арифметические_преобразования_1)).

Оператор RETURN завершает содержащий его блок процедуры. Если главная процедура имеет атрибут RETURNS, PL/I возвращает управление в операционную систему. Ниже приведены некоторые примеры операторов RETURN:

**return;**

**return (X\*\*2);**

**return (F(A, (B)));**

## Формальные и фактические параметры

Элементы данных, которые передаются в процедуру, называют *аргументами* или фактическими параметрами, в то время как элементы данных, ожидаемые процедурой и, описанные в операторе PROCEDURE, называются *формальными параметрами*. При вызове процедуры, PL/I ставит в соответствие каждому формальному параметру его фактический параметр. *Рисунок 2-4* иллюстрирует это понятие.

**┌─** Фактические параметры (аргументы)

V

**──────────────────────**

**CALL COMPUTE(A + (B + C), R/2, 3.14);**

.

. **┌─** Формальные параметры

**.** V

**───────**

**COMPUTE: PROCEDURE (X, Y, Z);**

.

.

**END COMPUTE;**

Рисунок 2‑4 Фактические и формальные параметры

При передаче параметра *ссылкой* (по имени), формальный параметр и соответствующий ему фактический параметр совместно используют общую память. В этом случае любые изменения, формального параметра в вызванной процедуре, приводят к изменению значения фактического параметра в блоке вызова.

При передаче параметра *значением*, формальный и фактический параметры совместно не используют общую память. В этом случае PL/I передает вызванной процедуре копию формального параметра, так что любые изменения формального параметра влияют только на копию, а не значенин фактического параметра.

Следующий пример программы иллюстрирует передачу параметров.

**┌── A:**

**│ procedure;**

**│ declare**

**│ ACTUAL fixed binary,**

**│ DUMMY fixed binary;**

**│**

**│ call X(ACTUAL);**

**│ call X((DUMMY));**

**│**

**│ ┌── X:**

**│ │ procedure (FORMAL);**

**│ │ declare FORMAL fixed binary;**

**│ │ FORMAL = 3;**

**│ └── end X;**

**└── end A;**

PL/I передает ACTUAL ссылкой. Поэтому, оператор присваивания в процедуре X изменяет значение ACTUAL во всей программе. PL/I передает DUMMY значением. Таким образом процедура изменяет только копию значения в процедуре.

PL/I передает параметры ссылкой, когда атрибуты данных фактического параметра совпадают с атрибутами данных формального параметра. PL/I передает параметр значением, когда он является:

* константой;
* именем точки входа (подпрограммы);
* выражением, состоящим из ссылок на переменные и операторы;
* ссылкой на переменную в скобках;
* вызовом функции;
* ссылкой на переменную, тип данных которой не совпадает формальным параметрам.

В последнем случае PL/I преобразует фактический параметр к типу данных, точности и масштабному коэффициенту фиктивного параметра. Следующая программа иллюстрирует это:

**┌── A:**

**│ procedure;**

**│ declare**

**│ X character(7),**

**│ (Y,Z) fixed binary;**

**│**

**│ call p(X,(Y),Z);**

**│**

**│ ┌── p:**

**│ │ procedure(A,B,C);**

**│ │ declare**

**│ │ A character(7),**

**│ │ B fixed binary,**

**│ │ C float binary;**

**│ │**

**│ │ A = 'Digital';**

**│ │ B = 100;**

**│ │ C = 2.5E2;**

**│ │**

**│ └── end p;**

**│**

**└── end A;**

Оператор CALL отправляет процедуре три фактических параметра X, Y и Z соответствующие трем формальным параметрам A, B и C. PL/I передает первый параметр ссылкой, потому что он соответствует формальному параметру и второй фактический параметр значением, потому что он возникает как выражение. PL/I преобразует третий фактический параметр к типу данных FLOAT BINARY и передает его значением.

## Оператор PROCEDURE

В PL/I можно определить процедуру с помощью оператора PROCEDURE в любом месте программы. Однако, для удобочитаемости желательно поместить все процедуры в одной секции вместе в начале или в конце главной программы. Главная программа представляет собой определение одной процедуры.

Оператор PROCEDURE обозначает точку входа в процедуру, определяет начало блока процедуры, описывает список формальных параметров и задает атрибуты возвращаемого значения функции. Процедура может состоять из одного или более операторов, включая соответствующий оператор END, который заканчивает определение процедуры. Оператор END может также быть точкой выхода из процедуры, несмотря на то, что внутри процедуры могут присутствовать операторы RETURN.

Общий вид оператора PROCEDURE:

**имя\_процедуры: PROCEDURE[(список\_параметров)]**

**[OPTIONS(орция,...)] [RETURNS(список\_атрибутов)]**

**[RECURSIVE];**

где список\_параметров - список формальных параметров процедуры, который необходимо объявить в теле процедуры на уровне основного блока.

Параметр может быть любым из следующих:

* скалярной переменной;
* массивом;
* главной структурой.

но не может иметь атрибутов:

* STATIC
* AUTOMATIC
* BASED
* EXTERNAL

OPTIONS(орция,...) определяет список из одной или нескольких опций: MAIN, STACK(b) или EXTERNAL.

Опция MAIN определяет процедуру в качестве первой процедуры получающей управление, когда программа начинает выполнение.

Опция STACK(b) устанавливает размер стека времени выполнения равным числу байтов, определенных b. Значение по умолчанию составляет 512 байт.

Опция EXTERNAL определяет процедуру как внешне скомпилированную процедуру. Опция EXTERNAL в заголовке процедуры делает процедуру доступной вне модуля. Часто полезно сгруппировать отдельно компилируемые процедуры в один файл, где процедуры ссылаются на одинаковые глобальные данные. В соответствии со стандартом подмножества G, можно скомпилировать каждую подпрограмму отдельно, повторяя описание области глобальных данных в каждом файле. Затем можно объединить отдельные модули с помощью редактора связей для получения единого загрузочного модуля.

**Примечание**: для совместимости с будущими реализациями PL/I Digital Research, вы должны отметить только процедуры верхнего уровня как OPTIONS(EXTERNAL), и объявить все глобально доступные данные как STATIC. Файл, содержащий группу процедур EXTERNAL может состоять из подпрограмм, исключая главную программу.

Следующий пример показывает использование опции EXTERNAL:

**┌── module:**

**│ procedure;**

**│ declare**

**│ 1 global\_data static,**

**│ 2 a\_field character(20) varying initial(''),**

**│ 2 b\_field fixed initial(0),**

**│ 2 c\_field float initial(0);**

**│ ┌── set\_a:**

**│ │ procedure (c) options(external);**

**│ │ declare c character(20) varying;**

**│ │ a\_field = c;**

**│ └── end set\_a;**

**│ ┌── set\_b:**

**│ │ procedure (x) options(external);**

**│ │ declare x fixed;**

**│ │ b\_field = x;**

**│ └── end set\_b;**

**│ ┌── set\_c:**

**│ │ procedure (y) options(external);**

**│ │ declare y float;**

**│ │ c\_field = y;**

**│ └── end set\_c;**

**│ ┌── sum:**

**│ │ procedure returns(float) options(external);**

**│ │ return (b\_field + c\_field);**

**│ └── end sum;**

**│ ┌── display:**

**│ │ procedure options(external);**

**│ │ put skip list(a\_field,b\_field,c\_field);**

**│ └── end display;**

**└── end module;**

Этот код определяет пять внешних процедур: set\_a, set\_b, set\_c, sum и display. Процедура, которая их использует, может быть, следующей:

**┌── call\_ext:**

**│ procedure options(main);**

**│ declare**

**│ set\_a entry (character(20) varying),**

**│ set\_b entry (fixed),**

**│ set\_c entry (float),**

**│ sum returns(float),**

**│ display entry;**

**│ call set\_a('Johnson,J');**

**│ call set\_b(25);**

**│ call set\_c(5.50);**

**│ put skip list(sum());**

**│ call display();**

**└── end call\_ext;**

Раздельная компиляция этих двух модулей, и связывание их вместе образуют единую исполняемую программу.

Атрибут RETURNS для процедуры функции определяет атрибуты значения, возвращаемого функцией.

Атрибут RECURSIVE указывает, что процедура может вызывать себя, либо прямо, либо косвенно.

## Организация программы на нижнем уровне

Организации исходного текста PL/I нижнего уровня включает спецификацию набора символов и правил формирования идентификаторов, ключевых слов и объявления имен, операторов, констант, разделителей и комментариев.

Язык PL/I имеет свободный формат. Исходная программа состоит из последовательности символов ASCII, которые составляют строки, разделенные символами возврата каретки. Исходный текст программы можно вводить не принимая во внимание позицию столбца или определенный формат строки. Однако, исходный текст легче читать и понимать, если следовать некоторым основным правилам форматирования:

* размещать один оператор в строке;
* использовать отступы для обозначения уровня вложенности блоков и группы DO.

Исходный текст может быть создан с помощью любого текстового редактора.

**Примечание**: все исходные программы на языке PL/I должны иметь тип файла PLI.

## Набор символов

Набор символов PL/I состоит из прописных и строчных букв, цифр и других символов. *Таблица 2-1* показывает символы, распознаваемые PL/I, и кратко описывает их использование.

Таблица 2‑1 Символы PL/I

|  |  |
| --- | --- |
| **Символ** | **Значение** |
| **=** | знак равенства (присвоение) |
| **+** | знак "плюс" (сложение) |
| **-** | знак "минус" (вычитание) |
| **\*** | звездочка (умножение) |
| **/** | наклонная черта (деление) |
| **(** | левая круглая скобка (ограничитель) |
| **)** | правая круглая скобка (ограничитель) |
| **,** | запятая (разделитель) |
| **.** | точка, используется в составных именах |
| **%** | символ процента (префикс INCLUDE или REPLACE) |
| **'** | апостроф, ограничитель строк |
| **;** | точка с запятой (ограничитель оператора) |
| **:** | двоеточие (разделитель для констант ENTRY или LABEL) |
| **^** | диакритический знак (логическое отрицание символа) |
| **~** | тильда (альтернатива логического отрицания) |
| **&** | амперсанд (логическое И символа) |
| **|** | вертикальная черта (логическое ИЛИ символа) |
| **!** | восклицательный знак (альтернатива логического ИЛИ) |
| **\** | наклонная черта влево (альтернатива логического ИЛИ) |
| **>** | правая угловая скобка (больше, чем) |
| **<** | левая угловая скобка (меньше, чем) |
| **\_** | подчеркивание (для удобочитаемости в идентификаторах) |
| $ | знак доллара (допустимый символ в идентификаторах) |
| ? | вопросительный знак (допустимый символ в идентификаторах) |

## Идентификаторы

Идентификатор это строка, содержащая от одного до тридцати одного символа, которые являются буквами, цифрами или знаком подчеркивания. Первый символ должен быть буквой. PL/I внутренне всегда преобразует буквы в верхний регистр. Поэтому, два идентификатора, которые отличаются только регистром, представляют собой один и тот же идентификатор.

В идентификаторах PL/I позволяет использовать символ вопросительного знака для доступа к внешним системным точкам входа (процедурам).

**Примечание**: Вы должны избегать использования вопросительных знаков для обеспечения совместимости снизу вверх с полной реализацией языка.

Каждый идентификатор в исходном тексте программы PL/I должен быть ключевым словом или объявленным именем. Ключевыми словами являются идентификаторы, имеющие особое предназначение в PL/I, при использовании в определенном контексте. Примерами ключевых слов являются имена встроенных функций, операторов и атрибутов данных. Ниже представлен список всех ключевых слов. Краткое описание команд языка PL/I содержит полный список ключевых слов с краткими пояснениями.

A ABS ACOS ADDR ALIGNED

ALLOCATE ASCII ASIN ATAN ATAND

AUTO AUTOMATIC B B1 B2

B3 B4 BASED BEGIN BIN

BINARY BIT BOOL BUILTIN BY

CALL CEIL CHAR CHARACTER CLOSE

COLLATE COLUMN COS COSD DCL

DEC DECIMAL DECLARE DIM DIMENSION

DIRECT DIVIDE DO E EDIT

ELSE END ENDFILE ENDPAGE ENTRY

ENV ENVIRONMENT ERROR EXP EXT

EXTERNAL F FILE FIXED FIXEDOVERFLOW

FLOAT FLOOR FOFL FORMAT FREE

FROM GET GO TO GOTO HBOUND

IF INCLUDE INDEX INIT INITIAL

INTO KEY KEYED KEYFROM KEYTO

LABEL LBOUND LENGTH LINE LINENO

LINESIZE LIST LOG LOG2 LOG10

MAIN MAX MIN MOD NULL

OFL ON ONCODE ONFILE ONKEY

OPEN OPTIONS OUTPUT OVERFLOW PAGENO

PAGESIZE POINTER PRINT PROC PROCEDURE

PTR PUT R RANK READ

RECORD RECURSIVE REPEAT REPLACE RETURN

RETURNS REVERT ROUND SEQUENTIAL SET

SIGN SIGNAL SIN SIND SINH

SKIP SQRT STACK STATIC STOP

STREAM SUBSTR SYSIN SYSPRINT TAB

TAN TAND TANH THEN TITLE

TO TRANSLATE TRUNC UNDEFINEDFILE UNDF

UNDERFLOW UFL UNSPEC UPDATE VAR

VARIABLE VARYING VERIFY WHILE WRITE

X ZERODIVIDE

Объявленные имена являются идентификаторами, использование которых или смысл вы определяете в операторе DECLARE ([*Раздел 3.6*](#_Оператор_DECLARE)). Ключевое слово может появиться в объявлении как определяемый пользователем идентификатор. Значение идентификатора зависит от места его появления. PL/I определяет значение по контексту. Например, INDEX является ключевым словом потому, что это имя встроенной функции PL/I. Однако в контексте объявления

**declare index fixed binary;**

index является объявленным именем, а не ключевым словом.

## Константы

Константы представляют собой элементы, записанные в текстовом виде, значение которых не может изменяться при выполнении программы. В PL/I основные константы следующие:

* арифметические (например: 3674.799);
* символьные строки (например: 'Ada Lovelace');
* битовые строки (например: '0010110'B).

## Ограничители и разделители

Отдельные элементы, такие как идентификаторы, должны быть различимы. PL/I распознает некоторые символы в качестве ограничителей и разделителей.

Обычно, *ограничители* (delimiters) окружают один или несколько текстовых элементов, в то время как *разделители* (separators) отмечают конец одного элемента и начало другого. В PL/I каждому идентификатору и арифметической константе должен предшествовать и завершать их один или несколько ограничителей или разделителей. Ограничителями могут быть пустое пространство (space), операторы или специальные символы.

### Пустое пространство

В PL/I *пустое пространство* может быть пробелом или символом табуляции (Ctrl-I), PL/I игнорирует любые символы возврата каретки, перевода строки или последовательности символов возврата каретки и перевода строки, которые включены в строковую константу. Например, оператор присваивания

**string = 'WHEN YOU HAVE A VERY LONG STRING LIKE THIS PL/I ALLOWS**

**YOU TO PUT SOME OF IT ON ANOTHER LINE';**

присваивает указанную символьную строку переменной string. Любые пробелы или символы табуляции, которые следуют после слова ALLOWS или предшествуют тексту 'YOU TO PUT' включаются в строку.

### Операции

Операция является символом математической или логической операции. В языке PL/I существует четыре типа операций, приведенные в *Таблице 2-2*.

**Примечание**: операции, состоящие из двух символов, такие как >=, называются составными операциями и не должны разделяться пробелами или символами табуляции.

Таблица 2‑2 Знаки операций PL/I

|  |  |
| --- | --- |
| **Символ** | **Значение** |
|  | **Арифметические операции** |
| **+** | сложение или префикс плюс |
| **-** | вычитание или префикс минус |
| **\*** | умножение |
| **/** | деление |
| **\*\*** | возведение в степень |
|  | **Операции сравнения** |
| **>** | больше |
| **^>** или **~>** | не больше |
| **>=** | больше или равно |
| **=** | равно |
| **^=** или **~=** | не равно |
| **<=** | меньше или равно |
| **<** | меньше |
| **^<** или **~<** | не меньше |
|  | **Операции битовых строк** |
| **^** или **~** | логическое НЕ |
| **&** | логическое И |
| **|** или**!** или **\** | логическое ИЛИ |
|  | **Операция сцепления строк** |
| **||** или**!!** или **\\** | обозначают конкатенацию (сцепление строк) |

### Специальные символы

*Таблица 2-3* показывает специальные символы, которые в PL/I могут также выполнять функции ограничителей или разделителей. Последующие разделы руководства содержат примеры их использования.

Таблица 2‑3 Специальные символы ограничителей и разделителей

|  |  |
| --- | --- |
| **Символ** | **Функция** |
| **:** | Двоеточие следует за константами ENTRY и LABEL. |
| **;** | Точка с запятой завершает операторы. |
| **,** | Запятая разделяет элементы списка. |
| **.** | Точка разделяет элементы составного имени. |
| **'** | Одиночный апостроф - разделитель для спецификации символьных констант и строки битов. |
| **->** | Стрелка - составной оператор, состоящий из знака "минус" и правой угловой скобки. Это разделитель в ссылке описателя указателя. |
| **=** | Знак равенства служит разделителем в операторе присваивания. |
| **(** | Левая круглая скобка. |
| **)** | Правая круглая скобка. Левая круглая скобка вместе с правой круглой скобкой используется, для обрамления списков и индексов, определения порядка вычисления выражений и разделения ключевых слов от операторов и имен опций. |

### Комментарии

Комментарии обеспечивают поясняющий текст в исходной программе PL/I. Компилятор игнорирует комментарии, поэтому вы свободно можете размещать их везде, где допустимы разделители. Комментарий начинается с составной пары символов /\* и заканчивается обратной составной парой \*/. Например,

**.**

**.**

**.**

**get list(name); /\* чтение имени \*/**

**.**

**.**

**.**

## Операторы препроцессора

PL/I позволяет модификацию исходного текста программы или включение внешних исходных файлов во время компиляции с помощью операторов препроцессора. Операторы препроцессора обозначаются предшествующим символом % перед ключевым словом:

**INCLUDE** или **REPLACE**

### Оператор %INCLUDE

Оператор %INCLUDE копирует исходный текст PL/I из внешнего файла во время компиляции. Оператор полезен для заполнения списка формата или описания структуры. Оператор %INCLUDE использует следующий формат:

**%INCLUDE 'имя\_файла';**

где имя\_файла обозначает имя копируемого файла в исходную программу. имя\_файла должно соответствовать стандартной спецификацией файла, **[d:]имя\_файла[.тип]**, и должно быть заключено в одинарные апострофы. Если спецификация диска отсутствует, PL/I предполагает диск, содержащий исходную программу. Когда компилятор встречает в исходном файле оператор %INCLUDE, то начинает читать файл, определенный в %INCLUDE. Когда компилятор достигает конца %INCLUDE файла, он возобновляет чтение исходного файла.

Следующий пример иллюстрирует использование оператора %INCLUDE:

**f:**

**procedure;**

**declare a fixed binary;**

**%include 'struc.lib';**

**declare c float;**

**.**

**.**

**.**

**end f;**

Компилятор включает исходный текст из файла struc.lib в месте расположения оператора %INCLUDE.

**Примечание**: PL/I не поддерживает вложенные операторы %INCLUDE.

### Оператор %REPLACE

Оператор %REPLACE позволяет программировать с помощью именованных констант. Оператор имеет вид:

**%REPLACE идентификатор BY константа;**

Компилятор заменяет каждое вхождение заданного идентификатора в исходном тексте на указанную константу. Константа может быть арифметической константой без знака или со знаком, битовой строкой или строкой символов. Можно написать несколько операторов %REPLACE как один оператор %REPLACE с элементами, разделенными запятыми.

Например, оператор

**%replace true by '1'b;**

заменяет все вхождения true константой строки битов '1'b, так что компилятор интерпретирует оператор

**do while(true);**

как

**do while('l'b);**

PL/I требует, чтобы все операторы %REPLACE располагались на уровне внешних блоков до любых вложенных внутренних блоков.

**Примечание**: для упрощения сопровождения и отладки программы, желательно размещать все операторы %REPLACE сразу после заголовка процедуры.

**Конец раздела 2**

# Типы и атрибуты данных

Элементами данных в программе PL/I являются константы или переменные. *Константа* представляет собой элемент данных, значение которого не может измениться при выполнении программы, в то время как значение *переменной* может измениться.

Каждый элемент данных ассоциируется с набором свойств, называемых *атрибутами*, которые определяют такие свойства, как объем необходимой памяти, операции, которые можно проводить с данными и значения диапазона индексов. Оператор DECLARE *явно* присваивает атрибуты переменным, в других случаях, например, для констант, атрибуты *неявно* присваиваются по умолчанию (см. [*Раздел 3.6*](#_Оператор_DECLARE_1)).

*Переменные* могут представлять собой одиночные элементы данных. Один элемент данных, переменная или константа, называется *скаляром*. Переменные могут также представлять несколько элементов данных, называемых *агрегатами*.. ([*Раздел 5*](#_Агрегаты_данных) описывает агрегаты.)

PL/I поддерживает 6 типов данных:

* Арифметические;
* Строковые;
* Метки;
* Процедуры и функции (entry);
* Указатели;
* Файлы.

Следующие разделы посвящены подробному описанию каждого типа данных.

## Арифмитические данные

PL/I поддерживает три типа арифметических данных:

* FIXED BINARY для представления целочисленных значений;
* FLOAT BINARY для представления чисел с плавающей двоичной точкой, которые могут изменяться от очень маленьких до очень больших;
* FIXED DECIMAL для представления десятичных чисел, которые имеют фиксированное количество общих цифр и фиксированное количество цифр справа от десятичной запятой.

Каждый элемент арифметических данных имеет связанное значение точности, выраженное в виде целочисленной константы p, заключенной в круглые скобки. Точность p определяет общее количество десятичных или двоичных цифр, которые может содержать элемент.

Для FIXED DECIMAL за числом p необязательно может следовать запятая, и целочисленная константа q, называемая *масштабным коэффициентом*. Масштабный коэффициент q определяет число цифр справа от десятичной точки.

Если в операторе DECLARE точность переменной не объявлена явно, PL/I неявно определяет ее согласно правилам по умолчанию. Масштабный коэффициент по умолчанию равен 0, означая отсутствие дробных цифр.

### Тип данных FIXED BINARY

Данные FIXED BINARY представляют собой целые числа. Переменная, объявленная как FIXED BINARY[(p)], является целым числом, которое имеет p двоичных цифр. Максимальный диапазон p

**1** ≤ **p** ≤ **15**

PL/I внутренне представляет этот тип данных в виде двоичного дополнительного кода. Поэтому, числа FIXED BINARY(15) могут быть диапазоне от -32768 до +32767.

Объем памяти выделяемый PL/I числам FIXED BINARY, зависит от объявленной точности:

Если **p ≤ 7**, PL/I выделяет один байт.

Если **7 < p ≤ 15**, PL/I выделяет два байта.

Точность FIXED BINARY по умолчанию равняется 15. Объявление переменной FIXED, или BINARY, или FIXED BINARY эквивалентно объявлению FIXED BINARY(15).

PL/I обрабатывает десятичные целые числа в исходной программе как данные FIXED BINARY, только если они появляются в контексте, который требуют значения FIXED BINARY, например, индексы или арифметические операции, включающие другие данные FIXED BINARY. В противном случае, по умолчанию константы являются FIXED DECIMAL. В PL/I преобразование из других типов данных обычно происходит с усечением (См. [*Раздел 4*](#_Преобразования_данных) о правилах преобразования). Например, следующий код присваивает переменной A значение 1.

**declare A fixed binary;**

**A = 1.99;**

### Тип данных FLOAT BINARY

Данные FLOAT BINARY полезны в научных приложениях для представления очень больших или очень маленьких чисел. Переменная, объявленная как FLOAT BINARY(p), состоит их трех частей: знака s, двоичных цифр p, которые являются дробью или мантиссой и представляют значащие разряды числа, и целого числа показателя степени e, которое представляет масштабный коэффициент. Например, число FLOAT BINARY 3.56E3 имеет следующие части:

**┌──** знак

**│ ┌────** мантисса

**│ │ ┌──** показатель степени **E**

**│ │ │**

V V V

**+ 3.56 3**

PL/I поддерживает числа FLOAT BINARY одинарной и двойной точности. *Таблица 3-1* показывает допустимую точность и примерный диапазон значений для каждого типа.

Таблица 3‑1 Числа PL/I FLOAT BINARY

|  |  |  |
| --- | --- | --- |
| **Тип** | **Точность p** | **Диапазон r** |
| **Одинарная** | 1 ≤ p ≤ 24 | 5.88×10-38  ≤ |x| ≤ 3.40×1038  (не IEEE)  1.18×10-38  ≤ |x| ≤ 3.40×1038  (IEEE) |
| **Двойная** | 25 ≤ p ≤ 53 | 9.46×l0-308 ≤ |x| ≤ 1.80×10308 (IEEE) |

Точность по умолчанию FLOAT BINARY равняется 24, поэтому объявление переменной FLOAT эквивалентно объявлению FLOAT BINARY(24).

Константа FLOAT BINARY представляет собой число, выраженное в экспоненциальном представлении как последовательность десятичных цифр с необязательной десятичной точкой, за которой следует буква E, а затем необязательное знаковое десятичное целое число экспоненты. Например, последовательность кода:

**A = 2.3E2;**

**B = -4.67E+5;**

**C = 1.98E-2;**

присваивает значение 230 переменной A, -467000 переменной B и 0.0198 переменной C.

Вы можете смешивать в выражении константы различных типов данных. PL/I автоматически преобразует их к общему типу данных прежде, чем вычислить выражение. Например, если в операторе присваивания p объявлена FLOAT BINARY

**declare P float binary;**

**.**

**.**

**.**

**p = p + 3.14159;**

PL/I преобразует константу FIXED DECIMAL 3.14159 в формат FLOAT BINARY прежде, чем выполнить сложение. (См. [*Раздел 4.1*](#_Арифметические_преобразования_2))

### Тип данных FIXED DECIMAL

Данные FIXED DECIMAL используются для вычислений, в которых необходимо сохранить точные десятичные значения, например, в коммерческих приложениях, связанных с долларами и центами. Данные FIXED DECIMAL с нулевым масштабным коэффициентом могут использоваться для представления целочисленных данных.

Переменная, объявленная FIXED DECIMAL[(p[,q])], является десятичным числом со знаком, в общей сложности с p десятичными цифрами, с q цифрами справа от десятичной точки. Максимальное количество цифр p для FIXED DECIMAL равняется 15, и масштабный коэффициент q должен быть неотрицательным и меньше или равным точности. Диапазон FIXED DECIMAL числа x

**-10\*\*(p-q) < |x| < 10\*\*(p-q)**

где:

**1** ≤ **p** ≤ **15 и 0** ≤ **q** ≤ **p**

Все десятичные константы с или без десятичной точки по умолчанию считаются FIXED DECIMAL. Единственным исключением является использование константы в контексте FIXED BINARY. Точность переменной FIXED DECIMAL по умолчанию равняется 7. По умолчанию масштабный коэффициент переменной FIXED DECIMAL равен 0. Форма константы FIXED DECIMAL по умолчанию неявно определяет точность и масштабный коэффициент. Например,

**3.25** по умолчанию **FIXED DECIMAL(3,2)**

**302**  по умолчанию **FIXED DECIMAL(3,0)**

Внутренне, PL/I представляет десятичные числа в упакованном дополнительном коде BCD. Число байтов, занимаемое числом FIXED DECIMAL, зависит от его объявленной точности. Если точность обозначена p, то число зарезервированных байтов является целой частью

**(p+2)/2**

т.е. минимум один и максимум восемь байтов.

PL/I усекает любое значение, масштабный коэффициент которого больше, чем у переменной FIXED DECIMAL, которой оно присваивается. Кроме того, PL/I устанавливает состояние FIXEDOVERFLOW, если значение, присваиваемое переменной, имеет больше значащих цифр слева от десятичной точки, чем позволяет объявленная точность переменной.

## Строковые данные

PL/I поддерживает два типа строковых данных:

* символьные строки;
* битовые строки.

*Символьная строка* являются последовательностью любых символов ASCII, включая пустую последовательность, которая является пустой строкой. *Битовые строки* является последовательность битов. *Длина строки* - число символов или битов в строке. Следующие разделы описывают каждый тип строковых данных.

### Данные типа символьная строка

Переменная, объявленная как CHARACTER(n), является символьной строкой длиной n, где n - значение между 1 и 254. Например, оператор

**declare A character(10);**

определяет переменную A как строку символов длиной в десять символов. Если символьная строка, присвоенная A, короче, чем A, PL/I дополняет строку пробелами справа до длины A. Если A присвоена более длинная строка, PL/I усекает строку справа.

*Константы символьных строк* представляют собой последовательность символов, заключенных в апострофы. Если апостроф является частью строки, он записывается как два апострофа подряд. Таким образом, строковая константа, значение которой

**What's Happening?**

записывается как:

**'What''s Happening?'**

Нулевая или *пустая символьная строка* имеет длину равную нулю и определяется при помощи двух последовательных апострофов.

Переменные символьных строк могут также иметь атрибут VARYING, указывающий, что переменная может представлять строки непостоянной длины с максимальной длиной n. Например, оператор

**declare A character(10) varying;**

определяет A для представления любого значения строки символов, длина которой может изменяться от 0 до 10.

PL/I допускает использование управляющих символов в строковых константах. Символ диакритического знака (^) в строковой константе обозначает управляющий символ. PL/I маскирует три старших бита символа, обнуляя их, таким образом, преобразуя в строке ^M или ^m в символ возврата каретки. Аналогичным образом, он преобразует строку ^I в символ горизонтальный табуляции. PL/I преобразует удвоенный символ диакритического знака (^^) в строке в один символ ^.

**Примечание**: Вы должны избегать использования управляющих символов, если требуется совместимость, так как соглашение об использовании Escape последовательностей отсутствует в других реализациях языка.

### Данные типа битовая строка

Строки битов представляют элементы *логических* данных. Битовая строка, содержащая все нулевые биты, является *ложью* (false). Битовая строка, содержащая любой из битов равный единице, является *истиной* (true).

Переменная, объявленная как BIT(n) является битовой строкой, содержащей n битов, где n - значение между 1 и 16.

Например, оператор

**declare A bit(3);**

определяет битовую строку длиной 3. Если битовая строка, присвоенная A, короче, чем A, PL/I дополняет строку нулевыми битами справа до длины A. Если A присваивается более длинная строка, PL/I усекает строку справа.

**Примечание**: переменные строки битов не могут иметь атрибут VARYING.

*Константу битовой строки* можно записать в любом из четырех различных форматов. Каждый формат соответствует основанию (степени), которая является числом битов, используемых для представления каждой цифры в константе. Константа битовой строки является последовательностью цифр и букв заключенных в апострофы, за которой следует буква B, и необязательная цифра, указывающая основание. По умолчанию основание равняется 2, обозначенное B или B1. *Таблица 3-2* показывает различные форматы.

Таблица 3‑2 Форматы константы битовой строки

|  |  |  |
| --- | --- | --- |
| **Формат** | **Основание** | **Цифры и/или буквы в представлении** |
| **B** | 2 | 0, l |
| **B1** | 2 | 0, l |
| **B2** | 4 | 0, 1, 2, 3 |
| **B3** | 8 | 0, 1, 2, 3, 4, 5, 6, 7 |
| **B4** | 16 | 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F |

**Примечание**: символы и/или цифры, используемые в последовательности, должны быть допустимыми для основы, определенной форматом.

Следующие примеры иллюстрируют эквивалентность дополнительных форматов формату с основанием 2:

**'101'B1** эквивалентно **'101'B**

**'1011B2** эквивалентно **'01000101'B**

**'101'B3** эквивалентно **'001000001'B**

'101'B4 эквивалентно **'000100000001'B**

**'9A'B4** эквивалентно **'10011010'B**

**'77'B3** эквивалентно **'111111'B**

## Данные управления программой

Элементы данных управления программой определяют последовательность выполнения работающей программы. PL/I поддерживает два типа управляющей информации:

* метки (LABEL);
* процедуры (ENTRY).

### Данные типа LABEL

Данные типа LABEL состоят из меток-констант и меток-переменных. *Метка-константа* является меткой, которая располагается перед исполняемым оператором. *Метка-переменная* является переменной, определенной в операторе DECLARE с помощью атрибута LABEL. Формат PL/I

**DECLARE name LABEL;**

по умолчанию присваивает атрибут VARIABLE. Метка-переменная может принимать значения различных меток-констант при выполнении программы.

Вы не можете явно объявить метку-константу в операторе DECLARE. Однако, любое имя, используемое в качестве метки оператора, представляет собой неявное объявление имени в качестве метки-константы. PL/I не допускает метки в следующих операторах:

* операторе DECLARE;
* в операторе, который начинается с ON-блока (см. [*Раздел 9.1*](#_Оператор_ON));
* в операторе, начинающемся с условия ELSE или THEN (см. [*Раздел 8.3*](#_Оператор_IF)).

Присвоение метке-переменной метки-константы или другой метки-переменной может быть выполнено по тем же правилам, что и присвоениях других типов переменных.

Единственными операторами, которые можно использовать с данными LABEL, являются операторы сравнения: равно (=) и не равно (^= или ~=).

Метки-константы и метки-переменные подчиняются тем же правилам области действия, что и объявленные имена. Элемент данных LABEL известен только в блоке, в котором он явно объявлен оператором DECLARE, или неявно, при использовании его в качестве метки-константы. Появление метки с тем же именем метки в любом другом блоке, включая вложенный блок, определяет новое объявление, локальное для этого блока.

Вы можете использовать метки-константы с индексом, используя в качестве индекса целую константу, возможно со знаком. Появление в блоке метки с индексом означает неявное описание массива меток-констант в этом блоке, причем описанный неявно массив определен только для индексов, которые встречаются в этом блоке. Вы можете также явно объявить массив меток-переменных в операторе DECLARE.

Следующая последовательность кода иллюстрирует массив меток-констант:

**case\_num(1):**

**case\_num(2):**

**case\_num(3):**

PL/I рассматривает case\_num, как если бы он был объявлен как массив в блоке, содержащем индексируемые метки. Поэтому, можно сосылаться на любой элемент в массиве, используя оператор GOTO (см. [*Раздел 8.5*](#_Оператор_нелокального_GOTO)), например:

**goto case\_num(i);**

где i - целое число. Значение переменной i определяет, какая метка должна получить управление.

**Примечание**: PL/I не рассматривает метки в операторах FORMAT (см. [*Раздел 11.3.5*](#_Оператор_FORMAT)) или PROCEDURE в качестве допустимых меток, а скорее как константы FORMAT и константы ENTRY, соответственно. Они не могут быть целями операторов GOTO.

Следующая последовательность кода иллюстрирует метки-переменные:

**declare do\_it label;**

**...**

**if (answer = 'yes') then**

**do\_it = geometric\_mean;**

**else**

**do\_it = arithmetic\_mean;**

**goto do\_it;**

**...**

В этом примере оператор GOTO передает управление к метке geometric\_mean или arithmetic\_mean, в зависимости от текущего значения метки-переменной do\_it, на основании результата проверки в операторе IF.

### Данные типа ENTRY

В PL/I все элементы данных ENTRY являются entry-константами или entry-переменными. Entry-константы соответствуют внутренним или отдельно скомпилированным внешним процедурам. Entry-переменные являются элементами данных, которым могут присваиваться значения entry-констант при выполнения программы.

Вызывающая программа должна использовать объявление ENTRY, чтобы определить характеристики параметров и возвращаемых значений для всех внешне скомпилированных процедур.

**Примечание**: необходимо убедиться, что объявление ENTRY соответствует внешне определенной процедуре, чтобы редактор связей смог правильно объединить сегменты программы.

Переменные, которым присваиваются значения entry-констант, также определяются с помощью объявления ENTRY. При необходимости, entry-переменные могут быть индексированы, в то время как entry-константы не могут. Данные ENTRY, как и данные LABEL могут сравниваться только на равенство или неравенство.

Объявление внешней entry-константы имеет вид:

**DECLARE имя\_входа [EXTERNAL]**

**[ENTRY[(список\_параметров)]**

**[RETURNS(атрибуты\_возвращаемого\_значения)];**

Объявление entry-переменной имеет вид:

**DECLARE имя\_входа [(пара\_индексов\_1,...,пара\_индексов\_n)]**

**[ENTRY[(список\_параметров)] VARIABLE**

**[RETURNS(атрибуты\_возвращаемого\_значения)];**

где атрибуты могут быть в любом порядке, но должны определить ENTRY или RETURNS.

Идентификаторы трактуются следующим образом:

* имя\_входа определяет имя ENTRY переменной или константы.
* пара\_индексов\_1,...,пара\_индексов\_n определяет необязательную размерность массива entry-переменной
* список\_параметров задает список описаний формальных параметров
* атрибуты\_возвращаемого\_значения - атрибуты возвращаемого значения процедуры-функции

Атрибут EXTERNAL указывает, что имя\_входа является отдельно скомпилированной процедурой. Атрибут VARIABLE указывает, что имя\_входа является entry-переменной, которой должно присваиваться значение entry-константы, при выполнении программы. Атрибут RETURNS подразумевает, что имя\_входа функция, а не подпрограмма.

Если процедура не имеет параметров, список\_параметров можно опустить. В этом случае можно также опустить атрибут ENTRY, если определен атрибут RETURNS. При определении списка индексных пар, должен использоваться атрибут VARIABLE. Если не определены EXTERNAL или VARIABLE, PL/I по умолчанию подразумевает EXTERNAL.

Если конкретный параметр имеет атрибут размерности, он должен присутствовать в качестве первого атрибута. Если параметр представляет собой структуру, информация о структуре, с номерами уровней, должна предшествовать определению атрибута. (См. [*Раздел 5.5*](#_Структуры))

Ниже приведены некоторые примеры объявлений ENTRY:

**declare X entry;**

**declare Y entry variable;**

**declare P (0:10) entry(fixed,float) variable;**

**declare Q entry(l, 2 fixed, 2 float,(5:10) decimal);**

**declare R returns(character(10));**

Следующая последовательность кода иллюстрирует элементы данных entry:

**declare**

**(X,Y) float binary,**

**A entry variable,**

**F(3) entry(float) returns(float) variable,**

**ZZ entry(float) returns(float);**

**Pl:**

**procedure;**

**X=5;**

**end Pl;**

**P2:**

**procedure;**

**X=25;**

**end P2;**

**Y=9;**

**if Y = 5 then**

**A = Pl;**

**else**

**A = P2;**

**call A;**

**F(2) = ZZ;**

**Y = F(2)(X);**

**put list(Y);**

## Данные типа POINTER

Элементы данных POINTER адресуют определенные области в памяти. Значение элемента данных POINTER является адресом переменной в программе. Объявление переменной POINTER имеет вид:

**DECLARE X POINTER;**

PL/I не выполняет преобразование между POINTER и другими типами данных, таким образом, оператор присваивания может только присвоить переменные указателя другим переменным указателя. Кроме того, переменные указателя не могут быть выведены в файл STREAM (см. [*Раздел 11*](#_Потоковый_ввод-вывод)). Для данных POINTER, как и для данных LABEL и ENTRY, допустимы только операторы сравнения на равенство и неравенство. Два указателя равны, если они представляют один и тот же блок памяти.

Вы можете использовать данные POINTER с базированными переменными, для динамического управления памятью. [*Раздел 7.2*](#_Оператор_ALLOCATE) описывает базированные переменные.

## Данные типа FILE

Элементы данных FILE состоят из файловых констант и файловых переменных, которые имеют доступ к внешним данным. Объявление файловых констант имеет общий вид:

**DECLARE file\_id FILE;**

Объявление файловых переменных имеет вид:

**DECLARE file\_id FILE VARIABLE;**

где file\_id - идентификатор PL/I, предназначенный для представления файла. Если file\_id не имеет параметров, PL/I автоматически обрабатывает идентификатор как EXTERNAL, чтобы он обращался к тому же набору данных во всех модулях, в которых он объявлен как EXTERNAL.

Если файл не открывается явно с помощью оператора OPEN содержащего опцию TITLE, PL/I получает доступ к дисковому файлу, с именем file\_id.DAT на диске по умолчанию.

[*Раздел 10*](#_Обработка_ввода_и) более подробно описывает данные FILE. Примеры использования данных FILE приводятся в *Руководстве программиста языка PL/I*.

## Оператор DECLARE

В PL/I необходимо использовать оператор DECLARE для объявления имен всех переменных в программе, которые не являются именами встроенных функций или псевдопеременных ([*Раздел 6.8*](#_Псевдопеременные)). Файловые константы и файловые переменные также должны быть объявлены в операторе DECLARE. Константы управления, такие, как операторы меток и имена процедур, объявляются неявно, использованием их в программе.

Оператор DECLARE связывает каждое имя переменной с надлежащими атрибутами для объявленного типа данных. Простая форма оператора DECLARE для скалярных переменных имеет вид

**DECLARE имя [список\_атрибутов];**

где имя - идентификатор переменной, и список\_атрибутов - одна или более характеристик имени переменной. Несколько атрибутов могут появиться в любом порядке, но должны быть разделены пробелами. Следующие примеры иллюстрируют операторы DECLARE:

**declare x fixed binary;**

**declare pi float binary(53);**

**declare overtime\_pay fixed decimal(5,2) initial(000.00);**

**declare EOF bit(l) initial('l'b);**

**declare list-head pointer static initial(null);**

## Составной оператор DECLARE

Для простоты и удобства, PL/I допускает несколько объявлений в одном операторе. Как правило, вы можете написать любую последовательность операторов DECLARE вида

**DECLARE описание\_1;**

**DECLARE описание\_2;**

**...**

**DECLARE описание\_n;**

в эквивалентной форме:

**DECLARE описание\_1, описание\_2, ... описание\_n;**

где каждый элемент описания разделен запятыми и нулем или несколькими символами пустого пространства, и оператор DECLARE завершен точкой с запятой.

Если несколько описаний элементов с одинаковыми атрибутами, вы можете вынести их вправо. То есть, вы можете записать последовательность определения в форме

**элемент\_1 атрибут\_A, элемент\_2 атрибут\_A, ... элемент\_n атрибут\_A**

в эквивалентной разложенной форме:

**(элемент\_1, элемент\_2, ... элемент\_n) атрибут\_A**

Например,

**declare (first\_name,last\_name) character(20) varying;**

Повторное применение этого правила также разрешено. Например, оператор:

**declare ((A,B) fixed binary, C float binary) static external;**

эквивалентен оператору:

**declare A fixed binary static external,**

**B fixed binary static external,**

**C float binary static external;**

## Атрибуты по умолчанию

Список атрибутов не может содержать противоречивые атрибуты, такие как два типа данных или два атрибута класса памяти. Если оператор DECLARE не определяет полный набор атрибутов, то компилятор использует атрибуты по умолчанию в соответствии со следующими правилами:

* Если атрибуты не определены, подразумевается FIXED BINARY(15).
* Если DECIMAL или BINARY определены без FIXED или FLOAT, подразумевается FIXED.
* Если FIXED или FLOAT определено без BINARY или DECIMAL, подразумевается BINARY.
* Если не указана точность для FIXED BINARY, подразумевается FIXED BINARY(15).
* Если не указаны точность и масштабный коэффициент для FIXED DECIMAL, подразумевается FIXED DECIMAL(7,0).
* Если не указана точность для FLOAT BINARY, предполагается FLOAT BINARY(24).
* Если не указана длина строки для BIT, предполагается BIT(1).
* Если не указана длина строки для CHARACTER, предполагается CHARACTER(1).

**Конец раздела 3**

# Преобразования данных

Преобразование данных представляет собой процесс, который изменяет представление заданного значения от одного типа к другому. В PL/I, все преобразования включает в себя источник, цель и результат. Источником является преобразуемый элемент данных, цель - тип, в который преобразуется исходный элемент и результатом является фактически преобразованное значение с типом данных целевого объекта.

PL/I выполняет преобразования в следующих общих категориях:

* арифметический в арифметический (тип и точность представления);
* арифметический в строковый;
* строковый в арифметический;
* формат указанный в редактируемом вводе-выводе (см. [*Раздел 11.3*](#_EDIT-directed_I/O)).

PL/I не выполняет преобразование значений ENTRY, FILE, LABEL или POINTER.

Часть гибкости и эффективность PL/I состоит в свободе объявления данных самых разных типов. С этой свободой приходит обязанность понимания, как язык преобразует данные из одного типа в другой, явно или неявно.

Следующий список показывает некоторые ситуации, в которых PL/I по умолчанию выполняет преобразование данных.

* В операторе присваивания PL/I преобразует тип выражения к типу переменной, которой это выражение присваивается.

**переменная = выражение;**

* В операторе RETURN PL/I преобразует указанное значение к типу, определенном в атрибуте RETURNS оператора PROCEDURE.

**proc\_name:**

**PROCEDURE RETURNS(return\_att);**

**...**

**RETURN (return\_exp);**

**END [proc\_name];**

* В любом арифметическом выражении, если операнды имеют разные типы, PL/I преобразует их к общему типу прежде, чем выполнить операцию. Например, если A - FLOAT BINARY, и B - FIXED BINARY, то в любой из следующих операций

**A + B**

**A – B**

**A \* B**

**A / B**

**A \*\* B**

общий тип будет FLOAT BINARY, и PL/I преобразует B в каждом случае.

* Во время обработки ввода-вывода PL/I преобразует данные в символьную строку и из символьной строки при использовании операторов PUT или GET соответственно. Например, если значение I - FIXED BINARY в операторе

**PUT LIST(I);**

PL/I преобразует I к CHARACTER. В операторе:

**GET LIST(I);**

PL/I преобразует символы во входном потоке от CHARACTER к FIXED BINARY.

* PL/I преобразует значения, определенные в некоторых операторах к целочисленным значениям. Например, в операторе цикла DO

**DO control\_variable = start\_exp TO end\_exp BY incr\_exp;**

**...**

**END;**

PL/I преобразует значения start\_exp, end\_exp и incr\_exp в целые числа (FIXED BINARY) прежде, чем выполнить оператор DO.

* PL/I имеет встроенные функции, выполняющие конкретные преобразования.

## Арифметические преобразования

PL/I выполняет арифметические преобразования в нескольких ситуациях. Первая ситуация - когда оператор присваивания присваивает арифметическое выражение арифметической переменной. PL/I преобразует в точность и масштабный коэффициент выражения к типу целевой переменной.

Другой случай - когда арифметическая функция возвращает арифметическое выражение в операторе RETURN. PL/I преобразует выражение к целевому типу данных с точностью и масштабным коэффициентом, определенным в атрибуте RETURNS объявления функции.

Когда у любой арифметической инфиксной операции, кроме возведения в степень, имеются операнды с различными типами данных, PL/I выполняет процесс в три этапа.

**Первый шаг**. PL/I определяет общий тип двух операндов.

* **Случай А**. Если один операнд FIXED BINARY и другой - FLOAT BINARY, общий тип - FLOAT BINARY.
* **Случай Б**. Если один операнд FIXED BINARY и другой операнд FIXED DECIMAL, общий тип - FIXED BINARY.
* **Случай В**. Если один операнд - FLOAT BINARY и другой операнд FIXED DECIMAL, общий тип - FLOAT BINARY.

*Таблица 4-1* приводит общий тип в выражениях, содержащих смешанные операнды.

Таблица 4‑1 Общие типы операндов в смешанных выражениях

|  |  |  |  |
| --- | --- | --- | --- |
| **Первый операнд** | **Второй операнд** | | |
| **FIXED BINARY** | **FLOAT BINARY** | **FIXED DECIMAL** |
| **FIXED BINARY** | FIXED BINARY | FLOAT BINARY | FIXED BINARY |
| **FLOAT BINARY** | FLOAT BINARY | FLOAT BINARY | FLOAT BINARY |
| **FIXED DECIMAL** | FIXED BINARY | FLOAT BINARY | FIXED DECIMAL |

**Второй шаг**. PL/I преобразует один из операндов к общему типу.

* **Случай A**. Если общий тип - FLOAT BINARY тогда
  + PL/I преобразует операнд FIXED BINARY(p) к FLOAT BINARY(p)
  + PL/I преобразует операнд FIXED DECIMAL(p,q) к FLOAT BINARY(p'), где p' = MIN(CEIL(p/3.322),53). MIN и CEIL являются встроенными функциями PL/I ([*Раздел 13*](#_Встроенные_функции)).
* **Случай Б**. Если общий тип - FIXED BINARY тогда
  + PL/I преобразует операнд FIXED DECIMAL(p,0) к FIXED BINARY(p'), где  
     p' = MIN(CEIL(p/3.322),15)

**Примечание**: PL/I не может преобразовать операнд FIXED DECIMAL(p,q) в FIXED BINARY, если q ^= 0.

**Третий шаг**.

После преобразование к общему типу PL/I определяет точность (и масштабный коэффициент) результата. Тип результата зависит от общего типа.

* **Случай А**. Если общий тип FIXED BINARY, результат FIXED BINARY. Если p1 - точность первого операнда, и p2 - точность второго операнда, PL/I получает точность результата p' в зависимости от операции.

Для сложения или вычитания

**p' = MIN(15,MAX(p1,p2)+1)**

Для умножения

**p' = MIN(15,p1+p2+1)**

Для деления вы должны использовать встроенную функцию DIVIDE с масштабным коэффициентом нуль, чтобы привести к результату FIXED BINARY, потому что PL/I подмножества G не поддерживает тип данных FIXED BINARY с ненулевым масштабным коэффициентом.

* **Случай Б**. Если общий тип - FLOAT BINARY, результат - FLOAT BINARY. Точность результата - MAX(p1,p2), где p1 и p2 - точность этих двух операндов.
* **Случай В**. Если общий тип FIXED DECIMAL, результат FIXED DECIMAL. Если первый операнд имеет точность и масштабный коэффициент (p1,q1), и второй операнд имеет точность и масштабный коэффициент (p2,q2), PL/I определяет точность и масштабный коэффициент результата (p', q') в зависимости от операции.

Для сложения или вычитания

**p' = MIN(15,MAX(p1-q1,p1-q2) + MAX(q1,q2)+1)**

**q' = MAX(q1,q2)**

Для умножения

**p' = MIN(15, p1+p2+1)**

**q' = (q1+q2)**

Для деления

**p' = 15**

**q' = 15-(p1+q1-q2)**

**Примечание**: соблюдайте осторожность при делении значений FIXED DECIMAL. Точность и масштабный коэффициент операндов должны быть такими, чтобы при операции деления не получился отрицательный масштабный коэффициент. Вы можете использовать встроенную функцию DIVIDE, чтобы управлять точностью частного.

Если используется оператор возведения в степень типа X\*\*Y, возможны два случая.

* **Случай А**. Y - десятичная целочисленная константа. Если X FIXED BINARY с точностью p и ((p+1)\*Y-1) ≤ 15, то результат FIXED BINARY с точностью

**p' = ((p+1)\*Y-1)**

Если X FIXED DECIMAL с точностью и масштабным коэффициентом (p,q) и ((p+1)\*Y-1) ≤ 15, то результат FIXED DECIMAL с точностью и масштабным коэффициентом (p',q'):

**p' = (p+1)\*Y-1**

**q' = q\*Y**

* **Случай Б**. Если любой операнд - FLOAT BINARY, PL/I преобразуем другой операнд к FLOAT BINARY и результат - FLOAT BINARY с точностью p' = MAX(p1, p2), где p1 и p2 - точность операндов.

В любой арифметической операции с участием преобразования PL/I усекает результат, если заявленная точность цели недостаточна для хранения значения. Усечение происходит справа для элементов данных FLOAT BINARY, и дробные цифры теряются в вычислениях FIXED DECIMAL. В вычислениях FIXED BINARY происходят непредсказуемые результаты, если абсолютное значение любого промежуточного значения превышает 32767.

Точность по умолчанию для значений с плавающей точкой - FLOAT BINARY(24). Однако, если вы объявляете литеральную константу больше чем с 7 значимыми десятичными цифрами, PL/I автоматически хранит ее как величину с двойной точностью. В выражениях, включающих операнды FLOAT BINARY различной точности, PL/I выполняет преобразование к большей точности. Например, если A - FLOAT BINARY(24), и B - FLOAT BINARY(53) в выражении:

A = A + B;

PL/I сначала преобразует A в двойную точность, выполняет сложение и затем преобразует результат обратно в одинарную точность.

**Примечание**: будьте внимательны при выполнении присвоений, включающих выражения смешанные точности. Наибольшее положительное число, представимое как значение одинарной точности, 3.40\*1038, тогда как наибольшее положительное число, представимое как значение двойной точности, 1.80\*l0308. Поэтому, если 3.40\*1038 < N ≤ 1.80\*10308, и вы назначаете N переменной одинарной точности, во время выполнения система оповестит состоянием арифметической ошибки:

**OVERFLOW(2)**

С другой стороны, наименьшее положительное число, представимое в качестве значения одинарной точности, 5.88\*10-39, тогда как наименьшее положительное число, представимое в качестве значения двойной точности, 9.46\*10-308, если 5.88\*10-39 < N ≤ 9.46\*10-308, и присвоить N переменной одинарной точности, во время выполнения система оповестит состоянием арифметической ошибки:

**UNDERFLOW(2)**

## Функции арифметических преобразований

PL/I предусматривает встроенные функции, для управления преобразованием из одного арифметического типа данных в другой. Ими являются

* BINARY
* DECIMAL
* DIVIDE
* FIXED
* FLOAT

Следующие разделы описывают эти функции.

### Встроенная функция BINARY

Встроенная функция BINARY имеет вид

**BINARY(X[,p]) | BIN(X[,p])**

где X арифметическая переменная или строковое выражение, которое будет преобразовано в арифметический тип данных BINARY, и p - намеченная точность.

При преобразовании арифметических переменных, если X FIXED BINARY или FIXED DECIMAL, результат FIXED BINARY. Если X FLOAT BINARY, результат - FLOAT BINARY.

Если p не указан, будут следующие результаты:

**X FLOAT BINARY(p)** возвращает **FLOAT BINARY(p)**

**X FIXED BINARY(p)** возвращает **FIXED BINARY(p)**

**X FIXED DECIMAL(p,q)** возвращает **FIXED BINARY(MIN(CEIL((p-q)\*3.322)+1,15))**

### Встроенная функция DECIMAL

Встроенная функция DECIMAL имеет вид

**DECIMAL(X[,p[,q]]) | DEC(X[,p[,q]])**

где X арифметическая переменная или выражение, которое будет преобразовано в арифметический тип данных FIXED DECIMAL, и p и q - точность и масштабный коэффициент результата. Ненулевой масштабный коэффициент допустим, только если X FIXED DECIMAL. Если p и q не определены, то результат следующий:

**X FIXED BINARY(p)** возвращает **FIXED DECIMAL(CEIL(p/3.322)+1,0)**

**X FLOAT BINARY(p)** возвращает **FIXED DECIMAL(MIN(CEIL(p/3.322),15),0)**

**X FIXED DECIMAL(p,q)** возвращает **FIXED DECIMAL(p,q)**

### Встроенная функция DIVIDE

Встроенная функция DIVIDE управляет точностью и масштабным коэффициентом результатов операций деления. Встроенная функция DIVIDE имеет вид:

**DIVIDE(X,Y,p[,q])**

где X и Y являются арифметические выражения, а X должен быть разделен на Y. p, является выражением FIXED BINARY, указывающим желаемую точность, и q - выражение FIXED BINARY, указывающее желаемый масштабный коэффициент. Если q не определен, он по умолчанию равен 0. Ненулевой масштабный коэффициент допустим, только если X и Y FIXED DECIMAL.

PL/I требуется функция DIVIDE для деления FIXED BINARY. В полном языке делении FIXED BINARY может создать ненулевой масштабный коэффициент, но PL/I подмножества G не поддерживает ненулевые масштабные коэффициенты для значений FIXED BINARY.

### Встроенная функция FIXED

Встроенная функция FIXED имеет вид:

**FIXED(X[,p[,q]])**

где X арифметическая переменная или выражение, которое будет преобразовано в арифметический тип данных FIXED, и p и q определяют точность и масштабный коэффициент результата.

Если X FIXED DECIMAL, результат FIXED DECIMAL. Иначе, результат FIXED BINARY. Если X FIXED BINARY, необходимо определить q равным 0. Ненулевой масштабный коэффициент допустим, только если X FIXED DECIMAL.

Если p или q не определены, то результат зависит от точности и масштабного коэффициента X следующим образом:

X FIXED BINARY(p) возвращает FIXED BINARY(p)

X FLOAT BINARY(p) возвращает FIXED BINARY(MIN(15,p)

X FIXED DECIMAL(p,q) возвращает FIXED DECIMAL(p,q)

### Встроенная функция FLOAT

Встроенная функция FLOAT имеет вид:

**FLOAT(X[,p])**

где X арифметическая переменная или выражение, которое будет преобразовано в арифметический тип данных FLOAT, и p - точность результата. Если p не определено, то результаты следующие:

X FIXED BINARY(p) возвращает FLOAT BINARY(p)

X FLOAT BINARY(p) возвращает FLOAT BINARY(p)

X FIXED DECIMAL(p,q) возвращает FLOAT BINARY(MIN(CEIL((p-q)\*3.322),53))

## Преобразования строк

PL/I выполняет преобразование между элементами арифметических и неарифметических данных представленных в виде строк, когда они объединены в выражениях. *Таблица 4-2* показывает встроенные функции, используемые для преобразования между арифметическими и неарифметическими типами данных.

Таблица 4‑2 Встроенные функции преобразования.

|  |  |
| --- | --- |
| **Преобразование** | **Встроенные функции PL/I** |
| Арифметического типа в битовую строку | BIT(S[,L]) |
| Арифметического типа в символьную строку | CHARACTER(S[,L]) |
| Битовой строка в арифметический тип | BINARY(X[,P]) |
| Битовой строка в символьную | CHARACTER(S[,L]) |
| Символьной строки в арифметический тип | BINARY(X[,P])  FLOAT(X[,p])  DECIMAL(X[,p[,q]]) |
| Символьной строки в битовую | BIT(S[,L]) |

Следующие разделы описывают эти встроенные функции PL/I и различные правила преобразования строковых операндов.

### Преобразование арифметического типа в битовую строку

Встроенная функция BIT имеет вид:

**BIT(S[,L])**

где S - арифметическое или строковое выражение, и L - положительное выражение FIXED BINARY.

PL/I сначала преобразует ABS(S) в FIXED BINARY в соответствии с правилами арифметических преобразований. Затем он преобразует промежуточное значение FIXED BINARY в битовую строку длиной L.

Если целевая длина больше L, PL/I дополняет промежуточный результат справа нулевыми битами. Если целевая длина - меньше, чем L, она обрезает справа лишние биты промежуточного результата.

### Преобразование арифметического типа в символьную строку

Встроенная функция CHARACTER имеет вид:

**CHARACTER | CHAR(S[,L])**

где S - арифметическое или строковое выражение и L - положительное выражение FIXED BINARY.

PL/I сначала преобразует различные арифметические типы данных, в промежуточные символьные строки следующим образом:

* **FIXED BINARY(p)**

PL/I преобразует источник в FIXED DECIMAL(p'), где p' = CEIL(p/3.322)+1, и затем преобразует результат FIXED DECIMAL(p') в символьную строку длиной p'+3 в формате, описанном выше.

Например, преобразование элемента данных FIXED BINARY(15) со значением -32 дает в результате символьную строку '⌴⌴⌴⌴⌴⌴-32'.

* **FLOAT BINARY(p)**

PL/I преобразует дробную часть в FIXED DECIMAL(p'), где p' = CEIL(p/3.322). Получающаяся символьная строка имеет длину p'+6 для одинарной точности или p'+7 для двойной точности в научном формате. Т.е. первый символ строки - знак "минус", если исходное значение отрицательное, в противном случае позиция содержит пробел.

Следующая позиция содержит цифру старшего разряда значения, за которой следуют десятичная точка и остальные p-1 цифры дробной части. Индикатор показателя степени E следует со знаком и значением порядка. Порядка одинарной точности содержит две цифры, а порядка двойной точности имеет три цифры.

Например, преобразование элемента данных FLOAT BINARY(24) со значением 250.1E1 приводит к символьной строке '⌴2.5010000E+03'.

* **DECIMAL(p,q)**, q = 0

Получающаяся символьная строка имеет длину p+3. Символы состоят из цифр источника, без начальных нулей, которым предшествует знак "минус", если исходное значение отрицательное, и дополненное слева пробелами, для создания символьной строки длиной p+3.

Например, преобразование элемента данных FIXED DECIMAL(3) со значением 330 приводит к символьной строке '⌴⌴330', где ⌴ обозначает пустую позицию. Преобразование значения ноль в результате производит пять пробелов и одну цифру ноль.

* **DECIMAL(p,q)**, q > 0

Получающаяся символьная строка имеет также длину p+3 с тем же форматом строки, как и выше, за исключением того, что включены десятичная точка и цифры дробной части.

Например, преобразование элемента данных FIXED DECIMAL(5,2) со значением 13.25 приводит к символьной строке '⌴⌴-13.25'. PL/I опускает начальные нули за исключением одного, непосредственно предшествующего десятичной точке.

После выполнения промежуточных преобразований PL/I дополняет строку пробелами справа, если целевая длина больше, чем длина промежуточного результата. И наоборот, если длина короче, чем промежуточный результат, PL/I усекает строку справа для создания строки с меньшей длиной.

### Преобразование битовой строки в арифметический тип

Встроенная функция BINARY описана в [*Разделе 4.2.1*](#_Встроенная_функция_BINARY). Когда она используется для преобразования битовой строки длиной n (0 ≤ n ≤ 15) в арифметический тип данных, PL/I сначала преобразует строку в его эквивалент FIXED BINARY(15). Затем PL/I преобразует промежуточное значение FIXED BINARY в целевое значение, в соответствии с правилами, которые обсуждались в [*Разделе 4.1*](#_Арифметические_преобразования).

Например, '1101'B при преобразовании в FIXED BINARY(15) дает значение 13.

### Преобразование битовой строки в символьную

Встроенная функция CHARACTER описана в [Разделе 4.3.2](#_Преобразование_арифметического_типа). Когда она используется для преобразования строки битов длиной n (0 ≤ n ≤ 15) в строку символов длиной n, PL/I преобразует бит имеющий значение ноль в символ 0 и бит со значением единицы в символ 1. Если целевая длина больше, чем длина источника, PL/I дополняет цель справа пробелами. Если целевая длина короче, чем исходная длина, PL/I усекает лишние символы справа.

### Преобразование символьной строки в арифметический тип

Функции преобразования осуществляются следующим образом:

* FIXED(X[,p[,q]]) или DECIMAL(X[,p[,q]]) возвращают значение FIXED DECIMAL. Если p не определено, по умолчанию оно равняется 15.
* BINARY(X[,p]) возвращает значение FIXED BINARY. Если p не определено, значение по умолчанию равняется 15. Результат содержит только целую часть X.
* FLOAT(X[,p]) возвращает значение FLOAT BINARY. Если p не определено, значение по умолчанию равняется 53. Если X имеет значение null или содержит все пробелы, преобразованное значение равно нулю. Если целевой объект не объявлен с достаточной точностью для содержания преобразованного значения, система времени выполнения устанавливает состояние OVERFLOW(2) или UNDERFLOW(2).

При выполнении преобразования символьной строки в арифметический тип символьная строка должна содержать допустимое значение арифметической константы. PL/I устанавливает состояние ERROR(1), если символьная строка имеет некорректное арифметическое представление.

Следующие примеры иллюстрируют различные преобразования из символьных в арифметические типы данных:

Таблица 4‑3 Преобразование символьной строки в арифметический тип

|  |  |  |
| --- | --- | --- |
| **Строка** | **Тип приемника** | **Результат** |
| '00987' | FIXED BINARY(15) | 987 |
| '9.87' | FIXED DECIMAL(6,2) | 0009.87 |
| '-9.87E2' | FLOAT BINARY(24) | -9.87E2 |
| '-9.87E2' | FIXED DECIMAL(9,2) | 0000987.00 |
| '-9.87E2' | FIXED DECIMAL(5.0) | 00987 |
| '-987.372' | FIXED DECIMAL(4,2) | ERROR |
| '2X3' | FIXED BINARY(15) | ERROR |
| '' | FIXED BINARY(15) | 0 |
| '⌴⌴⌴' | FIXED BINARY(15) | 0 |

### Преобразование символьной строки в битовую

Всторенная функция BIT описана в [*Разделе 4.3.1*](#_Преобразование_арифметического_типа_1). Когда она используется для преобразования символьных строк, PL/I преобразует каждый символ 0 в значение бита равное нулю, и каждый символ 1 в значение единицы. При выполнении преобразования символьной строки в битовую строку исходная символьная строка должна содержать только символы 0 и 1. Также она может содержать пробелы в начале или конце строки, но не внутри строки. PL/I устанавливает состояние ERROR(1), если символьная строка имеет некорректное представление.

Если целевая длина больше, чем исходная длина, то PL/I дополняет строку справа нулевыми битами. Если целевая длина короче, чем исходная длина, то усекает ее справа. Если источник - пустая строка или все позиции строки содержат пробелы, то результатом является строка из нулевых битов.

**Конец раздела 4**

# Агрегаты данных

Агрегат представляет собой группу из нескольких элементов данных. В PL/I существуют два вида агрегатов: массивы и структуры.

* *Массив* представляет собой упорядоченный набор элементов данных, называемых элементами, с одинаковыми атрибутами. Элементами массива могут быть скалярными элементами данных или структурами. PL/I позволяет ссылаться на весь массив по имени или ссылаться на отдельный элемент массива при помощи целочисленных индексов, которые обозначают относительную позицию элемента в массиве.
* *Структура* представляет собой совокупность элементов данных, называемых членами (members), которые могут иметь различные типы данных. Члены структуры могут быть массивами. PL/I позволяет ссылаться на всю структуру по имени или ссылаться на отдельный член структуры с помощью уточненной ссылки, которая содержит имя структуры и имя члена.

Переменную, которая представляет агрегат данных, называют или переменной массива или переменной структуры.

## Объявление массива

Вы определяете переменную массива, указывая ее атрибуты с точки зрения числа элементов в массиве и организации элементов. Эти атрибуты называют размерностями массива. Общая форма объявления переменной массива следующая:

**DECLARE имя(пара\_индексов,...) [список\_атрибутов];**

где имя - любой допустимый идентификатор PL/I. Каждая пара\_индексов определяет число элементов в каждой размерности массива и имеет формат:

**[Н:]В**

где Н - нижняя граница массива, и В - верхняя граница. Значения Н и В могут быть любыми целочисленными значениями при условии, что Н меньше или равно В.

список\_атрибутов - набор атрибутов данных, которые применяются ко всем элементам массива. Упорядочивание атрибутов неважно, но список пар индексов должен предшествовать списку атрибутов.

Число элементов для каждой размерности является размером и определяется как

**(верхняя граница) - (нижняя граница) + 1**

Общее количество элементов в массиве является произведение размеров каждой размерности.

Например, следующие операторы эквивалентны:

**declare A(3,4) character(2);**

**declare A(1:3,1:4) character(2);**

Оба оператора определяют массив, размерность (dimension) которого равняется двум, и элементы которого - строка из двух символов. Размер (extent) первой размерности равняется 3, и размер второй размерности равняется 4. Таким образом, можно представить A как массив из трех строк и четырех столбцов, элементы которого являются символьные строками из двух символов.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 |
| 1 | XX | XX | XX | XX |
| 2 | XX | XX | XX | XX |
| 3 | XX | XX | XX | XX |

Рисунок 5‑1 Двумерный массив

Оператор

**declare B(-2:5,-5:5,5:10) fixed binary;**

определяет, что массив B является трехмерным массивом, индексы которого изменяются в диапазоне от -2 до 5, -5 до 5, и от 5 до 10, соответственно. Соответствующие размеры восемь, одиннадцать и шесть, соответственно. Таким образом, массив B содержит 528 элементов данных типа FIXED BINARY.

При определении размерностей в массиве применяются следующие правила:

* В PL/I, отсутствуют формальные ограничения на количество размерностей массива. Тем не менее, реальный предел ограничивается объемом доступной свободной памяти для хранения данных и степенью сложности любого выражения, используемого для ссылки на отдельный элемент в массиве.
* Все индексы должны быть целочисленными константами.
* Нижняя граница должна быть меньше или равна верхней границе.
* Во времени выполнения ссылка за границы массива приводит к непредсказуемым результатам.

## Ссылки на массивы

В PL/I любая ссылка на отдельный элемент массива должна быть индексированной. Список индексов должен быть заключен в круглые скобки. В многомерных массивах число индексов должно совпадать с числом измерений.

Индексная ссылка на элемент массива может быть любой переменной или выражением, которое PL/I преобразует в целочисленное значение. Например,

**declare scores(20) fixed binary;**

**declare (counter, total) fixed binary;**

**total = 0;**

**do counter = 1 to 20;**

**total = total + scores(counter);**

**end;**

*Рисунок 5-2* иллюстрирует понятие индексных ссылок на элементы массива.

DECLARE ARRAY\_A(4) FIXED; /\* 4 СТОЛБЦА \*/

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |

↑ ↑

ARRAY\_A(1) ARRAY\_A(3)

Рисунок 5‑2 Ссылки на элементы массива

DECLARE ARRAY\_B(3,4) FIXED; /\* 3 СТРОКИ 4 СТОЛБЦА\*/

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 |  |
| 1 |  |  |  |  | ← ARRAY\_A(1,4) |
| 2 |  |  |  |  |
| 3 |  |  |  |  |
|  |  | ↑  ARRAY\_A(3,3) | | |

DECLARE ARRAY\_C(3,4) FIXED; /\* 2 ПЛОСКОСТИ 3 СТРОКИ 4 СТОЛБЦА\*/



Рисунок 5-2 (продолжение)

## Инициализация элементов массива

В объявлении массива для определения значений элементов перед выполнением можно использовать атрибут INITIAL. Например, оператор

**declare colors(4) character(10) varying**

**static initial ('RED','BLUE','GREEN','YELLOW');**

присваивает значение каждому элементу массива, как показано в *Рисунке 5-3*.

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 2 | 3 | 4 |
| RED | BLUE | GREEN | YELLOW |

Рисунок 5‑3 Инициализация массива

Если необходимо присвоить каждому элементу массива одинаковое значение, атрибут INITIAL может содержать коэффициент повторения в виде:

**INITIAL(значение[,значение] ...**

где значение имеет вид:

**[(коэффициент\_повторения)] константное\_выражение**

Коэффициент\_повторения является беззнаковой десятичной константой, указывающей сколько раз использовать указанную константу. Константное\_выражение может быть ссылкой на любую арифметическую или строковую константу или на встроенную функцию NULL и должно быть совместимым с инициализируемыми данными.

Например, оператор

**declare test\_scores(100) fixed binary**

**static initial((100)0);**

инициализирует все элементы массива test\_scores значением 0.

Оператор:

**declare grid(15) pointer;**

**static initial((15)null);**

инициализирует массив grid нулевыми значениями указателя.

Оператор:

**declare numbers(10) character(10)**

**static initial((10)'0123456789');**

инициализирует все десять элементов массива numbers константой из строки символов '0123456789' (см. *Приложение 1*).

Оператор:

**declare numbers(10) character(10)**

**static initial((5)'0123456789',(5)'0');**

инициализирует пять элементов массива numbers константой '0123456789' и пять элементов константой '0'.

PL/I сохраняет элементы внутри массива по строкам. Т.е., крайний правый индекс изменяется наиболее быстро. Если вы объявляете массив с атрибутом INITIAL, или ссылаетесь на весь массив в операторе GET или PUT, PL/I получает доступ к элементам в том же порядке.

Например, используя объявление массива:

**declare test\_scores(2,2,2) fixed static**

**initial (l,2,3,4,5,6,7,8);**

PL/I присваивает значения элементам в следующем порядке:

**test\_scores(1,1,1) = 1**

**test\_scores(1,1,2) = 2**

**test\_scores(1,2,1) = 3**

**test\_scores(1,2,2) = 4**

**test\_scores(2,1,1) = 5**

**test\_scores(2,1,2) = 6**

**test\_scores(2,2,1) = 7**

**test\_scores(2,2,2) = 8**

PL / I использует тот же порядок для вывода элементов в операторе PUT, как например:

**┌─ do i = 1 to 2;**

**│ ┌─ do j = 1 to 2;**

**│ │ ┌─ do k = 1 to 2;**

**│ │ │ put list(test\_scores(i,j,k));**

**│ │** └─ **end;**

**│** └─ **end;**

└─ **end;**

## Массивы в операторах присваивания

Только в некоторых ограниченных случаях PL/I допускает использование переменной массива, в качестве целевой переменной в операторе присваивания. Любой оператор вида:

**array\_variable\_A = array\_variable\_B;**

допустим, если массивы идентичны по размерности и типу данных, и оба массива имеют непрерывное хранение (connectedstorage) элементов. В этом случае каждому элементу array\_variable\_A присваивается соответствующий элемент array\_variable\_B. Например,

**declare A(20) fixed binary;**

**declare B(20) fixed binary;**

**A = B;**

Отдельные элементы массива могут также быть целями операторов присваивания. Например, в следующем фрагменте программы элементам одного массива присваиваются значения, вычисленные с использованием элементов в другого массива.

**declare array\_A(10) float binary;**

**declare array\_B(10) fixed binary static**

**initial (0,l,2,3,4,5,6,7,8,9);**

**declare i fixed binary;**

**do i = 1 to 10;**

**array\_A(i) = sqrt(array\_B(i));**

**end;**

Переменные массива не могут быть операндами для арифметических операций, таких, как + и –. Например, любой оператор вида:

**C= A + B;**

является недопустимым, если А, В и С являются переменными массива.

DRI PL/I **не позволяет** операторы типа:

* переменная\_массив = константа;
* переменная\_массив = выражение;

Например, следующая последовательность кода недопустима в PL/I:

**declare A(10) fixed binary;**

**declare n fixed binary static initial(2);**

**A = n;**

Однако, можно получить тот же эффект последовательностью:

**declare A(10) fixed binary;**

**declare n fixed binary static initial(2);**

**declare i fixed binary;**

**do i = i to 10;**

**A(i) = n;**

**end;**

## Структуры

Структура представляет собой агрегат, который может содержать элементы различных типов данных. Вы можете использовать структуры для представления данных, которые более точно отражают объекты реальной жизни.

Элементы данных, содержащиеся в структуре, называются ее членами (members). Структуры могут содержать скалярные элементы данных, массивы скалярных элементов, или другие структуры называемыми подструктурами. Структуры упорядочены иерархически. Главную структуру называют, *старшей структурой* и любую подструктуру называют *младшей структурой*.

Описание структуры определяет организацию уровней и имена элементов на каждом уровне в структуре. Каждое описание структуры должно содержать:

* имя главной структуры,
* имена и атрибуты данных ее членов,
* номер уровня каждого имени для определения его уровня в иерархии.

Объявления переменной структуры имеет вид:

**DECLARE | DCL [уровень] имя [список\_атрибутов] ...**

**[,[уровень] имя [список\_атрибутов]];**

Номера уровня предшествуют именам и должны быть разделены от них одними или более пробелами или символами табуляции. Номер уровня главной структуры всегда единица. Определения каждого элемента, включая его номер уровня, имя и атрибуты, должны быть разделены запятыми. Номера уровней членов младшей структуры должны быть больше номера предшествующего уровня структуры. Если несколько элементов структуры имеют одинаковый номер уровня - номер уровня можно вынести за скобки, расположив его слева. Следующая последовательность:

**уровень\_k элемент\_1, уровень\_k элемент\_2, ... уровень\_k элемент\_n**

эквивалентна последовательности:

**уровень\_k (элемент\_1, элемент\_2, ... эдемент\_n)**

Например, оператор:

**declare 1 A based,**

**2 B fixed binary,**

**C character(2));**

эквивалентен следующему:

**declare 1 A based,**

**2 B fixed binary,**

**2 C character(2);**

**Примечание**: имена структуры 1 уровня не могут иметь атрибутов типа данных, но могут иметь атрибут размерности. У имен структуры 1 уровня могут также быть атрибуты BASED, AUTOMATIC, EXTERNAL, PARAMETER или STATIC.

Следующий оператор является примером описания структуры:

**declare 1 bill,**

**2 name,**

**3 last\_name character(20),**

**3 firs\_name name character(20),**

**3 middle\_initial character(l),**

**2 address,**

**3 street character(20),**

**3 city character(10),**

**3 state character(3),**

**3 zip character(5),**

**2 charges,**

**3 shop fixed decimal(10,2),**

**3 snack\_bar fixed decimal(10,2),**

**3 misc fixed decimal(10,2),**

**3 dues fixed decimal(10,2);**

*Рисунок 5-4* показывает иерархию уровней, соответствующих этому объявлению.

BILL ──────┬────── NAMR ──────┬────── LAST\_NAME

│ ├────── FIRS\_NAME

│ └────── MIDDLE\_INITIAL

├────── ADDRESS ───┬────── STREET

│ ├────── CITY

│ ├────── STATE

│ └────── ZIP

└────── CHARGES ───┬────── SHOP

├────── SNACK\_BAR

├────── MISC

└────── DUES

Рисунок 5‑4 Иерархия уровней структуры

Номера уровней и имена могут быть учтены. При ссылках на члены структуры может возникнуть неоднозначность, так как имя члена структуры может появиться как имя члена другой структуры или имя элемента данных в подструктуре той же структуры. Эти неоднозначности возникают только с именами членов в общей области действия.

Для устранения таких неоднозначностей, используйте составные имена для ссылок на члены структур. В составном имени, имени члена предшествует список имен структуры в порядке возрастания номеров уровня, за каждым из которых следует точка и возможно ноль или более пробелов. Требуются только имена структур, которые однозначно определяют уникальную ссылку на имя члена. Например, в структуре:

**declare 1 A,**

**2 B,**

**3 C fixed,**

**3 D fixed,**

**2 BB,**

**3 C fixed,**

**3 D fixed;**

ссылка на элемент С, или D, или A.C, или A.D являются неоднозначными. Составные имена B.C, или B.D, или BB.C, или BB.D однозначно определяют элементы структуры.

Полностью определенные имена выглядят следующим образом:

**A.B.C, A.B.D, A.BB.C, A.BB.D**

*Рисунок 5-5* показывает иерархию уровней.

B ──────┬────── B ──────┬────── C FIXED

│ └────── D FIXED

└────── BB ─────┬────── C FIXED

└────── D FIXED

Рисунок 5‑5 Иерархия уровней структуры

## Смешанные агрегаты данных

Смешанный агрегат это либо массив, элементы которого включают структуры, или структура, членами которой являются массивы. Можно определить массив, элементы которого представляют собой структуру одного типа, задавая главному имени структуры атрибут размерности в объявлении структуры. Вы можете также задать младшим структурам атрибут размерности. При объявлении структуры с атрибутом размерности, каждый элемент структуры наследует размерность и становится массивом.

Например, оператор:

**declare 1 student\_list(100),**

**2 student\_name,**

**3 last\_name character(10),**

**3 first\_name character(10),**

**3 middle\_initial character(l),**

**2 social\_security\_number character(9),**

**2 country character(10),**

**2 grades(5) character(2);**

объявляет массив структур с главным именем структуры student\_list. *Рисунок 5-6* показывает массив этих структур с их элементами.

STUDENT\_LIST(100) ──────┬────── STUDENT\_NAME ──────┬────── LAST\_NAME

│ ├────── FIRS\_NAME

│ └────── MIDDLE\_INITIAL

│

├────── SOCIAL\_SECURITY\_NUMBER

├────── COUNTRY

└────── GRADES(5)

Рисунок 5‑6 Массив структур

Каждый элемент структуры массива содержит подмассив grades в качестве члена. Чтобы сослаться на запись в массиве, необходимо использовать полностью определенное имя вместе с индексами для имени структуры, которые имеют атрибут размерности и имя члена, если у него есть атрибут размерности. Индексы не должны появляться с соответствующим им именем, но должны быть в круглых скобках, разделенные запятыми и в правильном порядке.

Например, любая из следующих строк является полностью определенной, однозначной ссылкой на третью запись grade в шестьдесят первой записи массива student\_list:

**student\_list(61).grade(3)**

**student\_list.grade(61,3)**

**student\_list(61,3).grade**

## Ссылки на смешанные агрегаты данных

Можно сослаться на весь смешанный агрегат по имени. Ссылка на элементы данных в смешанном агрегате может быть частично индексирована, и/или определена неполным составным именем. Любая такая ссылка на смешанный агрегат должна идентифицировать непрерывное хранение (см. *Приложение 1*). *Непрерывное хранение* (connected storage) означает, что элементы данных занимают последовательные места хранения.

В качестве примера рассмотрим, как PL/I хранит элементы данных для двух объявлений:



Рисунок 5‑7 Массив структур и структура массивов

На *Рисунке 5-7а*, color является массивом и каждый из его членов, hue и intensity, наследует объявленную размерность. Таким образом, каждый выглядит как массив, но элементы не занимают последовательные места хранения.

На *Рисунке 5-7б*, color имеет два члена, оба из которых являются массивами. Элементы каждого массива занимают последовательные места хранения.

Как показано на *Рисунке 5-7а*, хранилище для color(3), или color.hue(100) непрерывное хранение, но хранение для color.hue не является непрерывным. Но, на *Рисунке 5-7б*, хранилище для color.hue непрерывное.

Каждый тип объявления имеет свои преимущества и недостатки. Конкретное приложение и метод доступа в программе определяют тип объявления и способ хранения.

**Конец раздела 5**

# Присваивания и выражения

## Оператор присваивания

Оператор присваивания задает значение переменной, равное значению выражения или константы. Оператор присваивания имеет вид:

**переменная = выражение;**

где переменная - скалярный элемент, массив, имя структуры или псевдо-переменная ([*Раздел 6.8*](#_Псевдопеременные_1)). Оператор присваивания не имеет явного ключевого слова.

PL/I не позволяет многократные присвоения в одном операторе. Например,

**A, B, C = 5;**

не допустим.

PL/I допускает операторы следующего вида:

**A = (B = C);**

В этом контексте PL/I обрабатывает знак равно (=) в круглых скобках как оператор отношения (см. [*Раздел 6.5*](#_Операторы_отношения)). Таким образом, если переменная B имеет то же значение, что и переменная C, отношение является истиной, и PL/I присваивает значение строки битов **'**1**'**b переменной A.

## Выражения

Выражение - любая допустимая комбинация операндов и знаков операций, которые PL/I вычисляет во время выполнения для получения значения.

Расположение ссылок, операторов и скобок в выражении определяют различные синтаксические правила. Ссылка может быть константой, переменной или функцией. Знак операции определяет выполняемые вычисления, используя операнды, к которым он применяется. Круглые скобки содержат различные части выражения.

В следующих разделах представлены соответствующие формулировки операндов, знаков операций и круглых скобок.

### Префиксные выражения

Префиксное выражение состоит из унарного префиксного оператора, за которым следует выражение, называемое операндом. PL/I сначала вычисляет операнд и затем применяет унарную операцию к результату.

Ниже приведены два примера префиксных выражений:

**^A /\* логическое отрицание A \*/**

**-SQRT(B) /\* минус квадратный корень B \*/**

### Инфиксные выражения

Инфиксное выражение состоит из двух выражений называемых операндами, разделенных инфиксным знаком операции. PL/I сначала вычисляет операнды, которые могут быть выражениями, и затем к результату применяет операцию.

В следующих двух примерах операции являются инфиксными.

**A+B /\* сумма A и B) \*/**

**C\*\*2 /\* C в квадрате \*/**

## Приоритет выполнения операций

PL/I вычисляет любое незаключенное в скобки выражение или подвыражение в соответствии с рядом правил приоритета операций. *Таблица 6-1* показывает установленную очередность выполнения операций от самого высокого до самого низкого приоритета. Операции с одинаковым приоритетом, перечислены в той же строке.

Таблица 6‑1 Приоритет операций PL/I

|  |  |  |
| --- | --- | --- |
| **Операция** | **Обозначение** | **Приоритет** |
| Возведение в степень | \*\* | 1 |
| Логическое НЕ (отрицание) | ^ или ~ | 1 |
| Префиксные операторы | +, - | 1 |
| Умножение, деление | \*, / | 2 |
| Сложение, вычитание | +, - | 3 |
| Конкатенация (сцепление) | || или !! или \\ | 4 |
| Операторы отношения | =, ^=, <, ^<, >, ^>, <=, >= | 5 |
| Логическое И (умножение) | & | 6 |
| Логическое ИЛИ (сложение) | | или ! или \ | 7 |

При вычислении незаключенного в скобки выражения PL/I сначала вставляет сбалансированную пару круглых скобок вокруг операций с наивысшим приоритетом и соответствующих им операндов. Затем он переходит к операциям с более низким приоритетом и их операндам, пока все выражение не будет должным образом заключено в скобки.

Когда встречаются операции с одинаковым приоритетом на том же уровне, PL/I вычисляет префиксные операции и возведение в степень справа налево. Остальные операции вычисляются слева направо.

Например, компилятор обрабатывает выражение в скобках:

2 + Z \* X \*\* Y\*\*2/5 -Q

как выражение

(2 + ((Z \* (X\*\*(Y\*\*2))) / 5))-Q

5 3 2 1 4 6

## Конкатенация (сцепление)

Инфиксная операция || связывает битовые или символьные строки. Оба операнда и результат должны иметь одинаковый тип. Длина получающейся строки всегда равна сумме длин операндов.

При соединении символьных строк, если любой операнд имеет атрибут VARYING, то и результат будет иметь атрибут VARYING. Например, последовательность кода:

**declare**

**A character(3),**

**B character(6) varying,**

**C character(20);**

**A = 'ABC';**

**B = 'ABCDEF';**

**C = A || B;**

присваивает переменной C символьную строку ABCABCDEF длиной 9.

## Операции сравнения

В PL/I операции сравнения это инфиксные операции, которые сравнивают отношение между двумя операндами в алгебраическом смысле. Вычисляемые значения можно сравнить по общим алгебраическим правилам, но невычисляемые значения можно сравнить только на равенство или неравенство.

Символьная строка, строка битов и арифметические элементы данных могут быть сравнены, используя любую операцию отношения. Данные ENTRY, FILE, LABEL и POINTER могут сравниваться только на равенство и неравенство операндов.

Значения ENTRY равны, только если они идентифицируют ту же точку входа в том же блоке.

Значения LABEL равны, только если они идентифицируют ту же переменную в том же блоке. Значение LABEL, которое идентифицирует метку в пустом операторе, не равно значению LABEL в любом другом операторе.

Значения POINTER равны, только если они идентифицируют то же место в памяти, или оба имеют значение null.

Значения FILE равны, только если они идентифицируют тот же блок параметров файла (см. [*Раздел 10.4*](#_Блок_параметров_файла)).

Если операнды отличаются по типу данных, PL/I сначала преобразовывает их к общему типу перед сравнением, и затем производит битовую строку длиной один со значением '1'B (истина), если операнды равны, и '0'B (ложь), если операнды не равны (см. [*Раздел 4*](#_Преобразования_данных_1)).

PL/I сравнивает символьные строки, расширив более короткий операнд справа с помощью пробелов, пока его длина не сравняется с более длинным операндом. Сравнение выполняется посимвольно слева направо, используя упорядоченную последовательность символов ASCII (см. *Приложение 3*). В этой последовательности значение любой прописной буквы - меньше любой строчной буквы, и значение любого цифрового символа - меньше, любого буквенного символа.

Например, имеются две строки JACK и JACKSON, PL/I сначала дополняет более короткую строку пробелами и затем сравнивает строки, начиная слева, как показано ниже, символ **⌴** обозначает пробел:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **J** | **A** | **C** | **K** | **⌴** | **⌴** | **⌴** |
| 4A | 41 | 43 | 4B | 20 | 20 | 20 |
|  |  |  |  |  |  |  |
| **J** | **A** | **C** | **K** | **S** | **O** | **N** |
| 4A | 41 | 43 | 4B | 53 | 4F | 4E |

Поскольку S (53h) больше, чем пробел (20h), JACKSON больше, чем JACK.

PL/I сравнивает строки битов, расширяя более короткую строку справа, с помощью нулевых битов. Затем выполняется поразрядное сравнение слева направо с нулем, который рассматривается меньше единицы. Например, '00010000'B - меньше, чем '00010001'B.

## Операции для работы с битовыми строками

*Таблица 6-2* показывает операции для работы с битовой строки PL/I.

Таблица 6‑2 Операции PL/I для работы с битовыми строками

|  |  |
| --- | --- |
| **Операция** | **Символ** |
| Отрицание (логическое НЕ) | **^** или **~** |
| Включающее ИЛИ (логическое ИЛИ) | **|** или **!** или **\** |
| Логическое И | **&** |

PL/I выполняет операции с битовой строкой на поразрядной основе. Унарная операция НЕ инвертирует значение каждого бита в операнде битовой строки, изменяя 0 на 1, и 1 на 0. Например, если задана строка битов A = '01110010'B, то ^A дает '10001101'B.

Операции логического ИЛИ и логического И требуют два операнда битовых строк. Если операнды имеют неравную длину, PL/I расширяет более короткий справа нулевыми битами, пока она не сравняется по длине с другим операндом. Результирующая строка имеет длину наибольшего операнда.

Операции логического ИЛИ и логического И следуют правилам булевой алгебры:

|  |  |  |  |
| --- | --- | --- | --- |
| **X** | **Y** | **X|Y** | **X&Y** |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 |

Дополнительные логические функции легко создаются с помощью встроенной функции BOOL (см. [*Раздел 13.3*](#_String-handling_Functions)).

## Возведение в степень

PL/I вычисляет возведение в степень как многократное умножение, если показатель степени - неотрицательная целочисленная константа. В остальных случаях, вычисление этой операции производится с использованием встроенных трансцендентных функций LOG и EXP. При обработке экспоненциального выражения PL/I выделяет следующие особые случаи:

* Если X=0 и Y>0, то X\*\*Y = 0.
* Если X=0 и Y<0, то система времени выполнения устанавливает состояние ERROR(3).
* Если X^=0 и Y=0, то X\*\*Y = 1.
* Если X<0 и Y не целое число, то система времени выполнения устанавливает состояние ERROR(3).

## Псевдопеременные

SUBSTR и UNSPEC - имена двух встроенных функций PL/I, которые можно использовать в качестве операнда источника в выражениях. Однако, SUBSTR и UNSPEC можно также использовать в качестве операнда приемника в левой части операторов присваивания. В этом случае SUBSTR и UNSPEC называют псевдопеременными, потому что они применяются как обычные переменные в программе.

### Встроенная символьная функция SUBSTR

Встроенная функция SUBSTR позволяет получить доступ к подстроке строки. Она принимает одну из следующих двух форм:

**SUBSTR(символьная\_переменная,i)**

**SUBSTR(символьная\_переменная,i,j)**

где символьная\_переменная - возможно индексированная ссылка на переменную CHARACTER или CHARACTER VARYING, и i и j выражения FIXED BINARY.

SUBSTR с двумя параметрами извлекает подстроку, начинающуюся в позиции i и продолжающуюся до конца строки. (Позиция 1 является первым символом строки, позиция 2 вторым, и т.д.) Результат не определен, если i превышает длину строки.

SUBSTR с тремя параметрами извлекает подстроку, начинающуюся в позиции i и длиной j символов. Результат не определен, если i или i+j превышает длину строки, где длина - объявленный фиксированный размер для переменных CHARACTER и текущая длина, для переменных CHARACTER VARYING.

Например, если переменная word содержит символьную строку 'Josephine', то в результате будут выполнены следующие присвоения.

**x = SUBSTR(word,l); /\* x = 'Josephine' \*/**

**y = SUBSTR(word,5); /\* y = 'phine' \*/**

**z = SUBSTR(word,1,4); /\* z = 'Jose' \*/**

Псевдопеременная SUBSTR похожа на встроенную функцию SUBSTR, за исключением того, что он появляется в левой части присваивания, и должна появляться только она. То есть, SUBSTR не может быть встроена в строковое выражение, когда она служит в качестве целевой строки назначения. SUBSTR появляется в этом контексте в одной из следующих форм:

**SUBSTR(символьная\_переменная,i) = символьное\_выражение;**

**SUBSTR(символьная\_переменная,i,j) = символьное\_выражение;**

Псевдопеременная SUBSTR с двумя параметрами присваивает символьной\_переменной символьное\_выражение, заданное подстрокой символьного выражения, начинающейся в позиции i и до конца символьной переменной.

Псевдопеременная SUBSTR с тремя параметрами присваивает символьной переменной символьное выражение, заданное подстрокой символьного выражения, начинающейся в позиции i и длиной j символов. Значения i и i+j должны быть текущей или фиксированной длиной строки, в противном случае могут возникнуть неопределенные результаты.

Та же символьная переменная может появиться в левой и правой части оператора присваивания без частичной перезаписи подстроки во время присвоения.

Например, если переменная word содержит символьную строку 'Collegiate', то после оператора

**substr(word,7) = substr(word,10,1);**

переменная содержит строку 'College**⌴⌴⌴**'

### Встроенная битовая функция SUBSTR

В PL/I операции с битовыми подстроками подобны символьным SUBSTR, показанным в [*Разделе 6.8.1*](#_Встроенная_символьная_функция) с некоторыми ограничениями. Во-первых, PL/I ограничивает точность строки битов в диапазоне 1-16, соответствующей одно- и двух- байтовым значениям. Для вычисления промежуточных значений точности во время компиляции, длина битовой подстроки должна быть константой.

Таким образом, формы для битовой подстроки следующие

**SUBSTR(битовая\_переменная,k)**

**SUBSTR(битовая\_переменная,i,k)**

где битовая\_переменная - возможно индексированная переменная типа BIT, k - целочисленная константа в диапазоне 1-16 и i - выражение FIXED BINARY.

Результат применения SUBSTR со строками битов идентичен описанным символьным операциям, за исключением того, что PL/I выбирает битовые строки длиной k, когда SUBSTR появляется в выражении и присваивает ее, когда SUBSTR появляется слева в качестве цель при операции сохранения строки битов.

Следующий раздел приводит пример битовой SUBSTR.

### Встроенная функция UNSPEC

Встроенная функция UNSPEC возвращает внутреннего представления значения аргумента строки битов. Встроенная функция UNSPEC имеет вид:

**переменная = UNSPEC(аргумент);**

где аргумент - возможно индексированная ссылка на элемент данных, который занимает одну - или две ячейки в памяти.

**Примечание**: PL/I не позволяет выражение в качестве аргумента UNSPEC.

Когда UNSPEC появляется как псевдопеременная в левой части оператора присваивания, PL/I преобразует присвоенное значение в битовую строку и непосредственно сохраняет ее в одно- или двухбайтовой переменной. Таким образом, UNSPEC позволяет получить доступ к одно- и двухбайтовым переменным, как будто они являются строками 8-разрядных и 16-разрядных элементов данных.

Псевдопеременная UNSPEC часто используется в безвыходных ситуациях, когда кажется, что обычные функции языка, не предоставляют доступ к основным объектам. Не используйте UNSPEC вместо более подходящих функций языка высокого уровня, поскольку UNSPEC зависит от конкретной реализации. Более того, каждый раз, когда кажется, что необходимо использовать UNSPEC, более тщательно исследуйте проблему, можно ли избежать использования UNSPEC.

Следующий пример показывает доступ к двум областям памяти. Операция UNSPEC загружает два абсолютных адреса в две переменные с типом указателя. В свою очередь, две базированные переменные совмещают эти две области памяти, что бы к ним можно было получить доступ как 16 и 8 разрядным величинам. Затем используется псевдопеременная SUBSTR для перемещения подстроки из одного расположения в другое.

**declare**

**(P, Q) pointer,**

**A bit(16) based(P),**

**B bit(8) based(Q),**

**I fixed;**

**...**

**I = 4;**

**unspec(P) = 'FF80'b4;**

**unspec(Q) = 'FFF0'b4;**

**...**

**substr(B,4,2) = substr(A,I,2);**

**...**

**Конец раздела 6**

# Управление памятью

## Классы памяти

Каждая переменная в программе PL/I связана с классом памяти. Класс памяти определяет, как и когда компилятор PL/I выделяет память для размещения переменной, и имеет ли переменная свою собственную область памяти или совместно использует ее с другой переменной.

PL/I поддерживает четыре различных класса памяти:

* AUTOMATIC (умолчание PL/I)
* BASED
* PARAMETER
* STATIC

Для классов памяти AUTOMATIC и STATIC компилятор выделяет область памяти перед выполнением, генерируя код, который автоматически связывает имя переменной с данной областью памяти во время выполнения. Для классов памяти BASED и PARAMETER, компилятор сохраняет имя переменной и атрибуты, но не выделяет память для них. Система времени выполнения выделяет и освобождает область памяти BASED и PARAMETER переменной при выполнении программы.

Для повышения производительности, PL/I обрабатывает области памяти AUTOMATIC так же как области памяти STATIC, за исключением процедур, обозначенных как RECURSIVE.

Атрибуты класса памяти являются свойствами элементов, массивов и переменных главных структур. Имена входов, имена файлов или элементов агрегатов данных не могут иметь этих атрибутов.

### Класс памяти AUTOMATIC

Компилятор выделяет область памяти переменной, принадлежащей классу памяти AUTOMATIC перед выполнением главной процедуры. Область памяти остается выделенной до завершения программы. С помощью атрибута INITIAL (см. [*Раздел 7.1.4*](#_Класс_памяти_STATIC)) переменным, принадлежащим к классу памяти AUTOMATIC, можно инициализировать их значения.

Обычно, класс памяти AUTOMATIC вызывает выделение области памяти для данных после входа в блок PROCEDURE или BEGIN, в котором появляется переменная. Для увеличения производительности PL/I выделяет область памяти AUTOMATIC статически.

Единственным исключением является случай рекурсии, при которой переменные AUTOMATIC должны использовать динамический механизм выделения памяти для предотвращения перезаписи данных при рекурсивных вызовах.

### Класс памяти BASED

*Базированная переменная* - переменная, которая описывает область памяти, доступ к которой можно получить с помощью указателя (см. [*Раздел 3.4*](#_Данные_типа_POINTER)). Указатель - адрес, где начинается область памяти базированной переменной, и сама базированная переменная определяет, как PL/I интерпретирует содержимое области памяти, расположенное в этом месте. Таким образом, базированная переменная вместе с указателем эквивалентна небазированной (обычной) переменной.

Можно представить себе базированную переменную как шаблон, который накладывается на область памяти, определенной его базой. Таким образом, базированная переменная и указатель могут ссылаться на область памяти, выделенную для самой базированной переменной, или на область памяти, выделенную для других переменных.

Объявление переменной типа BASED имеет вид:

**DECLARE имя BASED [(указатель)];**

где указатель - неиндексированная переменная типа POINTER или вызов функции без параметров, который возвращает значение POINTER.

Определенная с помощью указателя ссылка может быть неявной или явной. Если переменная объявлена как BASED без указателя, то каждая ссылка на переменную в программе должна включать явный спецификатор указателя следующего вида:

**выражение\_указателя -> переменная**

где указатель\_выражение – вырадение, возвращающая значение типа POINTER.

Если переменная объявлена, как BASED с указателем, можно ссылаться на нее без спецификатора указателя. Система времени выполнения повторно вычисляет ссылку указателя при каждом возникновении неполной переменной, используя выражение указателя, заданное в объявлении переменной.

Следующий пример иллюстрирует различие между явной и неявной ссылкой определяемой указателем.

**main:**

**procedure options(main);**

**declare**

**list\_A(100) fixed binary based,**

**list\_B(100) fixed binary based(list\_B\_ptr),**

**(list\_A\_ptr,list\_B\_ptr) pointer;**

**...**

**list\_A\_ptr -> list\_A(47) = 0; /\* явная ссылка \*/**

**list\_B(47) = 0; /\* неявная ссылка \*/**

**...**

**end main;**

Вы можете объявить указатель с тем же именем в различных блоках и использовать его, как неявную, определенную с помощью указателя ссылку. Однако PL/I принимает имя переменной указателя или имя функции со значением указателя, из объявления BASED, действующего в текущей области. Следующий пример иллюстрирует это понятие.

**A:**

**procedure options(main);**

**declare**

**(i,j) fixed binary,**

**p pointer,**

**x fixed binary based(p);**

**p = addr(i);**

**x = 2; /\* неявная ссылка, x сылается на i \*/**

**B:**

**procedure;**

**declare**

**p pointer; /\* локальный для B \*/**

**p = addr(j);**

**x = 12; /\* неявная ссылка, x все еще ссылается на i \*/**

**p -> x = 3; /\* прямая ссылка, x теперь ссылается на j \*/**

**end B;**

**end A;**

Следующие операторы являются примерами объявления базированных переменных:

**declare A character(8) based;**

**declare B pointer based(Q);**

**declare C fixed based(P);**

**declare D bit(8) based(F());**

### Класс памяти PARAMETER

Если переменная входит в список параметров, компилятор присваивает ей класс памяти PARAMETER. Параметрам область памяти выделяется процедурой вызова, при передаче параметров в вызываемую процедуру.

Ограничения использования класса памяти PARAMETER описаны в *Приложение 1*.

### Класс памяти STATIC

Компилятор выделяет переменной область памяти, принадлежащую к классу памяти STATIC перед выполнением главной процедуры. Область памяти остается выделенной до завершения программы. Значения переменных, относящиеся к классу памяти STATIC, можно инициализировать с помощью атрибута INITIAL.

Атрибут INITIAL предписывает компилятору назначать элементам данных STATIC начальные значения констант после выделения области памяти. Общая форма атрибута INITIAL следующая

**INITIAL (значение[,значение] ...)**

где значение имеет вид:

**[(коэффициэнт\_повторения)] константное\_выражение**

Необязательный коэффициэнт\_повторения - целое число, определяющее сколько раз, повторяется константа. Константное\_выражение должно быть литеральной константой, значение которой совместимо с инициализируемым типом данных. Она может состоять из арифметической константы (возможно) со знаком, или строковой константы, или значения указателя NULL.

Элементы массива можно инициализировать с помощью одного оператора. Инициализация начнется с первого элемента, и будет идти в порядке возрастания правых индексов до исчерпания констант. Количество констант не должно превышать размер инициализируемого массива. Члены структуры должны инициализироваться индивидуально.

Присвоение констант следует правилам операторов присваивания. Например, при присваивании более вместительной переменной символьной строки, имеющей меньшую длину, PL/I дополняет строку пробелами справа.

**Примечание**: только переменные STATIC могут иметь атрибут INITIAL для совместимости с ANSI стандартом PL/I подмножества G.

Следующий код иллюстрирует класс памяти STATIC и атрибут INITIAL:

**declare A fixed binary static initial(0);**

**declare B(8) character(2) initial((8)'AB') static;**

**declare**

**1 fcb static,**

**2 fcb\_drive fixed(7) initial(0),**

**2 fcb\_name character(8) initial('EMP'),**

**2 fcb\_type character(3) initial('DAT'),**

**2 fcb\_ext bit(8) initial('00'B4),**

**2 fcb\_fill(19) bit(8);**

## Оператор ALLOCATE

Оператор ALLOCATE явно выделяет память переменной с атрибутом BASED.

Оператор ALLOCATE имеет вид:

**ALLOCATE базированная\_переменная SET(переменная\_указатель);**

Оператор ALLOCATE предписывает системе времени выполнения, получить сегмент хранения из области динамической памяти, достаточно большой для размещения значения базированной\_переменной. Если сегмент требуемого размера не доступен, система времени выполнения устанавливает состояние ERROR(7).

Базированная переменная должна быть неиндексируемой ссылкой, объявленной с атрибутом BASED в области действия оператора ALLOCATE. Система времени выполнения сохранит выделенный адрес в переменную\_указатель с именем, определенным в пункте SET.

Выделенный этим способом сегмент, остается выделенным до выполнения соответствующего оператора FREE, использующего в качестве операнда выделенный адрес, сохраненный в переменной указателя.

## Многократное выделение памяти

При каждом выполнении ALLOCATE выделяет память. Программа может неоднократно выделять память для единственной базированной переменной, и пока у каждого выделения есть уникальный указатель, программа может ссылаться на них. Например,

**declare names(4) character(10) based;**

**declare (P,Q) pointer;**

**allocate names set(P);**

**P -> names(1) = 'John';**

**...**

**allocate names set(Q);**

**Q -> names(3) = 'Smith';**

**...**

В этом примере во время компиляции память для переменной массива names память не выделяется. Компилятор автоматически распределяет память для указателей P и Q во время компиляции. Во времени выполнения операторы ALLOCATE получают две различные области памяти для имен, на которые можно затем сослаться с помощью соответствующего указателя. *Рисунок 7-1* иллюстрирует эту концепцию.

|  |  |
| --- | --- |
| **Оператор** | **Распределение памяти** |
| **declare names (4) character (10) based;**  **declare (P,Q) pointer;**  **allocate names set (P);**  **P -> names(1) = 'John';**  **allocate names set (Q);**  **Q -> names(3) = 'Smith';** | Не распределена  ┌───┐ ┌───┐  │ P │ │ Q │  └───┘ └───┘  ┌───┐ ┌──────┐  │ P ├───>│John │  └───┘ ├──────┤  ├──────┤  ├──────┤  └──────┘  ┌──────┐  ├──────┤  ┌───┐ ├──────┤  │ Q ├───>│Smith │  └───┘ ├──────┤  └──────┘ |

Рисунок 7‑1 Многократные выделения памяти базированной переменной

**Примечание**: при многократных выделениях памяти с использованием базированной переменной, они все имеют один и тот же указатель, который ссылается только на самое последнее выделение, а не какое-либо из предшествующих.

## Оператор FREE

Область памяти базированной переменной остается выделенной до выполнения оператора FREE. Оператора FREE имеет вид:

**FREE [переменная\_указатель ->] базированная\_переменная;**

где переменная\_указатель адресует выделенную область памяти, которая ранее должна быть получена из области динамической памяти, используя оператор ALLOCATE. Если программа пытается освободить не занятую память, могут возникнуть непредсказуемые результаты.

Если переменная указатель не задана в операторе FREE, то базированная переменная должна быть объявлена с опцией ссылки указателя. В этом случае система времени выполнения возвращает память, адресуемую ссылкой указателя на область динамической памяти.

Подпрограммы времени выполнения, поддерживающие область динамической памяти, автоматически объединяют смежные сегменты памяти, при освобождении оператором FREE.

**Примечание**: когда оператор FREE освобождает выделенную память, указатель и содержимое области хранения становятся неопределенным. Если программа позже использует какую-либо ссылку на освобожденную память, могут возникнуть непредсказуемые результаты.

Следующий код иллюстрирует оператор FREE:

**declare**

**(P, Q, R) pointer,**

**A character(10) based,**

**B fixed based(R);**

**...**

**allocate A set(P);**

**allocate B set(R);**

**allocate A set(Q);**

**...**

**free P -> A;**

**free Q -> A;**

**free B;**

## Встроенная функция NULL

Встроенная функция NULL возвращает уникальное значение указателя, которое является недопустимым адресом памяти. Этот адрес полезен при маркировке различных значений указателей, как пустой, и особенно полезен при построении связного списка.

Связный список - структура данных, составленная из элементов, которые не только содержат область данных, но также содержат указатель на следующий элемент в списке. В таком списке, последний элемент не имеет следующего элемента, и его указатель имеет недопустимое значение (NULL). *Рисунок 7-2* показывает связный список.

┌─────────┐ ┌─────────┐ ┌─────────┐ ┌───────────┐ ┌─────────┐

│Указатель├─>│Элемент\_1│ ┌─>│Элемент\_2│ │Элемент\_n-1│ ┌─>│Элемент\_n│

│на список│ ├─────────┤ │ ├─────────┤ ├───────────┤ │ ├─────────┤

└─────────┘ │Указатель│ │ │Указатель│ │Указатель │ │ │Нулевой │

│ на ├─┘ │ на │...│ на ├─┘ │указатель│

│элемент\_2│ │элемент\_n│ │элемент\_n │ │ │

└─────────┘ └─────────┘ └───────────┘ └─────────┘

Рисунок 7‑2 Связный список

Встроенная функция NULL имеет формат:

**NULL[()]**

При выполнении программы указатель не обязательно имеет начальное нулевое значение. Однако значениям указателя можно задать нулевое значение при помощи значения, возвращенного NULL в опции INITIAL объявления переменной.

NULL является недопустимым описателем указателя базированной переменной. Например, следующая последовательность кода - недопустима в PL/I:

**declare A pointer;**

**declare list(10) fixed binary based(A);**

...

**A = null();**

**A -> list(10) = 32767; /\* это недопустимо! \*/**

Раздел 10 *Руководства программиста на языке PL/I* содержит примеры программ, которые иллюстрируют использование базированных переменных и функции NULL.

## Встроенная функция ADDR

Встроенная функция ADDR возвращает указатель на адрес памяти, занятой переменной, имя которой указано в качестве аргумента. Встроенная функция ADDR имеет вид:

**ADDR(имя\_переменной)**

**Примечание**: имя переменной должно иметь присвоенный адрес памяти и не может быть временным результатом, посредством использования функций и операторов, и при этом оно не может быть константой или именованной константой, например, константой FILE, ENTRY или LABEL.

## Совместное использование памяти

Использование базированных переменных в сочетании с встроенной функцией ADDR позволяет совместное использование участков памяти в PL/I. При совместном использовании памяти базированная переменная не явно задает участок памяти с помощью оператора ALLOCATE. То есть базированная переменная действует как шаблон, который накладывается на память содержащую значение существующей переменной.

Для совместного использования памяти, необходимо использовать встроенную функцию ADDR, чтобы установить адресный указатель базированной переменной равным адресу существующей переменной. После этого доступ к базированной переменной обеспечивает доступ к наложенной переменной. Единственное требование заключается в том, что длина базированной переменной в битах должна быть меньше или равна длине существующей переменной в битах.

Следующая программа иллюстрирует совместное использование памяти. Здесь, значение символьной строки совмещается с вектором строки битов. Выводом программы является значение символьной строки, записанное в шестнадцатеричной форме строки битов.

**declare**

**i fixed binary,**

**ptr pointer,**

**word character(8),**

**bit\_vector(8) bit(8) based(ptr);**

**ptr = addr(word);**

**get list(word);**

**do i = 1 to 8;**

**put edit(bit\_vector(i)) (x(2),b4(2));**

**end;**

Если ввести слово Digital в консоли, в области памяти, отведенной для хранения переменной, слово появляется в следующем виде:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| D | i | g | i | t | a | l | sp |

Базированная переменная bit\_vector является просто шаблоном, который накладывается на область хранение слова как показано ниже:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| xxxxxxxx  D | xxxxxxxx  i | xxxxxxxx  g | xxxxxxxx  i |  |

где x обозначает один бит. Таким образом, на выходе программа считывает строку бит, начинающуюся в расположении слова, и преобразует ее в шестнадцатеричное представление конкретных символов, хранящихся в слове.

**Примечание**: имеется важное соображение, связанное с совместным использованием памяти. Предыдущий пример зависит от знания внутреннего представления данных, используемого PL/I, а именно, что восемь битов представляют символ. Таким образом, программа зависима от реализации. Это нарушает идею подмножества G, ориентированную на написание переносимых программ. Язык PL/I позволяет использовать общую память, используя базированные переменные, но тогда программы становятся зависимыми от реализации языка.

## Замечания по программированию указателей

Базированные переменные и указатели являются мощными инструментами, потому что они предоставляют прямой доступ к памяти. Однако используйте их с осторожностью. Помните, что область памяти, полученная с помощью оператора ALLOCATE, остается выделенной, пока она не будет освобождена или до конца программы. Любые базированные переменные и указатели, относящиеся к выделенным участкам памяти, остаются активными только пока блок, в котором они объявлены, остается активным. При передаче управления вне блока, выделенная память становится недоступной.

**Примечание**: PL/I не может сообщить, что размер базированной переменной не соответствует размеру участка памяти, на которую она ссылается. Если базированной переменной присваивается значение и эта переменная больше длины переменной, на которую она накладывается, содержание смежных участков памяти может быть уничтожено.

При использовании базированных переменных и указателей распространены следующие ошибки:

* присвоение указателю где-нибудь в программе значения NULL и впоследствии использование его в другом месте в качестве ссылки базированных переменных.
* использование указателя, в качестве ссылки на базированную переменную, память которой была освобождена.
* использование указателя, значение которого было потеряно из-за деактивации блока, в котором он был объявлен.

**Конец раздела 7**

# Управление последовательностью выполнения

Операторы программы PL/I обычно выполняются последовательно. Можно использовать операторы управления последовательностью выполнения, чтобы изменить этот нормальный поток с помощью условных и безусловных переходов и управляемых циклов, как описано ниже. Вызовы процедур, включая вызовы функций также изменяют нормальную последовательность выполнения и таким образом считаются операторами управления последовательностью выполнения (см. [*Раздел 2.5*](#_Процедуры_1)).

## Простой оператор DO

DO-группа - последовательность операторов, которая начинается с оператора DO и заканчивается оператором END. Операторы должны быть выполняемые. DO-группа не может определить переменные, среда которых ограничена телом DO-группы. DO-группа может быть в одной из двух форм: простая неитеративная DO-группа и управляемая итеративная DO-группа.

Пример простого оператора DO показан на *Рисунке 8-1*, где оператор\_1 - оператор\_n составляет тело DO-группы.

|  |  |
| --- | --- |
| **DO;**  **оператор\_1;**  **...**  **оператор\_n;**  **END;** |  |

Рисунок 8‑1 Форма оператора DO

Следующая часть программы иллюстрирует простую DO-группу:

**...**

**do;**

**x = 3.14/2;**

**y = sin(x);**

**z = 5 + y;**

**end;**

**...**

## Управляемый оператор DO

Управляемый оператор DO имеет одну из двух основных форм:

**DO WHILE(условие);**

**DO управляющая\_переменная = спецификация\_do;**

где управляющая\_переменная - скалярная переменная, условие - логическое выражение и спецификация\_do - одна из следующих форм:

**start-exp [TO end-exp] [BY incr-exp] [WHILE(условие)]**

**start-exp [BY incr-exp] [TO end-exp] [WHILE(условие)]**

**start-exp [REPEAT repeat-exp] [WHILE(условие)]**

В этих общих формах start-exp - выражение, определяющее начальное значение управляющей переменной; end-exp - выражение, представляющее заключительное значение управляющей переменной; incr-exp - выражение, добавляемое к управляющей переменной после каждого выполнения цикла и repeat-exp - выражение, которое присваивается управляющей переменной после каждой итерации. условие - выражение, приводящее к значению строки битов, являющейся истиной, когда любой из битов в строке равен единице.

Если, часть TO end-exp включена, но BY incr-exp опущена, то PL/I предполагает, что incr-exp равно 1. Две формы, использующие TO и BY, выполняются одинаково и отличаются только в порядком этих двух элементов в тексте программы.

PL/I вычисляет выражение WHILE каждый раз прежде, чем выполнить DO-группу. Если условие - ложь, выполнение цикла завершается и управление передается оператору, расположенному после сбалансированного оператора END.

За исключением выражения REPEAT и выражения WHILE, PL/I вычисляет выражение в спецификации DO до выполнения цикла, так что изменения, внесенные в начальное, конечное или добавляемое значения, не влияют на количество раз выполнения цикла.

Однако, в случае присутствия необязательного REPEAT, PL/I повторно вычисляет repeat-exp после каждой итерации. Затем он присваивает это повторно вычисленное выражение управляющей переменной и выполняет проверку WHILE, если она присутствует.

PL/I определяет действия итеративных групп последовательностью эквивалентной последовательностью операторов IF и GOTO. Выражения e1, e2, e3 и e4 - корректные значения start-exp, end-exp, incr-exp, repeat-exp и условие, в то время как i представляет допустимую управляющую переменную.

### Оператор DO WHILE

**DO WHILE(e1);**

**...**

**END;**

эквивалентен последовательности операторов:

|  |  |
| --- | --- |
| **LOOP:**  **IF ^E1 THEN**  **GOTO END\_LOOP**  **.**  **.**  **.**  **GOTO LOOP**  **END\_LOOP:;** |  |

Рисунок 8‑2 Оператор DO WHILE

### Оператор DO REPEAT

**DO i = e1 REPEAT(e2);**

**...**

**END;**

эквивалентен последовательности операторов показанных на *Рисунке 8-3*:

|  |  |
| --- | --- |
| **i=e1;**  **LOOP:**  **.**  **.**  **.**  **i=e2;**  **GOTO LOOP;** |  |

Рисунок 8‑3 Оператор DO REPEAT

**Примечание**: в этом случае цикл продолжается до тех пор, пока не будет завершен встроенным оператором GOTO или STOP.

### Оператор DO REPEAT WHILE

**DO i = e1 REPEAT(e2) WHILE(e3);**

**...**

**END;**

эквивалентен последовательности операторов показанных на *Рисунке 8-4*:

|  |  |
| --- | --- |
| **i = e1;**  **LOOP:**  **IF ^E3 THEN**  **GOTO END\_LOOP**  **.**  **.**  **.**  **i = e2;**  **GOTO LOOP**  **END\_LOOP:;** |  |

Рисунок 8‑4 Оператор DO REPEAT WHILE

Таким образом, простая итеративная DO-группа:

**DO i = e1 TO e2;**

**...**

**END;**

эквивалентна последовательности операторов:

**DO i = e1 TO e2 BY e3;**

**...**

**END;**

где e3=1, которая может быть выражена как эквивалентная последовательность:

**i = e1;**

**LAST = e2;**

**INCR = e3;**

**LOOP:**

**IF end\_test THEN**

**GO TO END\_LOOP;**

**...**

**i = i + INCR;**

**GOTO LOOP;**

**END\_LOOP:;**

где оператор IF, содержащий end\_test, сравнивает управляющую переменную со значением LAST. Сравнение основывается на знаке, увеличивающегося на единицу значения INCR. Если INCR отрицателен, то end\_test

**IF i < LAST THEN**

**GOTO END\_LOOP;**

Иначе, end\_test становится

**IF i > LAST THEN**

**GOTO END\_LOOP;**

### Оператор DO BY WHILE

**DO e1 TO e2 BY e3 WHILE(e4);**

**...**

**END;**

эквивалентен последовательности операторов показанных на *Рисунке 8-5*:

|  |  |
| --- | --- |
| **i = e1;**  **LAST = e2;**  **INCR = e3;**  **LOOP:**  **IF ^e4 THEN**  **GO TO END\_LOOP;**  **IF end\_test THEN**  **GO TO END\_LOOP;**  **...**  **i = i + INCR;**  **GOTO LOOP;**  **END\_LOOP:;** |  |

Рисунок 8‑5 Оператор DO BY WHILE

В этих эквивалентных последовательностях значение LAST и INCR выполняют функции выражений e2 и e3. Арифметические преобразования и сравнения на каждом шаге выполняются по правилам PL/I.

## Оператор IF

Оператор IF позволяет условное выполнение оператора или группы операторов, основанное на истинном или ложном значении логического выражения. Необязательное предложение ELSE обеспечивает выполнение альтернативного оператора или группы операторов, когда логическое выражение имеет ложное значение.

Оператор IF имеет следующую форму:

**IF выражение THEN действие\_1 [ELSE действие\_2]**

где выражение - скалярное выражение, которое дает значение битовой строки. Действие\_1 и действие\_2 могут быть простыми операторами или составными операторами, расположенными в блоке BEGIN или DO-группе. Если действие\_1 или действие\_2 являются простым оператором, он не может быть оператором DECLARE, END, ENTRY, FORMAT или PROCEDURE. Операторы в действии\_1 и действии\_2, если включены, должны завершиться точкой с запятой. Поэтому, точка с запятой не включена в ранее приведенную общую форму оператора.

PL/I вычисляет выражение для получения битовой строки. Если какой-либо бит в строке равен 1, то PL/I выполняет действие\_1. В противном случае, управление передается в действие\_2, если включено, или следующему оператору в последовательности после оператора IF.

Операторы IF могут быть вложенными, в этом случае PL/I каждый ELSE связывает с самой внутренней несопоставленной парой IF THEN. Вы можете использовать пустые операторы, чтобы вызвать требуемое сопоставление IF ELSE. Например, в следующей последовательности кода, содержащей вложенные операторы IF, пустой оператор после второго ELSE соответствует второму тесту IF THEN.

**┌─ if A = Y then**

**│ ┌─ if Z = X then**

**│ │ ┌─ if W > B then**

**│ │ │ C = 0;**

**│ │ └─ else C = 1;**

**│ └─ else ;**

**└─ else A = Y2;**

## Оператор STOP

Оператор STOP безоговорочно останавливает программу, закрывает все открытые файлы и возвращает управление операционной системе. Оператор STOP можно использовать в любом месте, где необходимо остановить программу.

Оператор STOP имеет вид:

**STOP;**

## Оператор GOTO

Оператор GOTO безусловно передает управление некоторому помеченному оператору. Оператор GOTO имеет любую из форм:

**GOTO метка\_константа | метка\_переменная;**

**GO TO метка\_константа | метка\_переменная;**

где постоянная метка\_константа является литеральной меткой, которая появляется в качестве префикса некоторого помеченного оператора, и метка\_переменная - простая или индексированная переменная метки, которой присвоено значение метки константы.

Вычисленная константа метки должна помечать оператор в пределах оператора GOTO и не может быть во встроенной итеративной группе DO любого вида. Следующий пример иллюстрирует этот недопустимый вид перехода.



Передача управления с помощью оператора GOTO допустима только, когда целевая метка известна в блоке, содержащем оператор GOTO. Таким образом, передача управления, использующая операторы GOTO и метки ограничена в настоящее время активным блоком или содержимым блока.

Ниже приводятся примеры операторов GOTO:

**goto lab1; goto where; goto L(J);**

## Оператор нелокального GOTO

В нелокальном операторе GOTO вычисленная целевая постоянная метка-константа находится вне самого внутреннего блока, содержащего оператор GOTO.

Обычно, желательно избегать нелокальный GOTO, так как он усложняет отладку и сопровождение программы.

Бывают ситуации, когда нелокальный GOTO уместен. Часто в состоянии невосстанавливаемой ошибки полезно перейти непосредственно к глобальной метке восстановления после ошибки, где возобновляется выполнение программы. В этом случае PL/I автоматически отменяет все встроенные ON-блоки и сбрасывает информацию, возвращаемую процедурой.

Следующий код показывает пример нелокального GOTO за пределы процедуры определения:

**p: procedure;**

**on endfile(sysin);**

**begin;**

**goto eof;**

**end;**

**i = 1;**

**do while ('1'b);**

**get file(sysin) list(a(i));**

**i=i+1;**

**end;**

**end p;**

**...**

**call p;**

**eof:**

**Конец раздела 8**

# Обработка состояний

PL/I поддерживает перехват во время выполнения состояний, которые обычно заканчивали бы программу. Операторы ON, REVERT и SIGNAL предоставляют эту функциональную возможность.

Состояние - любое событие, которое прерывает нормальный ход выполнения программы. Состояние может быть сообщено системой во время выполнения или самой программой, из которой управление передается к предустановленному логическому блоку для этого условия.

Некоторые состояния невосстанавливаемые. Это означает, что указанный логический блок не может возвратить управление в точку, в которой возникло состояние, но вместо этого необходимо выполнить GOTO для перехода к нелокальной метке. Другие состояния восстанавливаемые, так что блок может выполнить некоторое локальное действие и затем возвратить управление в точку возникновения состояния.

PL/I распознает следующие общие категории состояний:

* *общее состояние ошибки* (ERROR)
* следующие *состояния арифметических ошибок*:
  + FIXEDOVERFLOW
  + OVERFLOW
  + UNDERFLOW
  + ZERODIVIDE
* и следующие *состояния ввода-вывода*:
  + ENDFILE
  + ENDPAGE
  + KEY
  + UNDEFINEDFILE

## Оператор ON

Оператор ON определяет действие, выполняемое при возникновении указанного состояния.

Оператор ON имеет следующий формат:

**ON имя\_состояния ON-блок;**

где имя\_состояния может быть одним из перечисленных выше состояний.

ON-блок включается, когда он готов перехватить состояние. ON-блок активен, когда он обрабатывает возникшее состояние. ON-блок может быть оператором PL/I, или несколькими операторами PL/I, расположенными в блоке BEGIN. PL/I обрабатывает ON-блок при возникновении конкретного состояния, указанного в операторе ON.

Выход из ON-блока не может быть выполнен с помощью оператора RETURN, хотя это ограничение не исключает возможности определения процедуры в блоке BEGIN.

Как только все операторы ON-блока выполнены, управление передается в точку возникновения состояния, при условии, что состояние восстанавливаемое. Альтернативно, ON-блок может выполнить нелокальный GOTO и передать управление некоторой метке расположенной вне ON-блока.

ON-блок остается активным, пока программа не выполняет соответствующий оператор REVERT, или управление не покидает блок, содержащий оператор ON. Вы можете установить несколько ON-состояний в том же блоке. Например, ERROR(1) и ERROR(2) являются различными состояниями. Однако, если вы устанавливаете несколько ON-блоков для того же состояния в том же блоке, PL/I автоматически восстанавливает предыдущий ON-блок прежде, чем установить новый.

┌─────────────────────────────────────┐

│ ┌─ А: │

│ │ PROCEDURE OPTIONS(MAIN); │

│ │ ... │

│ │ ON ENDFILE │

│ │ ON ENDPAGE │

│ │ ON ERROR(1) │

│ │ ... │

│ │ ┌─ B: │

│ │ │ PROCEDURE; │

│ │ │ ... │

│ │ │ ON ERROR(1) │

│ │ │ ... │

│ │ │ REVERT ERROR(1) │

│ │ │ ... │

│ │ └─ END B; │

│ │ ... │

│ └─ end А; │

└─────────────────────────────────────┘

Рисунок 9‑1 Активация ON-блока

На *Рисунке 9-1* PL/I сначала включает ON-блок для состояний ENDFILE, ENDPAGE и ERROR(1). В этой точке включаются три ON-блока. Когда поток управления переходит в процедуру B, PL/I включает второй ON-блок для состояния ERROR(1) и связывает его с активацией процедуры B. Теперь есть четыре, включенных ON-блока.

Выполнение оператора REVERT выключает текущий ON-блок для ERROR(1), связанный с процедурой B. Это восстанавливает ON-блок для ERROR(1) в охватывающей процедуре A, и снова включены три ON-блока. Обратите внимание, что, если B возвратит управление A, не выполняя оператор REVERT, PL/I автоматически выключает ON-блок.

## Оператор SIGNAL

Оператор SIGNAL программно вызывает возникновение конкретного состояние и вызывает соответствующий ему ON-блок, если он включен. Если ON-блок для состояния не включен, PL/I выполняет действие по умолчанию. Если условие невосстанавливаемое, действие по умолчанию выводит отладочные сообщения и завершает программу.

Оператор SIGNAL имеет вид:

**SIGNAL имя\_состояния;**

где имя\_состояния - одно из состояний, перечисленных ранее.

Например, оператор:

**signal zerodivide;**

вызывает текущий ON-блок ZERODIVIDE.

## Оператор REVERT

Оператор REVERT отключает текущий ON-блок для особого условия и восстанавливает, предшествовавшее ему, если оно существует.

Оператор REVERT имеет вид:

**REVERT имя\_состояния;**

где имя\_состояния - одно из состояний, перечисленных ранее.

Например, оператор:

**revert overflow;**

деактивирует текущий ON-блок состояния OVERFLOW.

**Примечание**: при выходе от блока PROCEDURE, PL/I автоматически деактивирует ON-блок, включенный в блоке.

## Состояние ERROR

Состояние ERROR - самая обширная категория из всех состояний PL/I. Она включает в себя подкоды состояний определенных системой и определяемых пользователем. Существует четыре группы подкодов состояния ошибки:

|  |  |  |  |
| --- | --- | --- | --- |
| (А) | 0 - 63 | Зарезервированы для PL/I | (Невосстанавливаемые) |
| (Б) | 64 - 127 | Определяемые пользователем | (Невосстанавливаемые) |
| (В) | 128 - 191 | Зарезервированы для PL/I | (Восстанавливаемые) |
| (Г) | 192 - 255 | Определяемые пользователем | (Восстанавливаемые) |

Обычно, коды ошибки специфичны для реализации. Смотрите *Руководство программиста Языка PL/I* для получения кодов, присвоенных в настоящее время группе А.

Оператор ON с состоянием ERROR имеет формы:

**ON ERROR[(целое\_выражение)] тело\_оператора;**

**SIGNAL ERROR[(целое\_выражение)];**

**REVERT ERROR[(целое\_выражение)];**

где целое\_выражение - конкретный подкод в диапазоне 0-255.

Например, оператор:

**ON ERROR(3) ... ;**

прерывает состояние ERROR с подкодом 3.

Форма:

**ON ERROR on-body;**

**ON ERROR (0) on-body;**

прерывает любое состояние ошибки, независимо от подкода.

Следующая последовательность кода показывает простой пример обработки состояния ошибки:

**on error(1)**

**begin;**

**put skip list('Invalid Input:');**

**goto retry;**

**end;**

**retry:**

**get list(x);**

Оператор GET читает переменную x из файла SYSIN, и если данные - неверные, система во время выполнения устанавливает состояние ERROR(1). В этом случае управление передается в ON-тело, которое выводит сообщение об ошибке на консоль и возобновляет выполнение в метке retry.

Используя оператор SIGNAL с оператором ON, можно вызвать невосстанавливаемые или восстанавливаемые состояния. Например, оператор:

**signal error(64);**

вызывает состояние ERROR(64), и если есть включенный ON-блок для ERROR(64), то соответствующее ON-тело получает управление. Иначе, программа заканчивается сообщением об ошибке. Оператор:

**signal error(255);**

выполняет аналогичное действие за исключением того, что программа не заканчивается, если ON-блок для состояния ERROR(255) не включен.

## Состояния арифметических ошибок

PL/I обрабатывает несколько арифметических состояний ошибки. Этими состояниями являются:

* FIXEDOVERFLOW[(i)]
* OVERFLOW[(i)]
* UNDERFLOW[(i)]
* ZERODIVIDE[(i)]

где i - необязательное целочисленное выражение, обозначающее определенный подкод ошибки. Подобно функции ERROR, операторы ON, REVERT и SIGNAL могут указать любое из ранее перечисленных состояний.

Если целочисленное выражение не определено, PL/I принимает значение нуля. Оператор ON с подкодом равным 0, перехватывает любой подкод от 0 до 255. PL/I делит подкоды арифметических состояний на значения, определяемые системой и определяемые программой, аналогично функции ERROR.

**Примечание**: все арифметические состояния ошибки невосстанавливаемые. При установке состояния ON для арифметического исключения, ON-тело должно передать управление глобальной метке. Иначе, программа завершается при выходе из ON-блока.

*Таблица 9-1* показывает системные коды для арифметических состояний ошибки.

Таблица 9‑1 Коды состояний арифметических ошибок

|  |  |
| --- | --- |
| **Состояние** | **Значение** |
| FIXEDOVERFLOW(1) | Десятичные операции |
| OVERFLOW(1) | Операции с плавающей точкой |
| OVERFLOW(2) | Преобразование точности с плавающей точкой |
| UNDERFLOW(1) | Операции с плавающей точкой |
| ZERODIVIDE(1) | Десятичное деление |
| ZERODIVIDE(2) | Деление с плавающей точкой |
| ZERODIVIDE(3) | Целочисленное деление |

## Встроенная функция ONCODE

Встроенная функция ONCODE возвращает значение FIXED BINARY(15), представляющее тип ошибки, которая установила самое последнее состояние. Если состояние не активно, ONCODE возвращает ноль. После того, как ON-блок активирован, ONCODE может определить точный источник ошибки.

Следующая последовательность кода иллюстрирует использование ONCODE.

**on error**

**begin;**

**declare**

**code fixed;**

**code = oncode( );**

**if code = 1 then**

**do;**

**put list('Bad Input:');**

**goto retry;**

**end;**

**put list('Error#',code);**

**end;**

**retry:**

## ON-блоки по умолчанию

PL/I имеет по умолчанию ON-блоки для каждой категории состояний, которые обычно выводят соответствующее сообщение об ошибке и заканчивают программу. PL/I не устанавливает состояние FIXEDOVERFLOW при переполнении FIXED BINARY, несмотря на то, что выполняет это при переполнении FIXED DECIMAL.

## Состояния ввода-вывода

При обработке ввода-вывода система времени выполнения программы может установить несколько состояний, касающихся доступа к конкретному файлу. Эти состояния - следующее:

**ENDFILE(файловая\_ссылка)**

**UNDEFINEDFILE(файловая\_ссылка)**

**KEY(файловая\_ссылка)**

**ENDPAGE(файловая\_ссылка)**

где файловая\_ссылка обозначает выражение со значением файла. Значение файла, для которого требуется результат, не должно обозначать открытый файл. [*Раздел 10.5*](#_Состояния_ввода-вывода) подробно описывает каждое из состояний ввода-вывода.

**Конец раздела 9**

# Обработка ввода и вывода

PL/I обеспечивает аппаратно-независимую систему ввода-вывода, которая позволяет программе передавать данные между памятью и внешним устройством, таким как консоль, построчное печатающее устройство (принтер) или дисковый файл.

Набор элементов данных, переданных к или от внешнего устройства, называют *набором данных*. Соответствующая внутренняя файловая константа или переменная называется файлом.

Как и другие элементы данных, необходимо объявить все файлы до их использования в программе. Общая форма объявления файла следующая

**DECLARE file\_id FILE [VARIABLE];**

где file\_id - идентификатор файла. Если отсутствует атрибут VARIABLE, объявление определяет файловую константу. Объявление с атрибутом VARIABLE определяет файловую переменную, которая может получить значение файловой константы с помощью оператора присваивания. Операции ввода-вывода с файловыми переменными допустимы только после присвоения файловой переменной значения файловой константы.

**Примечание**: по умолчанию PL/I присваивает файловой константе атрибут EXTERNAL. Если файловая переменная не объявлена как EXTERNAL, PL/I обрабатывает ее как локальную для блока, в котором она объявлена.

## Оператор OPEN

PL/I требует, чтобы файл был открыт прежде, чем выполнить любые операции ввода-вывода набора данных. Можно открыть файл явно, при помощи оператора OPEN, или неявно, получив доступ к файлу с помощью одного из следующих операторов ввода-вывода:

|  |  |
| --- | --- |
| * GET EDIT * PUT EDIT * GET LIST * PUT LIST | * READ * WRITE * READ VARYING * WRITE VARYING |

[*Разделы 11*](#_Потоковый_ввод-вывод_1) и [*12*](#_Record_I/O) содержат подробные описания различных операторов ввода-вывода.

Оператор OPEN имеет вид:

**OPEN FILE(file\_id) [атрибуты\_файла];**

где file\_id - идентификатор, который появляется в операторе объявления FILE, и атрибуты\_файла обозначают одно или несколько следующих ключевых слов PL/I:

* STREAM | RECORD
* PRINT
* INPUT | OUTPUT | UPDATE
* SEQUENTIAL | DIRECT
* KEYED
* TITLE
* ENVIRONMENT
* PAGESIZE
* LINESIZE

Все атрибуты необязательные, и их можно определять в любом порядке. Несколько атрибутов на той же строке могут конфликтовать между собой, таким образом, можно определить только один. Первый перечисленный является атрибутом по умолчанию. Подмножество G языка PL/I требует, чтобы любое описание конкретной информации было изолировано в атрибутах TITLE и ENVIRONMENT. См. *Приложение 1*.

Потоковый файл (с атрибутом STREAM) содержит данные ASCII переменной длины. Его можно рассматривать как последовательность данных из символов ASCII, организованных в строки и страницы. Каждая строка в потоковом файле определяется маркером строки (linemark), которая является символом перевода строки или парой символов перевода строки и возврата каретки. Каждая страница определяется маркером страницы (pagemark) или символом прогона страницы. Некоторые текстовые редакторы автоматически вставляют символ перевода строки после каждого возврата каретки, но у файлов, которые создает PL/I, могут быть переводы строки, без предшествующего возврата каретки. PL/I обнаруживает конец строки, когда он встречает символ перевода строки.

Записеориентированный файл (с атрибутом RECORD) содержит двоичные данные. PL/I получает доступ к данным блоками, определяемыми объявлением размера записи, или размером элемента данных, используемого для получения доступа к файлу. Также записеориентированный файл может иметь атрибут KEYED, позволяющий использовать ключи FIXED BINARY, для прямого доступ к записям фиксированной длины.

Атрибут PRINT применяется только к потоковым файлам (с атрибутом STREAM) и указывает, что данные предназначены для вывода на построчный принтер.

Для входного файла (с атрибутом INPUT) PL/I предполагает, что файл уже существует, когда он выполняет оператор OPEN. Для выходного файла (с атрибутом OUTPUT) PL/I при выполнении оператора OPEN создает файл. Если файл уже существует, PL/I сначала удаляет старый, затем создает новый файл.

Файл с атрибутом UPDATE может быть прочитан и записан. Если при выполнении оператора OPEN файл не существует, PL/I создает файл с атрибутом UPDATE. Файл с атрибутом UPDATE должен иметь атрибут RECORD и не может иметь атрибут STREAM.

Доступ к последовательным файлам (с атрибутом SEQUENTIAL) осуществляется последовательно с начала до конца. К файлам прямого доступа (с атрибутом DIRECT) получают доступ, в произвольном порядке используя ключи. PL/I автоматически присваивает файлам прямого доступа атрибуты RECORD и KEYED. PL/I требует, чтобы все файлы с атрибутом UPDATE имели атрибут DIRECT, для определения местоположения отдельных записей.

Файл с атрибутом KEYED является простым файлом с записями фиксированной длины. Ключ - относительная позиция записи в файле, основанном на записях фиксированного размера. Ключи используются, для доступа к ключевому файлу. PL/I автоматически присваивает файлу с атрибутом KEYED атрибут RECORD (записеориентированный).

Атрибут LINESIZE применяется только к файлам с атрибутами STREAM OUTPUT и определяет максимальную длину строки ввода или вывода в символах. По умолчанию LINESIZE имеет значение 80.

Атрибут PAGESIZE применяется только к файлам с атрибутами STREAM PRINT и определяет число строк на странице. По умолчанию PAGESIZE равен 60.

Атрибут TITLE(c) определяет программную связь между внутренним именем файла и внешним устройством или файлом в файловой системе операционной системы. Если атрибут TITLE не определен, по умолчанию имя внешнего файла является ссылкой на файл с расширением имени файла DAT. Например

**TITLE('file\_id.DAT')**

Символьная строка c может определить имя внешнего устройства или файла на диске. Как правило, имена внешних устройств зависят от конкретной реализации. В следующей таблице приведены имена внешних устройств, используемые в различных реализациях.

Таблица 10‑1 Имена внешних устройств

|  |  |  |
| --- | --- | --- |
| **Внешнее устройство** | **CP/M** | **IBM DOS** |
| Системная консоль | $CON | CON: |
| Системное устройство печати | $LST | LPT1: или PRN: |
| Системное устройство ввода | $RDR | AUX или COM1: |
| Системный перфоратор | $PUN | AUX или COM1: |

Если символьная строка c содержит имя файла на диске, она должна иметь следующий формат:

**d:имя\_файла.тип;пароль**

Спецификация диска - d:, тип файла и пароль - необязательные. Если пароль определяется в атрибуте TITLE, атрибут ENVIRONMENT (описанный позже в этом разделе) по умолчанию принимает вид ENVIRONMENT(Password(Read)).

**Примечание**: не все реализации поддерживают защиту паролем. См. *Приложение 1*.

Имя файла, тип файла или оба, могут быть записаны в виде 1$ или 2$. Если определен 1$ то, PL/I берет первое имя по умолчанию из командной строки и заполняет его в этой позиции заголовка. Аналогично, вместо 2$ берется второе имя по умолчанию и заполняется в позицию, где он появляется.

Необходимо указать имя файла, и нельзя использовать двусмысленные или подстановочные символы в спецификации имени файла, типа файла или диска. Физическое устройство ввода-вывода можно открыть только как файл STREAM. Устройства чтения должны иметь атрибут INPUT, и устройство перфорации или печати должны иметь атрибут OUTPUT.

Атрибут ENVIRONMENT определяет длину фиксированной и переменной записи для файлов с атрибутом RECORD, размер внутреннего буфера, режим открытия файла и уровень защиты паролем.

Атрибут ENVIRONMENT имеет форму:

**ENVIRONMENT(параметр,...)**

где параметр может быть одним из следующих:

* **Locked**  или **L**
* **Readonly** или **R**
* **Shared**  или **S**
* **Password[(уровень)]** или **P[(уровень)]**
* **Fixed(i)** или **F(i)**
* **Buff(b)**  или **B(b)**

Параметры в атрибуте ENVIRONMENT можно определять в любом порядке за исключением того, что Fixed(i) должен предшествовать Buff(b).

Для открытия файла по умолчанию используется режим Locked (заблокированный) и препятствует другим пользователям доступ к файлу, пока он открыт. Readonly (только для чтения) позволяет больше, чем одному пользователю открывать файл в режиме доступа только для чтения. Режим Shared (совместно используемый) или разблокированный, означает, что несколько пользователей могут открыть файл и получить доступ к нему. В режиме Shared к отдельным записям в файле можно применить встроенные функции LOCK и UNLOCK. Любой из режимов открытия можно сократить до одного символа. Таким образом, можно определить Locked или L, Readonly или R и Shared или S.

Параметр Password[(уровень)] определяет уровень защиты файла паролем. Допустимы следующее уровни защиты:

* Read | R
* Write | W
* Delete | D

Read означает, что для чтения файла требуется пароль. Этот режим используется по умолчанию. Write означает, что файл можно читать, но пароль необходим для записи в файл. Delete означает, что файл может быть прочитан или записан, но требуется пароль для удаления файл. Вы можете сократить каждый из уровней защиты до одного символа. Таким образом, можно указать: Read или R, Write или W и Delete или D.

Параметр Buff(b) предписывает системе ввода-вывода, использовать в качестве буфера область памяти из b байт, где b является выражением FIXED BINARY, значение которое PL/I округляет до ближайшего числа сверху, кратного 128 байтам. Если определен параметр Buff(b) и не определен параметр Fixed(i), система ввода-вывода предполагает, что файл имеет записи переменной длины и поэтому не может иметь атрибут KEYED, поскольку размер записи не является постоянным.

Параметр Fixed(i) определяет файл с записями фиксированной длины, содержащими i байтов каждая, где i является выражение FIXED BINARY, значение которого PL/I внутренне округляет до следующего числа кратного 128 байтам. Если используется этот параметр, необходимо также определить атрибут KEYED. При использовании этого параметра размер буфера по умолчанию составляет i байт, округленный сверху к следующему числу кратному 128 байтам.

Параметры Fixed(i), Buff(b) определяют файл, содержащий записи с фиксированной длиной из i байт, округленной, как описано выше, с размером буфера из b байт, опять же, округленным. Можно определить запись фиксированной длины с размером большим, чем размер буфера. При использовании этих параметров необходимо также определить атрибут KEYED.

## Назначение атрибутов файлов

При выполнении оператора OPEN PL/I устанавливает атрибуты файла прежде, чем связать файл с внешним набором данных. Если оператор OPEN не определяет полный набор атрибутов, PL/I дополняет их подразумеваемыми атрибутами. *Таблица 10-2* показывает подразумеваемые атрибуты для каждого указанного атрибута.

Таблица 10‑2 Подразумеваемые атрибуты PL/I

|  |  |
| --- | --- |
| **Указанный атрибут** | **Подразумеваемый атрибут(ы)** |
| DIRECT | RECORD KEYED |
| KEYED | RECORD |
| PRINT | STREAM OUTPUT |
| SEQUENTIAL | RECORD |
| UPDATE | RECORD |

**Примечание**: оператор OPEN не может содержать конфликтные атрибуты объявленные явно или по умолчанию через механизмы, определяющие подразумеваемые атрибуты.

Каждый тип оператора ввода-вывода неявно определяет конкретный набор атрибутов файла. Если оператор OPEN используется с явным определением атрибутов, подразумеваемые атрибуты операторов ввода-вывода не должны конфликтовать с атрибутами, указанными в операторе OPEN. *Таблица 10-3* приводит допустимые атрибуты для каждого оператора ввода-вывода.

Таблица 10‑3 Допустимые файловые атрибуты операторов ввода-вывода

|  |  |
| --- | --- |
| **Оператор ввода-вывода** | **Допустимые атрибуты** |
| GET FILE(f) LIST | STREAM INPUT |
| PUT FILE(f) LIST | STREAM OUTPUT |
| GET FILE(f) EDIT | STREAM INPUT |
| PUT FILE(f) EDIT | STREAM OUTPUT |
| READ FILE(f) INTO(x) | STREAM INPUT |
| READ FILE(f) INTO(x) | RECORD INPUT SEQUENTIAL |
| READ FILE(f) INTO(x) KEYTO(k) | RECORD INPUT SEQUENTIAL  KEYED ENVIRONMENT(Fixed(i)) |
| READ FILE(f) INTO(x) KEY(k) | RECORD INPUT DIRECT KEYED ENVIRONMENT(Fixed(i))  RECORD UPDATE DIRECT KEYED ENVIRONMENT(Fixed(i)) |
| WRITE FILE(f) FROM(v) | STREAM OUTPUT |
| WRITE FILE(f) FROM(x) | RECORD OUTPUT SEQUENTIAL |
| WRITE FILE(f) FROM(x) KEYFROM(k) | RECORD OUTPUT DIRECT KEYED ENVIRONMENT(Fixed(i))  RECORD UPDATE DIRECT KEYED ENVIRONMENT(Fixed(i)) |

*Таблица 10-4* приводит допустимые атрибуты, которые могут быть явно связаны с любым файлом с помощью оператора OPEN, или неявно при доступе с помощью оператора ввода-вывода.

Таблица 10‑4 Допустимые файловые атрибуты

|  |  |
| --- | --- |
| **Тип** | **Атрибут** |
| STREAM | INPUT ENVIRONMENT TITLE |
| STREAM | OUTPUT ENVIRONMENT TITLE LINESIZE |
| STREAM | PRINT ENVIRONMENT TITLE LINESIZE PAGESIZE |
| RECORD | INPUT SEQUENTIAL ENVIRONMENT TITLE |
| RECORD | OUTPUT SEQUENTIAL ENVIRONMENT TITLE |
| RECORD | INPUT SEQUENTIAL KEYED ENVIRONMENT TITLE |
| RECORD | OUTPUT SEQUENTIAL KEYED ENVIRONMENT TITLE |
| RECORD | INPUT DIRECT KEYED ENVIRONMENT TITLE |
| RECORD | OUTPUT DIRECT KEYED ENVIRONMENT TITLE |
| RECORD | UPDATE DIRECT KEYED ENVIRONMENT TITLE |

**Примечание**: после создания, набор атрибутов применяется только к текущему открытию файла. Можно закрыть файл и открыть его с другим набором атрибутов.

Ниже приведены некоторые примеры оператора OPEN. В каждом примере присутствуют исходный оператор с умолчаниями и дополненные атрибуты, показанные после оператора. Предполагается, что каждый файл объявлен как файловая константа.

Оператор: open file(f1);

Атрибуты: STREAM INPUT ENVIRONMENT(Locked,Buff(128))

TITLE('f1.DAT') LINESIZE(80)

Оператор: open file(f2) print env(r);

Атрибуты: STREAM OUTPUT PRINT ENVIRONMENT(Readonly,Buff(128))

TITLE('f2.DAT') LINESIZE(80) PAGESIZE(60)

Оператор: open file(f3) sequential

title('new.fil;John');

Атрибуты: RECORD INPUT SEQUENTIAL

ENVIRONMENT(Locked,Password(Read),Buff(128))

TITLE('new.fil;john')

Оператор: open title('a:' || c) file(f4)

direct keyed env(s,f(2000));

Атрибуты: RECORD DIRECT INPUT KEYED

ENVIRONMENT(Shared,Fixed(2048),Buff(2048))

TITLE('a:' || c)

Оператор: open update keyed file(f5)

env(locked,f(300),b(100));

Атрибуты: RECORD DIRECT UPDATE KEYED

ENVIRONMENT(Locked,Fixed(384),Buff(128))

TITLE ('f5.DAT')

Оператор: open file(f6) input direct

title('d:accounts.new;topaz')

env(shared,p(d),f(100),b(2000));

Атрибуты: RECORD DIRECT INPUT KEYED

ENVIRONMENT(Shared,Password(Delete),Fixed(128),Buff(2048))

TITLE('d:accounts.new;topaz')

В каждом из приведенных выше примеров PL/I позволяет целочисленные выражения везде, где появляется константа. Таким образом, оператор:

**open file(f1) linesize(k+3) pagesize(n-4) env(b(x+128));**

является допустимым оператором OPEN.

## Оператор CLOSE

Оператор CLOSE отсоединяет файл от внешнего набора данных, очищает и освобождает внутренние буферы и окончательно записывает выходные файлы на диск. Оператор CLOSE имеет вид:

**CLOSE FILE(идентификатор\_файла);**

где идентификатор\_файла - ссылка на файл. Вы можете впоследствии вновь открыть ранее описанный файл, используя оператор OPEN. Если файл не открыт, PL/I игнорирует оператор CLOSE.

## Блок параметров файла

PL/I связывает каждую файловую константу с блоком параметров файла (FPB[[2]](#footnote-2)). FPB является статически выделенным сегментом памяти, содержащим информацию о файле. Файловая переменная не имеет соответствующего FPB, пока ей не присвоена ее файловая константа. Каждый FPB содержит следующую информацию:

* заголовок файла, называющий внешнее устройство или набор данных, связаый с файлом;
* позиция столбца, которую поддерживает система времени выполнения, чтобы определить местоположение следующей позиции для получения или помещения данных в потоковый файл;
* текущее значение счетчика строки в выходных потокоаых файлах;
* текущее значение счетчика страниц для файлов печати;
* текущий номер записи;
* размер строки;
* размер страницы;
* размер записи фиксированной длины;
* размер внутреннего буфера;
* дескриптор файла, содержащий один из допустимых наборов атрибутов файла, описанных ранее.

Когда файл открыт, система времени выполнения поддерживает запись в структуре данных, названной Open List, для которой выделяется память из свободной области. Кроме того, пока файл открыт, FPB содержит указатель на адрес блока управления файлом (FCB[[3]](#footnote-3)) операционной системы.

## Состояния ввода-вывода

Во время обработки ввода-вывода система времени выполнения может активизировать несколько состояний, касающихся доступа к конкретному файлу. Эти состояния следующие:

**ENDFILE(файловая\_ссылка)**

**UNDEFINEDFILE(файловая\_ссылка)**

**KEY(файловая\_ссылка)**

**ENDPAGE(файловая\_ссылка)**

где файловая\_ссылка - файловая константа или файловая переменная, еще не открытого файла.

### Состояние ENDFILE

Система времени выполнения устанавливает состояние ENDFILE всякий раз, когда она читает символ конца-файла Ctrl-Z из файла STREAM, или она сталкивается с физическим концом файла в файле с атрибутом RECORD обрабатываемом в режиме SEQUENTIAL (последовательного доступа). Операция чтения файла (прямого доступа) DIRECT с помощью ключа за пределами конца файла - также устанавливает состояние ENDFILE. Выходные операции вывода, которые превышают емкость диска, устанавливают состояние ERROR(14).

### Состояние UNDEFINEDFILE

Система времени выполнения устанавливает состояние UNDEFINEDFILE всякий раз, когда программа пытается открыть файл для ввода, и файл не существует на указанном диске. Если программа пытается открыть защищенный паролем файл без правильного пароля, система времени выполнения также устанавливает состояние UNDEFINEDFILE. Также, система времени выполнения устанавливает это состояние, если программа пытается получить доступ к устройству физического ввода-вывода в качестве файла KEYED или UPDATE.

### Состояние KEY

Система времени выполнения устанавливает состояние KEY, когда программа использует недопустимое значение ключа во время операции ввода-вывода.

### Состояние ENDPAGE

Система времени выполнения устанавливает состояние ENDPAGE для файлов с атрибутом PRINT (печати), когда значение текущей строки достигает PAGESIZE для указанного файла. Текущая строка всегда начинается с нуля, и система времени выполнения увеличивает ее на единицу для каждого символа перевода строки, который отправляется в файл. Если файл изначально открыт с PAGESIZE(0), то система времени выполнения никогда не устанавливает состояние ENDPAGE. Система времени выполнения сбрасывает текущую строку на единицу всякий раз, когда:

* символ прогона страницы отправлен в выходной файл;
* выполняется оператор PUT с опцией PAGE;
* выполняется системное действие по умолчанию для ENDPAGE.

Если система времени выполнения устанавливает состояние ENDPAGE во время выполнения опции SKIP, обработка SKIP завершается.

Если ON-блок перехватывает условие ENDPAGE, но не выполняет оператор PUT с опцией PAGE, то значение текущей строки не сбрасывается в единицу. То есть, пока программа не выполнит оператор PUT с опцией PAGE, система времени выполнения продолжает увеличивать значение текущей строки, и не устанавливает состояние ENDPAGE. Значение текущей строки увеличивается до 32767, а затем снова начинается с 1.

### ON-блоки ввода-вывода по умолчанию

Если ON-блок получает управление для состояний ENDFILE, UNDEFINEDFILE или KEY и возвращается к точке, в которой возникло состояние, система времени выполнения прекращает текущую операцию ввода-вывода, и передает управление оператору, следующему за оператором ввода-вывода, который вызывает появление состояния.

Если ON-блок не включен для состояний ENDFILE, UNDEFINEDFILE или KEY, системное действие по умолчанию завершает программу и выводит соответствующее сообщение об ошибке.

Если ON-блок для ENDPAGE не включен, системное действие по умолчанию выполняет PUT PAGE в выходной файл и продолжает обработку.

## Встроенные функции состояний ввода-вывола

PL/I поддерживает несколько встроенных функций, которые полезны при обработке ввода-вывода. Эти функции следующие:

|  |  |
| --- | --- |
| * ONFILE * ONKEY | * PAGENO * LINENO |

### Функция ONFILE

Функция ONFILE возвращает значение символьной строки внутреннего имени файла, участвующего в последней операции ввода-вывода, которая привела к возникновению состояния. При ошибке преобразования функция ONFILE возвращает имя активного в тот момент файла. Если возникшее состояние не связано с файлом, то ONFILE возвращает пустую строку.

Следующий фрагмент программы иллюстрирует использование ONFILE.

**on error(1)**

┌─ **begin;**

│ **put list('Bad Data in file:',onfile());**

│ **goto retry;**

└─ **end;**

**retry:**

### Функция ONKEY

Функция ONKEY возвращает значение ключа, участвующего в операции ввода-вывода, вызвавшей состояние KEY. ONKEY действительна только в ON-теле активированного ON-блока. Следующий фрагмент кода иллюстрирует использование ONKEY.

**on key(newfile)**

**put skip list('bad key',onkey());**

### Функция PAGENO

Функция PAGENO возвращает текущий номер страницы для файла PRINT, с именем в качестве параметра. Когда возникает состояние ENDPAGE в результате выполнения оператора PUT, номер строки на единицу больше, чем размер страницы для файла.

### Функция LINENO

Функция LINENO возвращает текущий номер строки для файла PRINT, с именем в качестве параметра. Когда возникает состояние ENDPAGE в результате выполнения оператора PUT, номер строки на единицу большее, чем размер страницы для файла.

## Предопределенные файлы SYSIN и SYSPRINT

PL/I содержит две предопределенные файловые константы под названием SYSIN и SYSPRINT. Эти файловые константы не нужно объявлять, если указывается явная ссылка на файл с ними в операторах OPEN, GET, PUT, READ или WRITE.

В противном случае, PL/I открывает SYSIN с атрибутами по умолчанию:

**STREAM INPUT ENVIRONMENT(Locked,Buff(128)) TITLE('$CON') LINESIZE(80)**

и SYSIN становится консольной клавиатурой.

**Примечание**: TITLE в IBM DOS - 'CON:'.

PL/I открывает SYSPRINT с атрибутами по умолчанию:

**STREAM PRINT ENVIRONMENT(Locked,Buff(128)) TITLE('$CON')**

**LINESIZE(80) PAGESIZE(0)**

и SYSPRINT становится консольным выводом на дисплей.

**Примечание**: TITLE в IBM DOS - 'CON:'.

## Категории ввода-вывода

PL/I поддерживает два метода доступа к файлу:

* ввод-вывод STREAM (только последовательный доступ)
* ввод-вывод RECORD (последовательный или произвольный доступ)

### Потоковый (STREAM) ввод-вывод

Файл STREAM представляет собой последовательность символов ASCII, возможно содержащих маркеры строк и страниц. При передаче данных в файл STREAM в или из внешних устройств, PL/I может отформатировать данные и выполнить преобразование в другие типы данных. [*Раздел 11*](#_Потоковый_ввод-вывод_2) содержит полные описания операторов потокового (STREAM) ввода-вывода.

### Записеориентированный (RECORD) ввод-вывод

При записеориентированном (RECORD) вводе-выводе отдельные элементы данных называют записями, и они различаются по размеру согласно объявления данных. PL/I не выполняет преобразование при передаче данных, используя операторы ввода вывода RECORD (см. [*Раздел 12*](#_Записеориентированный_ввод-вывод)), а просто передает внутреннее представление элемента данных.

**Примечание**: различные компьютеры используют различные внутренние представления данных PL/I. Не рассчитывайте, что можно обменяться файлом между двумя различными компьютерами.

**Конец раздела 10**

# Потоковый ввод-вывод

PL/I поддерживает три формы потокового (STREAM) ввода-вывода:

* *Управляемый списком*. Передача данных без спецификации формата.
* *Управляемый редактированием*. Позволяет форматированный доступ к символьным элементам данных (см. [*Раздел 11.3*](#_EDIT-directed_I/O_1)).
* *Строчно-ориентированный*. Позволяет доступ к символьным данным переменной длины в неотредактированной форме. Обратите внимание, что PL/I обеспечивает строчно-ориентированный потоковый ввод-вывод, используя операторы READ и WRITE, которые могут быть не доступны в других реализациях PL/I.

Следующие правила относятся ко всем формам потокового ввода-вывода:

* Номер столбца, строки и страницы для всех файлов первоначально установлены равными 1;
* Каждое появление маркера строки и страницы сбрасывает позицию колонки в 1.
* Если входной или выходной символ является - специальным символом, позиция столбца увеличится на единицу.
* При выводе, если позиция столбца превышает размер строки, система времени выполнения записывает маркер строки, увеличивает номер строки на единицу и сбрасывает позицию столбца в единицу.
* Когда номер строки превышает размер страницы, система времени выполнения устанавливает состояние ENDPAGE. Если для ENDPAGE ON-блок не включен, система времени выполнения записывает маркер страницы, увеличивает номер страницы на единицу и сбрасывает позицию столбца и номер строки в единицу.

В *Таблице 11-1* приведены соглашения о присвоении имен появляющихся в данном разделе при описании различных операторов потокового ввода- вывода.

Таблица 11‑1 Соглашения о присвоении имен потокового ввода- вывода

|  |  |
| --- | --- |
| **Имя** | **Значение** |
| **file-id** | Идентификатор файла. |
| **nl** | Выражение FIXED BINARY, определяющее количество маркеров строки, пропускаемых при вводе или количество маркеров строки, записываемых перед элементом данных при выводе. |
| **список\_ввода** | Список переменных, разделенных запятыми, к которым PL/I передает элементы данных из входного потока. Входной список определяет число и порядок переменных, которым присваиваются входные данные из потока. В PL/I переменные должны быть скалярными величинами. Можно включать итеративные DO-группы во входной список, но они требуют дополнительный набор круглых скобок. Формат заголовка DO совпадает с оператором DO за исключением того, что пункт REPEAT не допускается. Общий формат (элемент\_1,...,элемент\_n DO шаг\_цикла). Например, следующие утверждения эквивалентны:  **do i = 1 to 10;**  **put list(A(i));**  **end;**  **put list((A(i) do i = 1 to 10));** |
| **список\_вывода** | Список выходных элементов, состоящий из констант, переменных или выражений разделенных запятыми. Выходной список может также может включать итеративные DO группы. |

## Ввод-вывод управляемый списком

К входному потоку при вводе-выводе управляемом списком применяются следующие ограничения:

* Элементы данных в потоке могут быть арифметическими константами, константами строки символов или константами строки битов.
* Каждый элемент данных должен сопровождаться разделителем, который состоит из серии пробелов, запятой, необязательно окруженной пробелами или символом конца строки.
* PL/I обрабатывает встроенный символ табуляции (Ctrl-I) как пробел.
* Данные символьной строки, которые фактически содержат пробелы или запятые, должны быть заключены в апострофы. В противном случае, PL/I обрабатывает пробелы или запятые как разделители.
* Запятая в качестве первого, непустого символа во входной строке или две последовательные запятые, необязательно разделенные одним или более пробелами, указывает нулевое поле во входном потоке. Нулевое поле указывает на то, что никакие данные не будут переданы к соответствующему элементу данных во входном списке. Таким образом, значение элемента данных назначения остается неизменным.

### Оператор GET LIST

Оператор GET LIST считывает данные, используя управляемый списком потоковый ввод-вывод. Оператор GET LIST имеет формат:

**GET [FILE(file-id)] [SKIP [(nl)]] LIST(список\_ввода);**

Опции FILE или SKIP можно определить в любом порядке. LIST должен быть последним. Если не определена опция FILE, PL/I использует FILE(SYSIN) . В операторе GET с опцией SKIP система времени выполнения игнорирует nl маркеров строки. Если не определен nl с опцией SKIP, то по умолчанию значения nl устанавливается равным 1, и система времени выполнения игнорируют 1 маркер строки.

После передачи всех элементов данных переменным, указанным во входном списке, позиция столбца во входном потоке остается на символе, следующем за прочтенным элементом данных.

Во входной поток можно включить дополнительные символьные строки в апострофах. Если они присутствуют, система времени выполнения не передает апострофы входной переменной. Аналогично, для констант строки битов, система времени выполнения не передает входной переменной апострофы и завершающий символ B.

PL/I ограничивает ввод строк одной строкой. Таким образом, ввод строки с консоли требует только ведущий апостроф, когда строка заканчивается возвратом каретки.

### Оператор PUT LIST

Оператор PUT LIST записывает данные, используя управляемый списком потоковый ввод-вывод. Оператор PUT LIST имеет формат:

**PUT [FILE(file\_id)] [SKIP[(nl)]] [PAGE] LIST(список\_вывода);**

Опции FILE, SKIP или PAGE можно определять в любом порядке. LIST должен быть последним. Если не определена опция FILE, PL/I использует FILE(SYSPRINT).

Если не определен nl с опцией SKIP, то по умолчанию значения nl устанавливается равным 1. Если nl = 0, система времени выполнения не записывает маркер строки, но сбрасывает позицию столбца к 1. В любом случае, при использовании опции SKIP, система времени выполнения сбрасывает позицию столбца к 1.

Опция PAGE допустима только для файлов PRINT. Каждый раз, когда система времени выполнения записывает маркер страницы, номера позиции столбца и строки сбрасываются к 1.

При записи элементов данных в потоковый файл PL/I преобразует элементы из выходного списка в их представление символьной строки. Система времени выполнения использует пробелы для разделения данных в выходной строке. Если элемент данных более длинный, чем оставшееся число символов в выходной строке, система времени выполнения записывает элемент в начале следующей строки. Если длина строки символов представления элемента данных превышает размер строки, система времени выполнения записывает элемент данных отдельно на одной строке, которая превышает размер строки.

Если передача при выводе превышает размер страницы, PL/I устанавливает состояние ENDPAGE.

PL/I обычно записывает символьные строки, заключенными в апострофы. Каждый встроенный апостроф записывается как два апострофа (''). Однако, если файл имеет атрибут PRINT, дополнительные апострофы опускаются. PL/I всегда записывает данные строки битов, заключенными в апострофы с завершающей буквой B.

## Строчно-ориентированный ввод-вывод

PL/I поддерживает две формы оператора READ и WRITE для обработки записей ASCII переменной длины в потоковом файле. Две формы, названные READ VARYING и WRITE VARYING обычно не доступны в других реализациях. В программах PL/I необходимо избегать использования этих операторов, если важна совместимость.

### Оператор READ VARYING

Оператор READ VARYING читает строку переменной длины из входного потока.

Оператор READ VARYING имеет формат:

**READ [FILE(file\_id)] INTO(v);**

где v - символьная строка переменной длины. Если не определена опция FILE, PL/I использует FILE(SYSIN).

Оператор READ VARYING читает данные из входного файла, пока не достигает максимальной длины v, или до чтения символа перевода строки. READ VARYING присваивает длину v к числу прочитанных символов, включая символ перевода строки.

**Примечание**: если файл не открыт явно, оператор READ VARYING неявно открывает файл с атрибутами STREAM и INPUT.

Учитывая объявление:

**declare**

**F file,**

**1 buffer,**

**2 buffch character(254) varying;**

оператор:

**read file(F) into(buffer);**

производит передачу данных как запись, поскольку цель представляет собой структуру, а не переменную CHARACTER VARYING. Однако PL/I интерпретирует оператор:

**read file(F) into(buffch);**

как передачу данных потокового ввода ASCII, поскольку целевая переменная является CHARACTER VARYING.

Оператор READ VARYING отличается от оператора READ только тем, что целевая переменная имеет атрибуты CHARACTER VARYING.

### Оператор WRITE VARYING

Оператор WRITE VARYING записывает потоковые данные ASCII переменной длины. Оператора WRITE VARYING имеет формат:

**WRITE [FILE(file\_id)] FROM(v);**

где v - символьная строка переменной длины. Если не определена опция FILE, PL/I использует FILE(SYSPRINT).

PL/I не добавляет дополнительные управляющие символы в выходную строку. Если приложение требует управляющие символы, их необходимо включить в строку. Напоминаем, что PL/I позволяет встроенные управляющие символы как часть строковых констант, обозначенных предшествующим символом ^ в строке.

**Примечание**: если файл не открыт явно, оператор WRITE VARYING неявно открывает файл с атрибутами STREAM и OUTPUT.

Например, учитывая предыдущее объявление, PL/I интерпретирует оператор:

**write file(F) from(buffer);**

как передачу данных записи. PL/I интерпретирует

**write file(F) from(buffch);**

как оператор WRITE VARYING, воздействующий на потоковый выходной файл ASCII, потому что исходная переменная - CHARACTER VARYING.

Оператор WRITE VARYING отличается от оператора WRITE только тем, исходная переменная имеет атрибуты CHARACTER VARYING.

## Ввод-вывод управляемый редактированием

Входной список и выходной списки ввода-вывода, управляемого редактированием походят на аналогичные для ввода-вывода, управляемого списком. Однако, ввод-вывод, управляемый редактированием, использует список форматов, чтобы определить, как PL/I читает и пишет данные.

### Список форматов

Список форматов - список элементов форматов, разделенных запятыми.

Существуют три типа элементов форматов:

* Элементы форматов данных, описывающие считываемые элементы данных.
* Элементы форматов управления, определяющих расположение элементов данных в потоке.
* Удаленные элементы форматов, которые ссылаются на другой список форматов.

Список форматов имеет форму:

**[n] элемент\_формата,... [,[n] элемент\_формата]**

где n - значение целочисленной константы в диапазоне 1 - 254 представляющее коэффициент повторения следующего элемента\_формата. Если он опущено, PL/I устанавливает значение коэффициент повторения равным единице. Элементом\_формата является или элементом формата данных или элементом формата управления.

Элементом\_формата может также быть удаленный элемент формата. Однако, в PL/I удаленный элемент формата должен быть единственным форматом в списке, и ему не может предшествоваться коэффициент повторения.

### Элементы формата данных

Элементы формата данных читают или пишут числовые или символьные поля в или из внешнего потокового набора данных. PL/I поддерживает следующие элементы формата данных:

**Формат A[(w)]**

Этот формат читает или записывает w символов данных символьной строки. В GET EDIT, необходимо включать w, чтобы быть совместимыми с полной версией PL/I. Однако, PL/I позволяет, опускать w в GET EDIT и, формат читает остаток из текущей строки до, но, не включая перевод строки и возврат каретки.

|  |  |  |
| --- | --- | --- |
| **Входное значение** | **Формат** | **Результат** |
| byte | A(6) | 'byte' |
| Napoleon | A(10) | 'Napoleon' |
| string | A | 'string' |

В PUT EDIT, если w опущен, то формат предполагает, что w равняется длине выходной строки. Если w больше, чем длина выходной строки, формат добавляет пробелы справа. Если w - меньше, чем длина выходной строки, формат обрезает строку в самой правой позиции.

|  |  |  |
| --- | --- | --- |
| **Значение** | **Формат** | **Результат** |
| abcdef | A(6) | abcdef |
| abcdef | A(3) | abc |
| ⌴ | A(4) | ⌴⌴⌴⌴ |

**Формат B[nl[(w)]**

Этот формат читает или пишет данные строки битов. В GET EDIT необходимо включать w. n задает число битов, которые будут использоваться для каждой цифры. Если n опущен, значение по умолчанию равняется 1, таким образом, B эквивалентен B1, и во входном потоке могут быть только 0 и 1, иначе PL/I устанавливает условие ERROR(1). Допустимые цифры для каждого значения n являются следующими:

**n Допустимые цифры**

1 0, 1

2 0, 1, 2, 3

3 0, 1, 2, 3, 4, 5, 6, 7

4 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

|  |  |  |
| --- | --- | --- |
| **Входное значение** | **Формат** | **Результат** |
| 00101 | B(5) | '00101'B |
| 22 | B2(2) | '1010'B |
| 7C4 | B4(3) | '011111000100'B |

В PUT EDIT формат B сначала преобразует переменную к типу битовой строки, и затем преобразует его в представление символьной строки. Если w отсутствует, формат B выводит получающуюся символьную строку. Если w включен, и его значение имеет длину больше, чем символьная строка, то формат B дополняет строку пробелами справа. Если получающаяся символьная строка длиннее w, система времени выполнения устанавливает состояние ERROR(1).

|  |  |  |
| --- | --- | --- |
| **Значение** | **Формат** | **Результат** |
| '00'B | B | 00 |
| '1'B | B(4) | 0001 |
| '011101'B | B3(2) | 35 |

**Формат E(w[,d])**

Этот формат читает и пишет данные с плавающей точкой. В GET EDIT, формат E преобразует входные символы в значения FLOAT BINARY. w - ширина поля и d - число цифр справа от десятичной точки.

|  |  |  |
| --- | --- | --- |
| **Входное значение** | **Формат** | **Результат** |
| ⌴⌴⌴⌴ | E(4) | 0 |
| 2.9E7 | E(5,3) | .29E+8 |
| 345678 | E(6,2) | .345678E+4 |

В PUT EDIT формат E преобразует элемент данных в FLOAT BINARY и представляет его в научной нотации. w должен быть, по крайней мере, на 7 больше, чем d, так как выходное поле появляется как +n.ddddE+eee, где + представляет позиции знака, n - ведущая цифра, dddd представляет дробную часть длины d, и E+eee представляет поле показателя степени.

|  |  |  |
| --- | --- | --- |
| **Значение** | **Формат** | **Результат** |
| 0 | E(11,3) | ⌴0.000E+000 |
| 4.7E-10 | E(11,3) | ⌴4.700E-010 |
| -30 | E(15) | ⌴-3.000000E+001 |

**Формат F(w[,d])**

Этот формат чтения и записи арифметических данных с фиксированной точкой. w - ширина, число символов в поле, и d - число символов справа от десятичной точки.

В GET EDIT, формата F считывает столько символов, сколько указано в w. Если символьная строка содержит десятичную точку, то десятичная точка определяет масштабный коэффициент. В противном случае, масштабный коэффициент определяет d. Формат F игнорирует начальные и конечные пробелы. Если поле содержит только знаки пробела, формат F считывает значение ноль.

|  |  |  |
| --- | --- | --- |
| **Входное значение** | **Формат** | **Результат** |
| ⌴⌴0⌴⌴ | F(5) | 0 |
| ⌴-6⌴ | F(4) | -6 |
| 13.09 | F(5) | 14 |

В PUT EDIT формат F преобразует элемент данных в FIXED DECIMAL, и затем использует d для определения масштабного коэффициента выходного значения. Если d опущен, масштабный коэффициент - ноль. Формат F округляет выходное значение, если у переменная не имеет точности 15. Формат F подавляет начальные нули за исключением одного сразу слева от десятичной точки.

|  |  |  |
| --- | --- | --- |
| **Значение** | **Формат** | **Результат** |
| 0 | F(5,1) | ⌴⌴0.0 |
| -27 | F(5,1) | -27.0 |
| .39 | F(6,2) | ⌴⌴0.39 |

### Элементы формата управления

Элементы формата управления, используются для размещения строк, страниц и пустого пространства. PL/I обрабатывает элементы формата, по мере появления их в списке формата, и игнорирует любые элементы, которые остаются после входного списка или исчерпания выходного списка. PL/I поддерживает следующие элементы формата управления.

**COLUMN(nc)**

Этот элемент перемещает указатель формата на столбец nc в потоке данных ввода или вывода. В GET EDIT, элемент COLUMN игнорирует эти символы, оставляя без внимания, располагая указатель формата в столбец nc. Если текущая позиция столбца меньше, чем nc, указатель формата перемещается в позицию столбца nc. Если текущая позиция столбца больше nc, указатель сначала перемещается на следующую строку, а затем перемещается в новое положение столбца nc. Если nc превышает самую правую позицию в строке, указатель формата перемещается в первый столбец новой строки. В GET EDIT, перемещение указателя формата отбрасывает вводимые символы.

В PUT EDIT, элемент COLUMN пишет пробелы в процессе позиционирования в столбец nc. Кроме того, если текущая позиция больше, чем nc, система времени выполнения выводит маркер строки, а затем выводит пробелы, пока она не достигает столбца nc новой строки. Если nc превышает размер строки, система времени выполнения записывает маркер строки и устанавливает позицию столбца равной 1.

**LINE(ln)**

Этот элемент применяется только для файлов PRINT и определяет номер строки следующего записываемого элемента данных. Константа ln должна быть больше ноля. Если текущий номер строки равен ln, LINE(ln) не оказывает никакого эффекта. Если текущий номер строки - меньше ln, то система времени выполнения выводит маркер строки, пока текущий номер строки не сравняется с ln. PL/I устанавливает состояние ENDPAGE, если число записанных маркеров строки превышает текущий размер страницы.

**PAGE**

Этот элемент используется только с файлами PRINT, и он заставляет систему времени выполнения записать маркер страницы, увеличить номер страницы на единицу и установить номер строки и позицию столбца равными 1.

**SKIP[(nl)]**

Этот элемент определяет число пропущенных или записанных маркеров строки (nl). Если nl отсутствует, по умолчанию устанавливается значение равное 1. Система времени выполнения устанавливает позицию столбца равной 1.

В GET EDIT nl означает число маркеров строки, пропускаемых прежде, чем переместиться к следующему элементу формата. Система времени выполнения отбрасывает первую строку, если программа выполняет SKIP(1) в качестве первого элемента формата сразу после явной или неявной операции открытия. SKIP(0) не определено для входных потоков.

В PUT EDIT nl определяет число записываемых маркеров строки. Если в процессе записи маркеров строки в файл PRINT превышен размер страницы, система времени выполнения устанавливает состояние ENDPAGE и, по возврату из ON-блока, прекращает обрабатывать операцию SKIP.

**X(sp)**

Этот элемент перемещает указатель формата на sp позиций в потоке данных ввода или вывода. В GET EDIT sp является числом продвигаемых символов. Система времени выполнения игнорирует маркеры строк и продолжает обработку следующей строки. В PUT EDIT sp является числом записываемых пробелов. Если достигнут конец строки, система времени выполнения записывает маркер строки, и операция заполнения пробелами продолжается на следующей строке.

### Элементы удаленного формата

Элемент удаленного формата использует список формата оператора FORMAT вместо элемента формата. Удаленный элемент формата имеет формат:

**R(метка\_формата)**

где метка\_формата - константа метки предшествующая ключевому слову FORMAT

**put edit(a,b,c) (r(elsewhere));**

### Оператор FORMAT

Оператор FORMAT определяет удаленный элемент формата и имеет общий вид:

**метка\_формата: FORMAT(список\_форматов);**

где метка\_формата - константа метки соответствующего оператора FORMAT, и список\_форматов - список элементов формата, аналогичный описанным в предыдущем разделе. Например, оператор FORMAT:

**L1: format(a(5),f(6,2),skip(3),a(2));**

ссылается как удаленный формат оператором:

**get edit (a,b,c) (r(L1));**

### Элементы формата шаблон

Элемент формата данных шаблон (picture) используется при выводе для редактирования числовых данных в форме десятичного числа с фиксированной точкой. Значением, в результате такого редактирования, является символьная строка, формат которой определяется числовым значением и спецификацией шаблона в элементе формата шаблон. Ниже приводится форма элемента формат шаблон:

**P'спецификация'**

где спецификация - константа символьной строки, описывающая спецификацию шаблона.

Элемент формата шаблона может появиться в операторе PUT EDIT, как и любой другой элемент формата данных.

**Синтаксис шаблона**

Константа символьной строки, описывающая спецификацию шаблона, должна состоять из одного или более специальных символов как показано в *Таблице 11-2*.

Таблица 11‑2 Символы формата шаблона

|  |  |
| --- | --- |
| **Символ** | **Результат** |
| $ + - S | статические или плавающие символы |
| \* Z | условные цифровые символы |
| 9 | цифровой символ |
| V | позиция символа десятичной точки |
| / , : B | символы вставки |
| CR DB | символы кредита и дебета |

Эти символы должны удовлетворять определенным правилам синтаксиса. Символы вставки могут появляться в любом месте допустимой спецификации шаблона, за исключением того, что они не должны разделять символы шаблона CR и DB.

Если все символы спецификации шаблона вставки удалены, получающаяся строка должна быть приемлема для распознавания недетерминированным конечным автоматом, показанным на *Рисунке 11-1*. Т.е. должно быть, возможно, начиная с узла Начало, проследить по этой схеме, до узла Принять, где переходы через край позволены, если край не помечен или, если край помечен, следующим символом в спецификации шаблона.

Следующие константы символьных строк определяют допустимые спецификации шаблона:

**'BB$\*\*\*,\*\*\*V.99BB'**

**'$----,999V.99BCR'**

**'99:99:99'**

**':BBB$SSSS,SSS.VSSBBB:'**



Рисунок 11‑1 Распознаватель спецификации шаблона

**Семантика шаблона**

Типы символов шаблона, появляющиеся в спецификации, определяют, как спецификация шаблона редактирует числовое значение в значение символьной строки.

В спецификации шаблона определенные символы встречаются как статические или дрейфующие символы. Этими символами являются:

* **$** знак доллара;
* **+** знак плюс;
* **-** знак минус;
* **S** знак S в верхнем регистре.

Такой символ статичен, если в спецификации шаблона он появляется только один раз. В противном случае, он дрейфует. Если он дрейфует, все его появления за исключением, одного соответствуют условным позициям символа.

В любом случае эти символы шаблона, вместе со знаком числового значения, определяют выходной символ, который занимает одну позицию при выводе. Эти выходные символы показаны в *Таблице 11-3*.

Таблица 11‑3 Выходные символы шаблона

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Знак | Статические/Дрейфующие символы | | | |
|  | S | + | - | $ |
| положительный | + | + | ⌴ | $ |
| отрицательный | - | ⌴ | - | $ |

Если символ шаблона статичен, выходной символ появляется в соответствующей позиции вывода.

Если символ шаблона дрейфующий, то выходной символ появляется ровно на одну позицию слева от первой ненулевой цифры, над которой символ шаблона дрейфует, или в последней позиции, над которыми он дрейфует. Во всех остальных случаях дрейфующий символ заменяются пробелами, что соответствует подавлению нулевой цифры в числовом значении.

**Символы \* и Z**

Символы \* и Z называют условными цифровыми символами шаблона или символами подавления нулей. Каждый такой символ в спецификации шаблона связан с цифрой в числовом значении.

Если цифра - ноль, выходной символ \* или пробел. Если цифра ненулевая, вывод - символ цифры.

**Символы 'B', '/', '.', ':' и ','**

Символы шаблона 'B', '/', '.', ':' и ',' называются символами вставки. 'B' - вставляет символ пробела. Символ ':' не является символом вставки, определенным в стандарте ANSI, но добавлен в PL/I, для вывода на экран числовых данных, которые представляют время.

PL/I выводит символы вставки в соответствующей позиции вывода, если знак вставки находится не в поле дрейфующего символа или символа подавления нулей. Если знак вставки находится в поле дрейфующего символа или символа подавления нулей, который вызывает подавление цифр, то PL/I подавляет символ вставки после выполнения предыдущих правил.

**Примечание**: в некоторых реализациях PL/I, 'B' - безусловный символ вставки, который всегда выводит пробел в соответствующей позиции вывода. Согласно стандарта ANSI, такой пробел в выводе может быть перезаписан дрейфующим символом или символом подавления нулей \*.

**Символ 9**

Символ шаблона 9 определяет, что соответствующая цифра в числовом значении появляется в соответствующей позиции вывода. Таким образом, 9 - безусловная позиция символа.

**Символ V**

Символ V устанавливает соответствие между цифрами в числовом значении и числовыми значениями позиций символа в спецификации шаблона. Этот символ определяет только позицию, где заканчиваются цифры целой части и начинаются цифры дробной части. Таким образом, символ V определяет выравнивание спецификации шаблона для числового значения.

Если символ V опущен, PL/I предполагает, что все позиции символов, подразумеваемые спецификацией шаблона, относятся к позициям символов целой части. Любые дробные цифры в числовом значении не появляются в результате.

**Примечание**: символ V шаблона - единственный символ, который не соответствует символьной позиции в результате. Таким образом, длина получающейся строки равняется длине спецификации шаблона, без учета V, то есть на один символ меньше, если V присутствует.

Символ V также влияет на подавление символов. PL/I никогда не подавляет дробные цифры, если он не подавляет все цифры.

За символом V PL/I выключает подавление, если оно включено. В результате PL/I не подавляет символы вставки, появляющиеся после символа V в шаблоне, такие как десятичная точка, если он не подавляет все.

**Символы CR и DB**

Символьные пары CR и DB, представляют кредит и дебет. Они действуют как символы знака. Если в спецификации шаблона появляется любой из них, и если знак числового значения отрицательный, то указанная пара появляется в результате. Если числовое значение положительное, то позиции, соответствующие этим символьным парам заменяются двумя пробелами.

**Правила по умолчанию**

Если числовое значение является нулем и если спецификация шаблона не содержит символов 9, то получающийся вывод - все \*, если символ \* присутствует везде. Иначе, выводятся - все пробелы. Это правило имеет приоритет над другими правилами.

Если знак числового значения отрицателен, и если спецификация шаблона не содержит ни одного из символов S, CR или DB, то PL/I устанавливает состояние ошибки преобразования ERROR(1).

Каждая спецификация шаблона подразумевает точность и масштабный коэффициент для числового значения в результате согласно следующим правилам:

* Символы вставки и символьные пары CR и DB не оказывают никакого влияния на точность и масштабный коэффициент.
* Точность результата на единицу меньше, количества статических (дрейфующих) символов; или число символов подавления нулей, плюс число из символов 9.
* Масштабный коэффициент результата равен нулю, если не присутствует V.
* Если V присутствует, масштабный коэффициент результата равняется числу дрейфующих символов, числу символов подавления нулей или числу символов 9, расположенных после символа V.

Примеры, показанные в *Таблицах 11-4* и *11-5*, иллюстрируют некоторые правила, включающие использование элементов формата данных шаблона.

Таблица 11‑4 Вывод отредактированный шаблоном

|  |  |  |
| --- | --- | --- |
| **Значение** | **Спецификация** | **Результат вывода** |
|  |  |  |
| 0.00 | BB$\*\*\*,\*\*\*V.99BB | $\*\*\*\*\*\*\*.00 |
| 0.01 | BB$\*\*\*,\*\*\*V.99BB | $\*\*\*\*\*\*\*.0l |
| 0.25 | BB$\*\*\*,\*\*\*V.99BB | $\*\*\*\*\*\*\*.25 |
| 1.50 | BB$\*\*\*,\*\*\*V.99BB | $\*\*\*\*\*\*1.50 |
| 12.34 | BB$\*\*\*,\*\*\*V.99BB | $\*\*\*\*\*12.34 |
| 123.45 | BB$\*\*\*,\*\*\*V.99BB | $\*\*\*\*123.45 |
| 1234.56 | BB$\*\*\*,\*\*\*V.99BB | $\*\*1,234.56 |
| 12345.67 | BB$\*\*\*,\*\*\*V.99BB | $\*12,345.67 |
| 123456.78 | BB$\*\*\*,\*\*\*V.99BB | $123,456.78 |
|  |  |  |
| 0.00 | $$$$$B$$$V.99 | $.00 |
| 0.01 | $$$$$B$$$V.99 | $.01 |
| 0.25 | $$$$$B$$$V.99 | $.25 |
| 1.50 | $$$$$B$$$V.99 | $1.50 |
| 12.34 | $$$$$B$$$V.99 | $12.34 |
| 123.45 | $$$$$B$$$V.99 | $123.45 |
| 1234.56 | $$$$$B$$$V.99 | $1 234.56 |
| 12345.67 | $$$$$B$$$V.99 | $12 345.67 |
| 123456.78 | $$$$$B$$$V.99 | 1$123 456.78 |

Таблица 11-4 Вывод отредактированный шаблоном (продолжение)

|  |  |  |
| --- | --- | --- |
| **Значение** | **Спецификация** | **Результат вывода** |
|  |  |  |
| 0.00 | 99/99/99 | 00/00/00 |
| 0.01 | 99/99/99 | 00/00/00 |
| 0.25 | 99/99/99 | 00/00/00 |
| 1.50 | 99/99/99 | 00/00/02 |
| 12.34 | 99/99/99 | 00/00/12 |
| 123.45 | 99/99/99 | 00/01/23 |
| 1234.56 | 99/99/99 | 00/12/35 |
| 12345.67 | 99/99/99 | 01/23/46 |
| 123456.78 | 99/99/99 | 12/34/57 |
|  |  |  |
| 0.00 | \*\*:\*\*:\*\* | \*\*\*\*\*\*\*\* |
| 0.01 | \*\*:\*\*:\*\* | \*\*\*\*\*\*\*\* |
| 0.25 | \*\*:\*\*:\*\* | \*\*\*\*\*\*\*\* |
| 1.50 | \*\*:\*\*:\*\* | \*\*\*\*\*\*\*2 |
| 12.34 | \*\*:\*\*:\*\* | \*\*\*\*\*\*12 |
| 123.45 | \*\*:\*\*:\*\* | \*\*\*\*1:23 |
| 1234.56 | \*\*:\*\*:\*\* | \*\*\*12:35 |
| 12345.67 | \*\*:\*\*:\*\* | \*1:23:46 |
| 123456.78 | \*\*:\*\*:\*\* | 12:34:57 |
|  |  |  |
| 0.00 | /++++,+++.V++/ |  |
| 0.01 | /++++,+++.V++/ | / +01/ |
| 0.25 | /++++,+++.V++/ | / +25/ |
| 1.50 | /++++,+++.V++/ | / +1.50/ |
| 12.34 | /++++,+++.V++/ | / +12.34/ |
| 123.45 | /++++,+++.V++/ | / +123.45/ |
| 1234.56 | /++++,+++.V++/ | / +1,234.56/ |
| 12345.67 | /++++,+++.V++/ | / +12,345.67/ |
| 123456.78 | /++++,+++.V++/ | /+123,456.78/ |

Таблица 11‑5 Вывод отредактированный шаблоном

|  |  |  |
| --- | --- | --- |
| **Значение** | **Спецификация** | **Результат вывода** |
| 0.00 | S\*\*\*B\*\*\*.V\*\* | \*\*\*\*\*\*\*\*\*\*\* |
| -0.01 | S\*\*\*B\*\*\*.V\*\* | -\*\*\*\*\*\*\*\*01 |
| 0.25 | S\*\*\*B\*\*\*.V\*\* | +\*\*\*\*\*\*\*\*25 |
| -1.50 | S\*\*\*B\*\*\*.V\*\* | -\*\*\*\*\*\*1.50 |
| 12.34 | S\*\*\*B\*\*\*.V\*\* | +\*\*\*\*\*12.34 |
| -123.45 | S\*\*\*B\*\*\*.V\*\* | -\*\*\*\*123.45 |
| 1234.56 | S\*\*\*B\*\*\*.V\*\* | +\*\*l 234.56 |
| -12345.67 | S\*\*\*B\*\*\*.V\*\* | -\*12 345.67 |
| 123456.78 | S\*\*\*B\*\*\*.V\*\* | +123 456.78 |
|  |  |  |
| 0.00 | $SSSSBSSSV.SS |  |
| -0.01 | $SSSSBSSSV.SS | $ -.01 |
| 0.25 | $SSSSBSSSV.SS | $ +.25 |
| -1.50 | $SSSSBSSSV.SS | $ -1.50 |
| 12.34 | $SSSSBSSSV.SS | $ +12.34 |
| -123.45 | $SSSSBSSSV.SS | $ -123.45 |
| 1234.56 | $SSSSBSSSV.SS | $ +1 234.56 |
| -12345.67 | $SSSSBSSSV.SS | $ -12 345.67 |
| 123456.78 | $SSSSBSSSV.SS | $+123 456.78 |

Таблица 11-5 Вывод отредактированный шаблоном (продолжение)

|  |  |  |
| --- | --- | --- |
| **Значение** | **Спецификация** | **Результат вывода** |
|  |  |  |
| 0.00 | \*\*\*.\*\*\*S | \*\*\*\*\*\*\*\* |
| -0.01 | \*\*\*.\*\*\*S | \*\*\*\*\*\*\*- |
| 0.25 | \*\*\*.\*\*\*S | \*\*\*\*\*\*\*+ |
| -1.50 | \*\*\*.\*\*\*S | \*\*\*\*\*\*2- |
| 12.34 | \*\*\*.\*\*\*S | \*\*\*\*\*12+ |
| -123.45 | \*\*\*.\*\*\*S | \*\*\*\*123- |
| 1234.56 | \*\*\*.\*\*\*S | \*\*1.235+ |
| -12345.67 | \*\*\*.\*\*\*S | \*12.346- |
| 123456.78 | \*\*\*.\*\*\*S | 123.457+ |
|  |  |  |
| 0.00 | $\*\*\*,\*\*\*V\*\*CR | \*\*\*\*\*\*\*\*\*\*\*\* |
| -0.01 | $\*\*\*,\*\*\*V\*\*CR | $\*\*\*\*\*\*\*O1CR |
| 0.25 | $\*\*\*,\*\*\*V\*\*CR | $\*\*\*\*\*\*\*25 |
| -1.50 | $\*\*\*,\*\*\*V\*\*CR | $\*\*\*\*\*\*150CR |
| 12.34 | $\*\*\*,\*\*\*V\*\*CR | $\*\*\*\*\*1234 |
| -123.45 | $\*\*\*,\*\*\*V\*\*CR | $\*\*\*\*12345CR |
| 1234.56 | $\*\*\*,\*\*\*V\*\*CR | $\*\*1,23456 |
| -12345.67 | $\*\*\*,\*\*\*V\*\*CR | $\*12,34567CR |
| 123456.78 | $\*\*\*,\*\*\*V\*\*CR | $123,45678 |
|  |  |  |
| 0.00 | /++++,+++.V++/ |  |
| -0.01 | /++++,+++.V++/ | / 01/ |
| 0.25 | /++++,+++.V++/ | / +25/ |
| -1.50 | /++++,+++.V++/ | / 1.50/ |
| 12.34 | /++++,+++.V++/ | / +12.34/ |
| -123.45 | /++++,+++.V++/ | / 123.45/ |
| 1234.56 | /++++,+++.V++/ | / +1,234.56/ |
| -12345.67 | /++++,+++.V++/ | / 12,345.67/ |
| 123456.78 | /++++,+++.V++/ | /+123,456.78/ |

### Оператор GET EDIT

Оператор GET EDIT считывает данные, используя список формата. Оператор GET EDIT имеет формат:

**GET [FILE(file\_id)] [SKIP[(nl)]]**

**EDIT(входной\_список)(список\_форматов);**

Опции FILE или SKIP можно определять в любом порядке. EDIT должна быть последней. Если опция FILE не определена, PL/I использует файл (SYSIN).

Оператор GET EDIT считывает элементы данных из входного потока в переменные, заданные во входном списке, пока не будет исчерпан список ввода или достигнут конец файла. Оператор GET EDIT соединяет каждый элемент входного списка со следующим последовательным элементом списка формата, применяя элементы формата управления, по мере встречи с ними в процессе обработки. Если оператор GET EDIT исчерпывает входной список, прежде чем достигает конца списка форматов, остающиеся элементы формата игнорируются. Если оператор GET EDIT исчерпывает список формата, прежде чем достижения конец входного списка, список форматов будет повторно обработан с начала.

### Оператор PUT EDIT

Оператор PUT EDIT записывает элементы выходных данных согласно списка форматов. Оператор PUT EDIT имеет формат:

**PUT [FILE(file\_id)] [SKIP[(nl)]] [PAGE]**

**EDIT(выходной\_список)(список\_форматов);**

Опции FILE, SKIP или PAGE можно определять в любом порядке. EDIT должна быть последней. Если не определена опция FILE, PL/I назначает файл (SYSPRINT).

Оператор PUT EDIT выводят выражения из выходного списка в соответствии с элементами формата из списка формата. Оператор PUT EDIT также применяет любые элементы формата управления, с которыми встречаются во время этого процесса. Оператор PUT EDIT игнорирует необработанные элементы формата в конце оператора. Если оператор PUT EDIT встречается с концом выходного списка во время обработки, он перезапускает список форматов с начала.

**Конец раздела 11**

# Записеориентированный ввод-вывод

Записеориентированные файлы содержат двоичные данные, которые PL/I передает к или из внешнего устройства без преобразования. Существует два вида записеориентированных файлов:

* Последовательные (SEQUENTIAL), с которыми PL/I получает доступ к записям в порядке их появления в файле.
* Прямые (DIRECT), с которыми PL/I получает доступ к записям в произвольном порядке с помощью ключей.

В последующих описаниях операторов записеориентированного ввода-вывода: file\_id - файловая переменная или файловая константа. x - скаляр или подключенный тип агрегированных данных, который не имеет атрибутов CHARACTER VARYING, и k - значение ключа или переменная FIXED BINARY.

## Оператор READ

Оператор READ читает записеориентированные файлы фиксированной или переменной длины. Оператор READ имеет формат:

**READ FILE(file\_id) INTO(x);**

Если для открытия файла не используется оператор OPEN, оператор READ выполняет неявное открытие с атрибутами RECORD, SEQUENTIAL и INPUT.

Оператор READ читает количество байтов, определенное длиной x. Если файл открывается с опцией ENVIRONMENT, определяющей размер записи фиксированной длины, оператор READ читает объем данных согласно заявленному размеру записи. Если длина x не соответствует заявленному размеру записи, оператор READ или дополняет до x нулевыми битами или усекает его справа.

## Оператор READ с KEY

Оператор READ с опцией KEY напрямую обращается к отдельным записям в файле. Оператор READ с KEY имеет формат:

**READ FILE(file\_id) INTO(x) KEY(k);**

где k - выражение FIXED BINARY, которое определяет относительную запись для доступа. Значения ключа начинается с нуля и продолжается до тех пор, пока значение ключа, умноженное на фиксированную длину записи, не достигнет емкости диска.

Если не используется оператор OPEN для открытия файла, оператор READ с KEY выполняет неявное открытие с атрибутами RECORD, INPUT, DIRECT и KEYED. PL/I не позволяет оператору READ с KEY доступ к записям переменной длины.

## Оператор READ с KEYTO

Оператор READ с опцией KEYTO извлекает значения ключей из входного файла во время последовательного доступа. Программа может сохранить значения ключей в памяти или в другом файле, и впоследствии выполнить прямой доступ к записям входного файла, используя значения ключей.

Оператор READ с KEYTO имеет формат:

**READ FILE(file\_id) INTO(x) KEYTO(k);**

где k - переменная FIXED BINARY, присвоенная относительному номеру записи, предназначенной для чтения.

Если для открытия файла не используется, оператор OPEN и оператор READ с KEYTO выполняет неявное открытие с атрибутами RECORD, INPUT, SEQUENTIAL и KEYED.

## Оператор WRITE

Оператор WRITE записывает данные из памяти во внешний набор данных без преобразования. Оператора WRITE имеет формат:

**WRITE FILE(file\_id) FROM(x);**

Если оператор OPEN не используется для открытия файла, оператор WRITE выполняет неявное открытое с атрибутами RECORD, OUTPUT и SEQUENTIAL.

Выходной размер записи точно соответствует длине x. Если файл открывается с опцией ENVIRONMENT, определяющей размер записи фиксированной длины, оператор WRITE записывает объем данных согласно объявленному размеру записи. Если длина x не соответствует объявленному размеру записи, оператор WRITE или дополняет до x нулевыми битами или усекает ее справа.

## Оператор WRITE с KEYFROM

Оператор WRITE с опцией KEYFROM непосредственно получает доступ к файлу для вывода. Оператор WRITE с KEYFROM имеет формат:

**WRITE FILE(file\_id) FROM(x) KEYFROM(k);**

где k обозначает выражение FIXED BINARY, приводящее к значению ключа, которое PL/I обрабатывает как READ с опцией KEY, описанным в [*Разделе 12.2*](#_Оператор_READ_с).

Если не используется оператор OPEN для открытия файла, оператор WRITE с KEYFROM выполняет неявное открытие с атрибутами RECORD, DIRECT, OUTPUT и KEYED.

**Конец раздела 12**

# Встроенные функции

Встроенная функция - вычислительная подпрограмма, обеспеченная как часть библиотеки подпрограмм времени выполнения PL/I. Можно использовать ссылку на встроенную функцию в качестве ссылки на функцию, определяемую пользователем.

Не следует объявлять имя встроенной функции. Если имя встроенной функции в программе объявлено повторно, на него нельзя сослаться как на встроенную функцию в рамках этого объявления. Однако, можно использовать встроенную функцию во вложенном блоке, повторно объявляя имя с атрибутом BUILTIN.

Встроенных функций PL/I подразделяются на следующие категории:

|  |  |
| --- | --- |
| * арифметические; * математические; * обработки строки; | * преобразования; * обработки состояний; * вспомогательные. |

## Арифметические функции

Арифметическими функциями являются:

ABS FLOOR MOD TRUNC

CEIL MAX ROUND

DIVIDE MIN SIGN

Арифметические встроенные функции возвращают информацию об атрибутах указанных арифметических значений и выполняют общие арифметические вычисления.

## Математические функции

К математическим функциям относятся:

ACOS COS LOG SIND TAND

ASIN COSD LOG2 SINH TANH

ATAN COSH LOG10 SQRT

ATAND EXP SIN TAN

Встроенные математические функции выполняют математические вычисления в арифметике с плавающей точкой. Математические функции включают:

* часто используемые тригонометрические функции и их инверсии;
* логарифмические функции с основанием 2 (двоичным), с основанием e (натуральным) и с основанием 10 (десятичным);
* возведение числа «e» в степень;
* функции гиперболического синуса и косинуса;
* функцию извлечения квадратного корня.

Каждая из этих функций принимает единственный аргумент FLOAT BINARY и возвращает результат FLOAT BINARY. Точность результата зависит от точности параметра. Если аргумент имеет одинарную точность, результат будет иметь одинарную точность. Если аргумент - двойной точности, результат - двойной точности. Если параметр - выражение, содержащее операнды различной точности, PL/I сначала выполняет преобразование согласно правилам, изложенным в [*Разделе 4.2*](#_Функции_арифметических_преобразований).

Также можно определить аргумент, который не является FLOAT BINARY, но PL/I автоматически преобразует его.

Все подпрограммы функций, за исключением SQRT, используют алгоритмы, основанные на аппроксимациях полиномами Чебышева. Подпрограмма функции SQRT основывается на методе Ньютона.

Как правило, эти алгоритмы масштабируют заданный аргумент в конечный интервал, обычно, -1 ≤ X ≤ 1, и затем вычисляют приближение Чебышева с помощью соответствующего, используя соответствующее рекуррентное соотношение. Наибольшим источником ошибок в этих подпрограммах является результат усечения значимых цифр в процессе масштабирования. За исключением этого, подпрограммы имеют среднюю точность 7 значащих десятичных цифр для одинарной точности, 15 цифр для двойной точности.

## Функции обработки строки

Функции обработки строк следующие:

|  |  |  |  |
| --- | --- | --- | --- |
| * BOOL * COLLATE * COPY | * INDEX * LENGTH * REVERSE | * SEARCH * SUBSTR * TRANSLATE | * TRIM * VERIFY |

Встроенные функции обработки строк выполняют операции с символьными и битовыми строками.

## Функции преобразования строк

Функции преобразования строк следующие:

|  |  |  |
| --- | --- | --- |
| * ASCII * BINARY * BIT | * CHARACTER * DECIMAL * FIXED | * FLOAT * RANK * UNSPEC |

Встроенные функции преобразования конвертируют данные из одного типа в другой. PL/I внутренне использует эти функции для выполнения автоматических (неявных) преобразований.

## Функции обработки состояний

Функции обработки состояний следующие:

* ONCODE
* ONFILE
* ONKEY

Встроенные функции, обрабатывающие состояния, возвращают информацию о состояниях, сообщенных системой времени выполнения. Эти функции не имеют параметров и возвращают значение только, когда выполняется в ON-блоке. ON-блок может быть активизирован, когда указанное состояние вызвано программно, или в результате прерывания, вызванного возникновением указанного состояния.

## Вспомогательные функции

К вспомогательным относятся следующие функции:

|  |  |  |  |
| --- | --- | --- | --- |
| * ADDR * DATE * DIMENSION | * HBOUND * LBOUND * LINENO | * LOCK * NULL * PAGENO | * TIME * UNLOCK |

Вспомогательные встроенные функции возвращают информацию о базированных переменных, дате и времени, текущем номере строки и номере страницы в файле, информацию о размерностях массива, и обеспечивают возможность заблокировать и разблокировать отдельные записи в файле.

## Список встроенных функций

Данный раздел описывает конкретный формат, атрибуты параметров, результат и свойства каждой встроенной функции.

|  |  |
| --- | --- |
| **ABS** |  |
| Категория: | Арифметическая |
| Формат: | ABS(X) |
| Параметры: | X может быть любым арифметическим выражением. |
| Результат: | Возвращает абсолютное значение X. |
| Алгоритм: | Если X ≥ 0 то возвращается X, иначе возвращается -X. |
| Тип результата: | Тот же, что у X. |
| Примеры: | ABS(-100) возвращает 100  ABS(18.78) возвращает 18.78 |
| **ACOS** |  |
| Категория: | Математическая |
| Формат: | ACOS(X) |
| Параметры: | X является арифметическим выражением, -1 ≤ X ≤ 1. |
| Результат: | Возвращает арккосинус X. Например, ACOS(X) является углом в радианах, косинус которого равен Х в диапазоне 0 ≤ ACOS(X) ≤ π. |
| Тип результата: | FLOAT BINARY |
| Алгоритм: | ACOS(X) равняется PI/2 - ASIN(X) |
| Состояние ошибки | Если X не находится в интервале -1 ≤ X ≤ 1, система времени выполнения устанавливает состояние ERROR(3). |
| Примеры: | ACOS(0.866) возвращает 5.236490E-01  ACOS(0.86603) возвращает 5.235897302627563E-001 |
| **ADDR** |  |
| Категория: | Вспомогательная |
| Формат: | ADDR(X) |
| Параметры: | X является переменной. |
| Результат: | Возвращает указатель, который идентифицирует место хранения переменной X. |
| Тип результата: | POINTER |
| **ASCII** |  |
| Категория: | Преобразование |
| Формат: | ASCII(I) |
| Параметры: | I является выражением FIXED BINARY. |
| Результат: | Возвращает один символ, положение которого в упорядоченной последовательности ASCII соответствует I (см *Приложение 6* с кодами ASCII).. |
| Тип результата: | CHARACTER(I) |
| Алгоритм: | ASCII(I) равняется SUBSTR(COLLATE(),MOD(I,128)+1,1) |
| Примечание | ASCII(I) является функцией обратной RANK(I), т.е. ASCII(Rank(C))=C для любого символа C. |
| Примеры: | ASCII(88) возвращает 'X'  ASCII(40) возвращает '(' |
| **ASIN** |  |
| Категория: | Математическая |
| Формат: | ASIN(X) |
| Параметры: | X является арифметическим выражением, -1 ≤ X ≤ 1. |
| Результат: | Возвращает арксинус X. Например, ASIN(X) является углом в радианах, синус которого равен Х в диапазоне π/2 ≤ ASIN(X) ≤ π/2. |
| Тип результата: | FLOAT BINARY |
| Алгоритм: | Аппроксимация с помощью полиномов Чебышева |
| Состояние ошибки | Если X не находится в интервале -1 ≤ X ≤ 1, система времени выполнения устанавливает состояние ERROR(3). |
| Примеры: | ASIN(0.866) возвращает 1.0471462E+00  ASIN(0.86603) возвращает 1.047206282615661E+000 |
| **ATAN** |  |
| Категория: | Математическая |
| Формат: | ATAN(X) |
| Параметры: | X является арифметическим выражением. |
| Результат: | Возвращает арктангенс X. Например, ATAN(X) является углом в радианах, тангенс которого равен Х в диапазоне -π/2 ≤ ATAN(X) ≤ π/2. |
| Тип результата: | FLOAT BINARY |
| Алгоритм: | Аппроксимация с помощью полиномов Чебышева |
| Примеры: | ATAN(0.577) возвращает 5.2333600E-01  ATAN(0.57735) возвращает 5.235985517501830E-001 |
| **ATAND** |  |
| Категория: | Математическая |
| Формат: | ATAND(X) |
| Параметры: | X является арифметическим выражением. |
| Результат: | Возвращает арктангенс X в градусах. Например, угол в градусах, тангенс которого равен Х в диапазоне -90° ≤ ATAND(X) ≤ 90°. |
| Тип результата: | FLOAT BINARY |
| Алгоритм: | ATAND(X) равняется 180/π \* ATAN(X) |
| Примеры: | ATAN(0.577) возвращает 2.9984940E+01  ATAN(0.57735) возвращает 2.999998664855957E+001 |
| **BINARY** |  |
| Категория: | Преобразования |
| Формат: | BINARY (X[,p]) |
| Параметры: | X арифметическое или строковое выражение, которое может быть преобразовано в арифметическое значение. Если X DECIMAL с ненулевым масштабным коэффициентом, то должна быть задана целочисленная константа p, определяющая точность результата. |
| Результат: | Арифметическое значение BINARY, эквивалентное X. |
| Тип результата: | Если X FLOAT BINARY – возвращает FLOAT BINARY, иначе FIXED BINARY |
| Примеры: | Если X = 12.675 FIXED DECIMAL(6,3) то  BINARY(X,15) возвращает 12  BINARY(12.675,15) возвращает 1.2000000E+01 |
| **BIT** |  |
| Категория: | Преобразование |
| Формат: | BIT(S[,L]) |
| Параметры: | S - арифметическое или строковое выражение. L - положительное выражение FIXED BINARY. |
| Результат: | Преобразует S в битовую строку длиной L, если L определен. В противном случае, преобразует S в битовую строку, длина которой определяется правилами преобразования в [*Разделе 4.3.3*](#_Преобразование_битовой_строки). |
| Тип результата: | BIT |
| Примеры: | BIT(3,8) возвращает 00000110  BIT(-4,16) возвращает 0000100000000000 |
| **BOOL** |  |
| Категория: | Строковая |
| Формат: | BOOL(X,Y,Z) |
| Параметры: | X,Y - битовые выражения.  Z - константа битовой строки, длиной четыре бита. |
| Результат: | Возвращает логическую функцию над X и Y, определенную константой битовой строки Z следующим образом. Пусть Z1, Z2, Z3, Z4 -битовые значения в Z, читая слева направо. Тогда битовые значения A, B и четырехразрядная строка Z определяют логическую функцию BOOL(A,B,Z):  A B BOOL(A,B,Z)  0 0 Z1  0 1 Z2  1 0 Z3  1 1 Z4  Затем вызывается функция BOOL(X,Y) с битовыми строками X и Y следующим образом. Если X и Y имеют не подходящую длину, более короткая строка дополняется справа нулевыми битами, пока они не станут одинаковой длины. После этого BOOL(X,Y,Z) определяется, как битовая строка, энный бит которой получен из предыдущей таблицы, позволив A быть энным битом X и B - энным битом Y. |
| Тип результата: | BIT(n), где n равняется MAX(LENGTH(X), LENGTH(Y)). |
| Примеры: | BOOL('0011'B,'0101'B,'1001'B) возвращает '1001'B  BOOL('01011'B,'11'B, '1001'B) возвращает '01100'B |
| **CEIL** |  |
| Категория: | Арифметическая |
| Формат: | CEIL(X) |
| Параметры: | X - арифметическое выражение. |
| Результат: | Возвращает наименьшее целое ≥ X. |
| Алгоритм: | -FLOOR(-X) |
| Тип результата: | Целочисленное значение того же типа как и X. |
| Примеры: | CEIL(7.9) возвращает 8, CEIL((5/3)) возвращает 2 |
| **CHARACTER** |  |
| Категория: | Преобразование |
| Формат: | CHARACTER(S[,L]) |
| Параметры: | S - арифметическое или строковое выражение. L положительное выражение FIXED BINARY. |
| Результат: | S преобразуется в символьную строку длиной L, когда L определена. В противном случае, S преобразуется в символьную строку, длина которой определяется правилами преобразования в [*Разделе 4*](#_Преобразования_данных_2). |
| Тип результата: | CHARACTER |
| Примеры: | Если X = -13.25 то  CHARACTER(X,10) возвращает '⌴⌴⌴⌴-13.25'  CHARACTER(2\*(3+7)-6,10) возвращает '⌴⌴⌴⌴⌴⌴⌴⌴14' |
| **COLLATE** |  |
| Категория: | Строковая |
| Формат: | COLLATE() |
| Параметры: | Отсутствуют |
| Результат: | Возвращает символьную строку из 128 символов ASCII в порядке возрастания, начиная с 0-го. (Набор символов ASCII приведен в *Приложении 3*) |
| Тип результата: | CHARACTER(128) |
| Примечание: | В PL/I-86 V1.2, COLLATE() возвращает символьную строку длиной 256. |
| **COPY** |  |
| Категория: | Строковая |
| Формат: | COPY(S,I) |
| Параметры: | S - выражение символьной строки  I - выражение FIXED BINARY |
| Результат: | Возвращает I копий S соединенных вместе.  Если I ≤ 0, COPY возвращает пустую (null) строку. |
| Тип результата: | CHARACTER |
| Примеры: | COPY('\*', 80) возвращает значение CHARACTER(80), содержащее 80 символов звездочка. |
| **COS** |  |
| Категория: | Математическая |
| Формат: | COS(X) |
| Параметры: | X является арифметическим выражением. |
| Результат: | Возвращает косинус X в радианах. |
| Тип результата: | FLOAT BINARY |
| Алгоритм: | Аппроксимация с помощью полиномов Чебышева |
| Примеры: | COS(3.1415/3.0) возвращает 5.000267465490945E-001 |

|  |  |
| --- | --- |
| **COSD** |  |
| Категория: | Математическая |
| Формат: | COSD(X) |
| Параметры: | X является арифметическим выражением. |
| Результат: | Возвращает косинус X в градусах. |
| Тип результата: | FLOAT BINARY |
| Алгоритм: | COSD(X) равняется COS(X\*π/180) |
| Примеры: | COSD(0.500) возвращает 9.9996180E-01  COSD(0.50000) возвращает 9.999617934226980E-001 |
| **COSH** |  |
| Категория: | Математическая |
| Формат: | COSH(X) |
| Параметры: | X является арифметическим выражением. |
| Результат: | Возвращает гиперболический косинус X. |
| Тип результата: | FLOAT BINARY |
| Алгоритм: | COSH(X) равняется (EXP(X) + EXP(-X))/2 |
| Примеры: | COSH(2.75) возвращает 7.8532790E+00  COSH(2.75000) возвращает 7.853279590606689E+000 |
| **DATE** |  |
| Категория: | Вспомогательная |
| Формат: | DATE() |
| Параметры: | Отсутствуют |
| Результат: | Возвращает символьную строку, представляющую дату в форме YYMMDD, где  YY - текущий год (00-99)  MM – текущий месяц года (00-12)  DD - текущий день месяца (00-31) |
| Тип результата: | CHARACTER(6) |
| Примеры: | DATE() возвращает '830325' |
| Примечание: | Функция доступна только, если поддерживается операционной системой. Если не поддерживается, DATE возвращает пробелы. |
| **DECIMAL** |  |
| Категория: | Преобразование |
| Формат: | DECIMAL(X[,p[,q]]) |
| Параметры: | X арифметическое или строковое выражение, которое может быть преобразовано в арифметическое значение.  p - целочисленная константа, 1 ≤ p ≤ 15.  q - целочисленная константа, 1 ≤ q ≤ p. |
| Результат: | Преобразует X в значение DECIMAL. p и q необязательные, но когда определены представляют точность и масштабный коэффициент, соответственно. Если задан только p, q предполагается равным нулю. Если p и q не заданы, то точность и масштабный коэффициент результата определяются правилами преобразования, приведенными в [*Разделе 4.3.2*](#_Преобразование_арифметического_типа_2). |
| Тип результата: | FIXED DECIMAL |
| Примеры: | DECIMAL(125,6,2) возвращает 125.00 |

|  |  |
| --- | --- |
| **DIMENSION** |  |
| Категория: | Вспомогательная |
| Формат: | DIMENSION(X,N) | DIM(X,N) |
| Параметры: | X -переменная массив. N - положительное целое выражение |
| Результат: | Возвращает положительное целое число, представляющее степень энной размерности массива, на который ссылаются X. |
| Тип результата: | FIXED BINARY |
| **DIVIDE** |  |
| Категория: | Арифметическая |
| Формат: | DIVIDE(X,Y,p) или DIVIDE(X,Y,p,q) |
| Параметры: | X и Y -арифметические выражения. |
| Результат: | Возвращает частное от деления X на Y с константами точности результата p, и масштабным коэффициентом q. Если q отсутствует, его значение предполагается равным 0. Если X и Y FIXED BINARY, q должен быть опущен или равен нулю. |
| Тип результата: | Общий арифметический тип X и Y. |
| Примеры: | DIVIDE(189.07,37.56,15,5) возвращает 5.03381  DIVIDE(296,49,15) возвращает 6  DIVIDE(233.456e2,1.19e1,24) возвращает 1.9710920E+02 |
| **EXP** |  |
| Категория: | Математическая |
| Формат: | EXP(X) |
| Параметры: | X и Y -арифметические выражения. |
| Результат: | Возвращает значение e в степени X, где e - основание натурального логарифма. |
| Тип результата: | FLOAT BINARY |
| Алгоритм: | Аппроксимация с помощью полиномов Чебышева |
| Примеры: | EXP(5.13) возвращает 1.6901700E+02  EXP(5.13333) возвращает 1.695808563232421E+002 |
| **FIXED** |  |
| Категория: | Преобразование |
| Формат: | FIXED(X[,p[,q]]) |
| Параметры: | X арифметическое или строковое выражение, которое может быть преобразовано в арифметическое значение.  p и q - целочисленные константы. |
| Результат: | Преобразует X в арифметическое значение FIXED. p и q необязательные, но если они определены, то определяют точность и масштабный коэффициент результата. Если задан только p, то предполагается, что q равен 0. Если p и q не заданы, то точность и масштабный коэффициент определяются правилами преобразования в [*Разделе 4*](#_Преобразования_данных_3). |
| Тип результата: | Если X FIXED DECIMAL или CHARACTER, результат FIXED DECIMAL. Иначе, он FIXED BINARY. |
| Примеры: | Если s = '01010010'b, то  FIXED(S,8) возвращает 82  FIXED(s,24) возвращает 8.2000000E+01 |

|  |  |
| --- | --- |
| **FLOAT** |  |
| Категория: | Преобразование |
| Формат: | FLOAT(X[,p]) |
| Параметры: | X арифметическое или строковое выражение, которое может быть преобразовано в арифметическое значение.  p – необязательная целочисленная константа. |
| Результат: | Преобразует X в арифметическое значение FIXED. p необязательная, но если она определена, то определяет точность результата. Если p не задан, то точность определяются правилами преобразования в [*Разделе 4*](#_Преобразования_данных_3). |
| Тип результата: | FIXED BINARY |
| Примеры: | Если y = 4589 FIXED BINARY(15), то  FLOAT(Y,24) возвращает 4.5890000E+03 |
| **FLOOR** |  |
| Категория: | Арифметическая |
| Формат: | FLOOR(X) |
| Параметры: | X арифметическое выражение. |
| Результат: | Вычисляет наибольшее целое число ≤ X |
| Тип результата: | Целочисленное значение того же типа как у X. |
| Примеры: | FLOOR(7.9) возвращает 7  FLOOR((5/3)) возвращает 1 |
| **HBOUND** |  |
| Категория: | Строковая |
| Формат: | HBOUND(X,N) |
| Параметры: | X переменная массива. N - положительная целочисленная константа |
| Результат: | Возвращает верхнюю границу энной размерности переменной типа массив X. |
| Тип результата: | FIXED BINARY |
| **INDEX** |  |
| Категория: | Преобразование |
| Формат: | INDEX(X,Y[,I]) |
| Параметры: | X и Y строковые выражения одного типа, или битовое или символьное. Необязательный третий параметр I является целочисленным выражением. Если заданы только два параметра, то значение третьего параметра по умолчанию - 1. |
| Результат: | Возвращает целочисленное значение, указывающее позицию первого вхождения строки Y в строке X, начиная сканирование от положения I в X. Если X или Y имеет значение NULL или если Y не входит в Х, INDEX возвращает значение ноль. |
| Тип результата: | FIXED BINARY |
| Примеры: | Если s = '01010010'b, то  INDEX('123456789', '7') возвращает 8  INDEX('ABAB', 'AB', 2) возвращает 3 |

|  |  |
| --- | --- |
| **LBOUND** |  |
| Категория: | Вспомогательная |
| Формат: | LBOUND(X,N) |
| Параметры: | X переменная массива. N - положительная целочисленная константа. |
| Результат: | Возвращает нижнюю границу энной размерности массива, на которую ссылается X. |
| Тип результата: | FIXED BINARY |
| **LENGTH** |  |
| Категория: | Строковая |
| Формат: | LENGTH(X) |
| Параметры: | X - строковое выражение, битовое или символьное. |
| Результат: | Возвращает число символов или битов в строке X. Если X имеет атрибут VARYING, LENGTH(X) возвращает текущую длину X. |
| Тип результата: | FIXED BINARY |
| Примеры: | LENGTH('Himalayan') возвращает 9  LENGTH(") возвращает 0 |
| **LINENO** |  |
| Категория: | Вспомогательная |
| Формат: | LINENO(F) |
| Параметры: | F файловая переменная. |
| Результат: | Возвращает текущий номер строки файла, на который ссылается F. Файла должен иметь атрибут PRINT. |
| Тип результата: | FIXED BINARY |
| **LOCK** |  |
| Категория: | Вспомогательная |
| Формат: | LOCK(F,I) |
| Параметры: | F - файловая константа или переменная, определяющая файл, который должен быть открыт в режиме общего доступа. I - целое число FIXED BINARY(15), которое задает номер записи относительно размера записи, определенной в параметре ENVIRONMENT. |
| Результат: | Возвращает бит 1, если операция успешна или бит 0, если неудачна. Функция блокирует запись, определенную I, чтобы другой пользователь не мог заблокировать или получить доступ к ней. Запись остается заблокированной, пока не будет разблокирована функцией UNLOCK, или до завершения программы. |
| Тип результата: | BIT(1) |
| Примечание: | Функция доступна, только если поддерживается операционной системой. |

|  |  |
| --- | --- |
| **LOG** |  |
| Категория: | Математическая |
| Формат: | LOG(X) |
| Параметры: | X является арифметическим выражением, X > 0. |
| Результат: | Возвращает натуральный логарифм X. |
| Тип результата: | FLOAT BINARY |
| Алгоритм: | Аппроксимация с помощью полиномов Чебышева |
| Состояние ошибки | Если X ≤ 0 система времени выполнения устанавливает состояние ERROR(3). |
| Примеры: | LOG(10.0) возвращает 2.3025850E+00  LOG(10.00000) возвращает 2.302585124969482E+000 |
| **LOG2** |  |
| Категория: | Математическая |
| Формат: | LOG2(X) |
| Параметры: | X является арифметическим выражением, X > 0. |
| Результат: | Возвращает двоичный логарифм X. |
| Тип результата: | FLOAT BINARY |
| Алгоритм: | LOG2(X) равняется LOG(X)/LOG(2) |
| Состояние ошибки | Если X ≤ 0 система времени выполнения устанавливает состояние ERROR(3). |
| Примеры: | LOG2(10.0) возвращает 3.3219270E+00  LOG2(10.00000) возвращает 3.321927785873412E+000 |
| **LOG10** |  |
| Категория: | Математическая |
| Формат: | LOG10(X) |
| Параметры: | X является арифметическим выражением, X > 0. |
| Результат: | Возвращает десятичный логарифм X. |
| Тип результата: | FLOAT BINARY |
| Алгоритм: | LOG10(X) равняется LOG(X)/LOG(10) |
| Состояние ошибки | Если X ≤ 0 система времени выполнения устанавливает состояние ERROR(3). |
| Примеры: | LOG10(125.0) возвращает 2.0969100E+00  LOG2(125.00000) возвращает 2.096910013008051E+000 |
| **MAX** |  |
| Категория: | Арифметическая |
| Формат: | MAX(X,Y) |
| Параметры: | X и Y являются арифметическим выражением. |
| Результат: | Функция возвращает наибольший из двух аргументов |
| Алгоритм: | Если X ≥ Y возвращает X, в противном случае Y. |
| Тип результата: | Общий арифметический тип X и Y. |
| Примеры: | MAX(234, 64) возвращает 234  MAX(3.77e5, 9.856e3) возвращает 3.7700E+05 |

|  |  |
| --- | --- |
| **MIN** |  |
| Категория: | Арифметическая |
| Формат: | MIN(X,Y) |
| Параметры: | X и Y являются арифметическим выражением. |
| Результат: | Функция возвращает наименьший из двух аргументов. |
| Алгоритм: | Если X ≤ Y возвращает X, в противном случае Y. |
| Тип результата: | Общий арифметический тип X и Y. |
| Примеры: | MIN(234, 64) возвращает 64  MIN(3.77e5, 9.856e3) возвращает 9.8560E+03 |
| **MOD** |  |
| Категория: | Арифметическая |
| Формат: | MOD(X,Y) |
| Параметры: | X и Y являются арифметическим выражением. |
| Результат: | Функция возвращает остаток от деления X по модулю Y |
| Алгоритм: | Если Y = 0 возвращает X, иначе X-(Y)\*FLOOR(X/Y). |
| Тип результата: | Результатом является значение, имеющее общий арифметический тип X и Y. |
| Примеры: | MOD(7,3) возвращает 1  MOD(-7,3) возвращает 2  MOD(7,-3) возвращает -2  MOD(-7,-3) возвращает -1 |
| Примечание: | если Y=0, MOD(X,Y) всегда возвращает значение с тем же знаком, что и Y, и меньше ABS(Y) по величине. |
| **NULL** |  |
| Категория: | Вспомогательная |
| Формат: | NULL[()] |
| Результат: | Возвращает значение указателя null, которое указывает на недопустимое место хранения. |
| Тип результата: | POINTER |
| **ONCODE** |  |
| Категория: | Состояние |
| Формат: | ONCODE() |
| Результат: | Возвращает значение подкода ошибки последнего возникшего состояния. Состояния ошибок и соответствующие им коды ошибок перечислены в [*Разделе 9.4*](#_Состояние_ERROR) в Таблице 9-1. |
| Тип результата: | FIXED BINARY |
| **ONFILE** |  |
| Категория: | Состояние |
| Формат: | ONFILE() |
| Результат: | Возвращает имя файла, для которого возникли последние состояния ENDFILE или ENDPAGE. |
| Тип результата: | CHARACTER |

|  |  |
| --- | --- |
| **ONKEY** |  |
| Категория: | Состояние |
| Формат: | ONKEY() |
| Результат: | Возвращает значение ключа записи в символьной строке, который вызвал состояние преобразования или ввода-вывода. |
| Тип результата: | CHARACTER |
| **PAGENO** |  |
| Категория: | Вспомогательная |
| Формат: | PAGENO(F) |
| Параметр: | F - значение файла. |
| Результат: | Возвращает номер страницы в файле, определенном F. Файл должен иметь атрибут PRINT. |
| Тип результата: | FIXED BINARY |
| **RANK** |  |
| Категория: | Преобразование |
| Формат: | RANK(C) |
| Параметры: | C - символьное значение одиночной длины. |
| Результат: | Возвращает целочисленное представление ASCII символа C (см. *Приложение 6*). |
| Тип результата: | FIXED BINARY |
| Алгоритм: | RANK(C) равняется INDEX(COLLATE(),C)-1 |
| Примеры: | RANK('Y') возвращает 89  RANK('5') возвращает 53 |
| **REVERSE** |  |
| Категория: | Строковая |
| Формат: | REVERSE(S) |
| Параметры: | S - выражение символьной строки |
| Результат: | Возвращает строку той же длины, что и S, с символами расположенными в обратном порядке. |
| Тип результата: | CHARACTER |
| Примеры: | REVERSE('ABC') возвращает 'CBA' |
| **ROUND** |  |
| Категория: | Арифметическая |
| Формат: | ROUND(X,K) |
| Параметры: | X – арифметическое выражение.  K - целочисленная константа со знаком. |
| Результат: | Возвращает X округленное до K цифр справа от десятичной точки, если K ≥ 0. Возвращает X округленное до -K цифр слева от десятичной точки, если K < 0. |
| Тип результата: | Тот же, что у X |
| Алгоритм: | Возвращает SIGN(X)\*FLOOR(ABS(X)\*B\*\*N)+0.5)/B\*\*N где  B=2, если X - BINARY, B=10, если X – DECIMAL  и N=K, если X - FIXED, иначе N=K-E, если X - FLOAT и E - показатель степени X. |
| Примеры: | ROUND(12345.24689, 3) возвращает 12345.24700  ROUND(34567.12345, -3) возвращает 35000.00000 |
| **SEARCH** |  |
| Категория: | Строковая |
| Формат: | SEARCH(S,C) |
| Параметры: | S и C - выражения символьной строки. |
| Результат: | Возвращает целочисленное значение, указывающее позицию первого символа в S, которая соответствует символу в C. Возвращает 0, если в S отсутствуют символы из C. |
| Тип результата: | FIXED BINARY |
| Примеры: | SEARCH('$\*\*\*12.95', 101234567891) возвращает 5 |
| **SIGN** |  |
| Категория: | Арифметическая |
| Формат: | SIGN(X) |
| Параметры: | X – арифметическое выражение. |
| Результат: | Возвращает -1, 0 или 1 для указания знака X. |
| Тип результата: | FIXED BINARY |
| Алгоритм: | ЕслиX < 0 - возвращает -1  X = 0 - возвращает 0  X > 0 - возвращает 1 |
| Примеры: | SIGN(-76.45e4) возвращает -1  SIGN(199.98) возвращает 1 |
| **SIN** |  |
| Категория: | Математическая |
| Формат: | SIN(X) |
| Параметры: | X – арифметическое выражение. |
| Результат: | Возвращает синус X в радианах. |
| Тип результата: | FLOAT BINARY |
| Алгоритм: | Аппроксимация с помощью полиномов Чебышева |
| Примеры: | SIN(3.1415/6.0) возвращает 4.999866265466036E-001 |
| **SIND** |  |
| Категория: | Математическая |
| Формат: | SIND(X) |
| Параметры: | X – арифметическое выражение. |
| Результат: | Возвращает синус X в градусах. |
| Тип результата: | FLOAT BINARY |
| Алгоритм: | SIND(X) равен SIN(X\*π/180) |
| Примеры: | SIND(0.50) возвращает 8.7265340E-03  SIND(0.50000) возвращает 8.726534433662890E-03 |
| **SINH** |  |
| Категория: | Математическая |
| Формат: | SINH(X) |
| Параметры: | X – арифметическое выражение. |
| Результат: | Возвращает гиперболический синус X. |
| Тип результата: | FLOAT BINARY |
| Алгоритм: | SINH(X) равен (EXP(X)-EXP(-X))/2 |
| Примеры: | SINH(2.75) возвращает 7.7893520E+00  SINH(2.75000) возвращает 7.789351940155029E+000 |
| **SQRT** |  |
| Категория: | Математическая |
| Формат: | SQRT(X) |
| Параметры: | X является арифметическим выражением, X ≥ 0. |
| Результат: | Возвращает квадратный корень из X. |
| Тип результата: | FLOAT BINARY |
| Алгоритм: | Метод Ньютона |
| Состояние ошибки | Если X < 0 система времени выполнения устанавливает состояние ERROR(3). |
| Примеры: | SQRT(2) возвращает 1.4142135E+00  SQRT(2.0000000) возвращает 1.414213562373094E+000 |
| **SUBSTR** |  |
| Категория: | Строковая |
| Формат: | SUBSTR(X,I[,J]) |
| Параметры: | X - битовая или символьная строка.  I и J - значения FIXED BINARY |
| Результат: | Возвращает строку, которая является копией строки X, начиная с элемента I и длиной J. Если J не задан, он принимает значение по умолчанию длину остатка от строки, равную LENGTH(X)-I+1. |
| Тип результата: | Тот же, что у X |
| Состояние ошибки | Не отслеживается. Если параметры вне диапазона, могут произойти непредсказуемые результаты. |
| Примеры: | Если word = 'Digital Research' и s = '01110101'b, то  SUBSTR(word,11) возвращает 'search'  SUBSTR(S,4,2) возвращает '10'b |
| **TAN** |  |
| Категория: | Математическая |
| Формат: | TAN(X) |
| Параметры: | X является арифметическим выражением. |
| Результат: | Возвращает тангенс из X в радианах. |
| Тип результата: | FLOAT BINARY |
| Алгоритм: | TAN(X) = SIN(X)/COS(X) |
| Состояние ошибки | Если COS(X) = 0 система времени выполнения устанавливает состояние ERROR(3). |
| Примеры: | TAN(3.1415/4-0) возвращает 9.999536742781560E-001 |
| **TAND** |  |
| Категория: | Математическая |
| Формат: | TAND(X) |
| Параметры: | X является арифметическим выражением. |
| Результат: | Возвращает тангенс из X в градусах. |
| Тип результата: | FLOAT BINARY |
| Алгоритм: | TAND(X) равняется TAN(X\*π/180) |
| Состояние ошибки | Если COS(X\*π/180) = 0 система времени выполнения устанавливает состояние ERROR(3). |
| Примеры: | TAND(0.50000) возвращает 8.72686747144603E-003 |
| **TANH** |  |
| Категория: | Математическая |
| Формат: | TANH(X) |
| Параметры: | X является арифметическим выражением. |
| Результат: | Возвращает гиперболический тангенс из X. |
| Тип результата: | FLOAT BINARY |
| Алгоритм: | TANH(X) = (EXP(X)-EXP(-X))/(EXP(X)+EXP(-X)) |
| Примеры: | TANH(2.75) возвращает 9.9185970E-01  TANH(2.75000) возвращает 9.918597340583800E-001 |
| **TIME** |  |
| Категория: | Вспомогательная |
| Формат: | TIME() |
| Параметры: | Отсутствуют |
| Результат: | Возвращает символьную строку, представляющую время в форме HHMMSStttttt, где  HH - текущий час (00-23)  MM – текущие минуты (00-59)  SS - текущие секунды (00-59)  tttttt - милисекунды |
| Тип результата: | CHARACTER(12) |
| Примеры: | TIME() возвращает '133427000000' |
| Примечание: | Функция доступна только если поддерживается операционной системой. tttttt имеют точность, определяемую системными часами, и в большинстве случаев имеют точность намного хуже, чем одна микросекунда. Если не поддерживается, TIME возвращает пробелы. |
| **TRANSLATE** |  |
| Категория: | Строковая |
| Формат: | TRANSLATE(X,Y,[Z]) |
| Параметры: | X,Y,Z - символьные выражения. |
| Результат: | Если Z отсутствует, он, предполагается равным COLLATE(). Если Y короче, чем Z, она дополняется вправо пробелами, пока его длина не сравняется с длиной Z. Символы из строки X, встреченные в строке Z, заменены на соответствующие им символы из строки Y. |
| Тип результата: | Тот же, что у X |
| Примеры: | TRANSLATE('BDA', '123', 'ABC') возвращает '2D1' |
| **TRIM** |  |
| Категория: | Строковая |
| Формат: | TRIM(S[,L,T]) |
| Параметры: | S,L,T - символьные выражения. |
| Результат: | TRIM(S) возвращает символьную строку с удаленными начальными и конечными пробелами. TRIM(S,L,T) возвращает символьную строку с удаленными всеми ведущими символами S, которые появляются в L, и удаленными всеми конечными символы S, которые появляются в T. |
| Тип результата: | CHARACTER |
| Примеры: | TRIM(' ABCDE ') возвращает 'ABCDE'  TRIM('$\*\*\*1.23 ') возвращает '1.23' |
| **TRUNC** |  |
| Категория: | Арифметическая |
| Формат: | TRUNC(X) |
| Параметры: | X - арифметическое выражение. |
| Результат: | Возвращает целую часть выражения X  если X < 0 возвращает CEIL(X),  если X ≥ 0 возвращает FLOOR(X) |
| Тип результата: | Значение целого числа со знаком того же типа как X. |
| Примеры: | TRUNC(52.146) возвращает 52  TRUNC(-52.146) возвращает -52 |
| **UNLOCK** |  |
| Категория: | Вспомогательная |
| Формат: | UNLOCK(F,I) |
| Параметры: | F - файловая константа или переменная, определяющая файл, который должен быть открыт в режиме общего доступа. I - целое число FIXED BINARY(15), которое задает номер записи относительно размера записи, определенной в параметре ENVIRONMENT. |
| Результат: | Возвращает бит 1, если операция успешна или бит 0, если неудачна. Функция разблокирует запись, определенную I, чтобы другой пользователь мог получить доступ к ней. Запись остается разблокированной, пока не будет блокирована функцией LOCK, или до завершения программы. |
| Тип результата: | BIT(1) |
| Примечание: | Функция доступна, только если поддерживается операционной системой. |
| **UNSPEC** |  |
| Категория: | Вспомогательная |
| Формат: | UNSPEC(X) |
| Параметры: | X ссылка на элемент данных, внутреннее представление которого в памяти составляет 16 или менее битов. |
| Результат: | Возвращает содержание области памяти, занятой X. |
| Тип результата: | Битовая строка, длина которой равняется продолжительности внутреннего представления элемента данных, связанного с X. |
| Примеры: | Если num = 25000 (FIXED BINARY) то  UNSPEC(num) возвращает 0110000110101000 |

|  |  |
| --- | --- |
| **VERIFY** |  |
| Категория: | Строковая |
| Формат: | VERIFY(S,C) |
| Параметры: | S,C - символьные выражения. |
| Результат: | Возвращает целочисленное значение 0, если каждый из символов в S присутствует в C. Иначе, возвращает целое число, которое указывает позицию крайнего левого символа S, который отсутствует в C. |
| Тип результата: | FIXED BINARY |
| Примеры: | VERIFY('ABCDE','ABDE') возвращает 3  VERIFY('ABC123','1A2B3C4D') возвращает 0  VERIFY('','A') возвращает 0  VERIFY('A','') возвращает 1 |

**Конец раздела 13**

# Операторы PL/I

Этот раздел перечисляет форматы оператора PL/I в алфавитном порядке.

## Оператор ALLOCATE

**ALLOCATE базированная\_переменная SET(переменная\_указатель);**

Примеры:

declare

A character(16) based(P),

P pointer;

allocate A set(P);

## Оператор присваивания

**переменная = выражение;**

Примеры:

B = C\*D;

unspec (E) = F(I);

## Оператор BEGIN

**BEGIN;**

## Оператор CALL

**CALL Имя\_процедуры((инд\_1,...,инд\_n)] [(Список\_аргументов)];**

Примеры:

call P1;

call P2(A,B,C);

## Оператор CLOSE

**CLOSE FILE(идентификатор\_файла);**

Примеры:

close file(INP);

close file(B:PAYFINES.DAT);

## Оператор DECLARE (для скалярных переменных)

**DECLARE имя [список\_атрибутов];**

Примеры:

declare index\_count fixed binary(15);

declare pi float binary(53);

declare overtime\_pay fixed decimal(6,2);

## Оператор DECLARE (для массивов)

**DECLARE имя(пара\_индексов,...) [список\_атрибутов];**

Примеры:

**declare B(-2:5,-5:5,5:10);**

**declare class\_grades(30:100) character(25) varying;**

## Оператор DECLARE (для структур)

**DECLARE | DCL [уровень] имя [список\_атрибутов] ...**

**[,[уровень] имя [список\_атрибутов]];**

Примеры:

declare A fixed;

declare 1 B,

2 C NAME character(20),

2 D ADDRESS,

3 STREET character(20),

3 CITYST character(20),

3 ZIP character(5);

declare ZZ(10) fixed;

declare A fixed external;

## Оператор DECLARE (для данных ENTRY)

**DECLARE proc-name [(пара\_индексов\_1,...,пара\_индексов\_n)]**

**[ENTRY(список\_параметров)]**

**[EXTERNAL] [VARIABLE]**

**[RETURNS(атрибуты\_возвращаемого\_значения)];**

Примеры:

declare x entry;

declare p(0:10) entry(fixed,float) variable;

declare r returns(character(10));

## Оператор DECLARE (для данных FILE)

**DECLARE file\_id FILE (VARIABLE);**

Примеры:

declare f5 file;

declare f(5) file variable;

## Оператор DO

**DO (управляющая-переменная] спецификация\_do;**

где do-спецификация может быть одной из следующих:

**start-exp [TO end-exp] [BY incr-exp] [WHILE(условие)]**

**start-exp [BY incr-exp] [TO end-exp] [WHILE(условие)]**

**start-exp [REPEAT repeat-exp] [WHILE(условие)]**

Примеры:

do J=0;

do while(A<B);

do J = 1 TO 10;

do K = 10 TO 0 BY -2 while(A<B);

do P=START repeat P->NEXT while(P^=NULL);

## Оператор END

**END [метка];**

Примеры:

end;

end P1;

## Оператор FORMAT

**label: FORMAT(список\_элементов\_формата);**

Примеры:

L1: format(A(5));

L2: format(10 B4(2));

## Оператор FREE

**FREE [переменная\_указатель ->] базированная\_переменная;**

Примеры:

free A;

free P->A;

## Оператор GET EDIT

**GET [FILE(file\_id)] [SKIP[(nl)]]**

**EDIT(входной\_список) (список\_элементов\_формата);**

Примеры:

get edit(A,B,C)((3)f(5,2));

get file(INP) edit((Z(I) do I = 1 to 3))(A);

## Оператор GET LIST

**GET [FILE(file\_id)] [SKIP[(n1)]] LIST(входной\_список);**

Примеры:

get list(X,Y,Z);

## Оператор GOTO

**GOTO | GO TO метка\_константа | метка\_переменная;**

Примеры:

go to the\_end;

goto lab(K);

## Оператор IF

**IF выражение THEN действие\_1 [ELSE действие\_2]**

Примеры:

if A=2 then B=A\*\*2;

else;

if J>K then I = I+1;

else I = 1+3;

## Оператор %INCLUDE

**%INCLUDE 'спецификация\_файла';**

Примеры:

%include 'mathlib.pli';

%include 'constnts.dcl';

## Оператор NULL

**;**

Примеры:

;

else ;

## Оператор ON

**ON имя\_состояния ON-блок;**

Примеры:

on endfile(INP)

begin;

put list('END OF INPUT');

stop;

end;

on error put list(oncode());

## Оператор OPEN

**OPEN FILE(file\_id) [атрибуты\_файла];**

Примеры:

open file(INP) input;

open file(SYSPRINT) output;

## Оператор PROCEDURE

**имя\_процедуры: PROCEDURE | PROC [(список\_параметров)]**

**[OPTIONS(орция,...)) [RETURNS(список\_атрибутов)]**

**[RECURSIVE]**

Примеры:

P1: proc(A,B,C);

P2: procedure (ZZ) returns(float);

P3: proc(N) returns(fixed bin) recursive;

P4: procedure options(main);

## Оператор PUT EDIT

**PUT [FILE(file\_id)] [SKIP[nl]] [PAGE]**

**EDIT(выходной\_список)(список\_элементов\_формата);**

Примеры:

put edit(A,B,C) (F(5,2),X(3),2 E(l0,2));

put edit((Z(I) do I = 1 to 10))(A);

## Оператор PUT LIST

**PUT [FILE(file-id)] [SKIP[(n1)]] [PAGE] LIST (выходной\_список);**

Примеры:

put list(A,B,C);

put file(F) list((Z(I) do I = 1 to 10));

## Оператор READ VARYING

**READ [FILE(file-id)] INTO(v);**

Примеры:

read file(F) into(buffer);

## Оператор READ (для файлов SEQUENTIAL RECORD)

**READ FILE(file-id) INTO(x);**

Примеры:

read file(INP) into(XX);

## Оператор READ с KEY

**READ FILE(file-id) INTO(X) KEY(k);**

Примеры:

read file(INP) into(STRUC) key(IKEY);

## Оператор READ с KEYTO

**READ FILE(file-id) INTO(x) KEYTO(k);**

Примеры:

read file(INP) into(z) keyto(IKEY);

## Оператор %REPLACE

**%REPLACE идентификатор BY константа;**

Примеры:

%replace true by '1'b;

%replace rows by 10, columns by 6;

## Оператор RETURN

**RETURN [(возвращаемое\_значение)];**

Примеры:

return;

return(X);

return(A\*\*2);

## Оператор REVERT

**REVERT имя\_состояния;**

Примеры:

revert error;

revert endfile;

## Оператор SIGNAL

**SIGNAL имя\_состояния;**

Примеры:

signal error;

signal endfile(sysin);

## Оператор STOP

**STOP;**

## Оператор WRITE VARYING (для файлов STREAM)

**WRITE [FILE(file-id)] FROM(v);**

Примеры:

declare (XX,YY) character(200) varying;

write file(OUTPUT) from(XX);

write from(YY);

## Оператор WRITE (для файлов SEQUENTIAL RECORD)

**WRITE FILE(file-id) FROM(x);**

Примеры:

write file(OUTP) from (XX);

write file(F) from(STRUC);

## Оператор WRITE с KEYFROM

**WRITE FILE(file-id) FROM(x) KEYFROM(k);**

Примеры:

write file(KP) from(REC) keyfrom(IKEY);

**Конец раздела 14**

# Атрибуты данных

Этот раздел кратко описывает все атрибуты данных и классы памяти PL/I. Приводятся сокращения атрибутов. Обратитесь к соответствующим разделам с подробными описаниями атрибутов.

## ALIGNED

Атрибут ALIGNED, обычно вызывает выравнивание границ хранения переменной. В PL/I он не оказывает никакого эффекта, но включено для совместимости с другими реализациями. Например

**declare A(0:3) bit(4) aligned;**

## AUTOMATIC | AUTO

Класс памяти AUTOMATIC, определяет, что память выделяется переменной после активации блока, содержащего объявление. В PL/I область памяти AUTOMATIC вылепляется статически, за исключением рекурсивных процедур. Например

**declare A fixed binary; /\* эквивалентно следудущему \*/**

**declare A fixed binary auto;**

## BASED или BASED(p) или BASED(q())

Класс памяти BASED, определяет контролируемое пользователем выделение памяти переменной. В данном случае p - переменная указателя, и q - функция со значением указателя. Например

**declare A fixed binary based,**

**B(5) character(10) based(p),**

**C fixed binary based(f());**

## BINARY | BIN или BINARY(p) | BIN(p)

Атрибут BINARY определяет переменную BINARY с точностью p.

для переменных FIXED p ≤ 15

для переменных FLOAT p ≤ 53

Например

**declare I fixed binary(7),**

**F float binary(40);**

## BIT(n)

Атрибут BIT(n) определяет битовую строку длиной n, где n ≤ 16. Например,

**declare A bit(3);**

## BUILTIN

Атрибут BUILTIN определяет, что объявленное имя является одной из встроенных функций PL/I. Если имя встроенной функции объявлено в каком-либо блоке в качестве переменной, необходимо повторно объявить его с атрибутом BUILTIN, если необходимо ссылаться на него как на встроенную функцию в каком-либо содержащем его блоке. Например,

**declare sqrt builtin;**

## CHARACTER(n) | CHAR(n)

Атрибут CHARACTER(n) определяет символьную строку длиной n, где n ≤ 254. Например,

**declare A character(10),**

**B(5) character(4);**

## DECIMAL[(p[,q])] | DEC[(p[,q])]

Атрибут DECIMAL определяет десятичное число с помощью точности и масштаба (p, q), где p ≤ 15 и q ≤ p. Если q не определен, по умолчанию q = 0. Если не определены p и q одновременно, PL/I использует значение по умолчанию (7,0). Например,

**declare A fixed decimal(6,2);**

## ENTRY[(список\_параметров)]

Атрибут ENTRY определяет процедуры и функции, в которых список\_параметров, аналогичный используемому в объявлениях PROCEDURE. Например:

**declare H entry,**

**Z entry(10) (fixed),**

**Y entry(float) returns(float),**

**X entry variable;**

## ENVIRONMENT(параметры) | ENV(параметры)

Атрибут ENVIRONMENT определяет размер записи фиксированной и переменной длины для файлов с атрибутом RECORD, размеры внутреннего буфера, режим открытия файла и уровень защиты паролем. Параметром может быть один из следующих:

**Locked | L**

**Readonly | R**

**Shared | S**

**Password[(**уровень**)] | P[(**уровень**)]**

**Fixed(i) | F(i)**

**Buff(b) | B(b)**

где i - длина фиксированной записи, и b - размер внутреннего буфера. Оба выражены как целочисленные константы, Например,

**open file keyed env(f(100),b(4000));**

**open file(f6) input direct title('d:accounts.new;topaz')**

**env(shared,password(d),f(100),b(2000));**

## EXTERNAL | EXT

Атрибут EXTERNAL определяет область действия объявленного элемента - EXTERNAL. Т.е. элементом известным во всех блоках, где он объявлен как EXTERNAL. Например,

**declare A character(8) external;**

## FILE

Атрибут FILE определяет файл данных. Например

**declare F file,**

**FV file variable;**

## FIXED[(p[,q])]

Атрибут FIXED определяет арифметические данные с фиксированной точкой с точностью и масштабным коэффициентом (p, q). Если определено для данных BINARY, q должен быть 0. Например,

**declare A fixed binary,**

**B fixed decimal(5,2);**

## FLOAT[(p)]

Атрибут FLOAT определяет арифметические данные с плавающей точкой с точностью p, где p ≤ 53. Например,

**declare A float binary;**

## INITIAL(список\_значений) | INIT(список\_значений)

Атрибут INITIAL заставляет компилятор присваивать начальные значения переменной STATIC перед выполнением программы. Список\_значений является списком констант, разделенных запятыми, которые могут быть преобразованы в тип инициализируемой переменной. Любой константе в списке может предшествовать коэффициент повторения в круглых скобках. Например,

**declare A character(3) static initial('ABC'),**

**B(2) fixed binary static initial((2)5);**

## LABEL

Атрибут LABEL определяет переменную типа LABEL. Например,

**declare somewhere label;**

## PARAMETER

PARAMETER - класс памяти, который компилятор присваивает элементам данных, которые появляются в списке параметров. Область памяти для параметров выделяется, когда процедура вызова передает параметры в вызванную процедуру. В примере ниже, компилятор присваивает x класс памяти PARAMETER.

Пример:

**declare A entry(x float) external returns(float);**

## POINTER | PTR

Атрибут POINTER определяет переменную типа POINTER. Например,

**declare (p,q) pointer;**

## RETURNS(список\_атрибутов)

RETURNS (когда используется с атрибутом ENTRY) описывают список атрибутов значения, возвращаемого функцией. Например,

**declare A entry(float) returns(fixed);**

## STATIC

STATIC класс памяти, который заставляет компилятор выделять память переменным перед выполнением программы. Например,

**declare A character(10) static,**

**B fixed binary static initial(O);**

## VARIABLE

Атрибут VARIABLE (когда используется с атрибутами FILE или ENTRY) определяет элемент как переменную вместо константы. Например,

**declare F file variable,**

**P entry variable;**

## VARYING | VAR

Атрибут VARYING определяет символьную строку переменной длины. Например,

**declare A character(100) varying;**

**Конец раздела 15**

Приложение 1 Замечания по реализации

PL/I Digital Research основывается на американском национальном стандарте X3.74, подмножестве общего назначения PL/I (Подмножество G). Digital Research реализовала PL/I для нескольких микрокомпьютерных архитектур и операционных систем.

Это приложение описывает отличия между различными реализациями и отличия между этими реализациями и стандартом подмножества G.

В этом приложении используются следующие обозначения:

* **DRI PL/I** относится ко всем PL/I Digital Research.
* **PL/I-80™** относится к любой версии 8-разрядных реализаций PL/I совместимых с 8080 микропроцессорами, такими как 8080, 8085 и Z80®. Функциональные возможности, характерные для конкретной версии обозначены цифрами выпуска, т.е. PL/I-80 R1.4.
* **PL/I-86™** относится к любой версии 16-разрядных реализаций PL/I совместимой с 8086 микропроцессорами, такими как 8086 и 8088. Функциональные возможности, характерные для конкретной версии обозначены цифрами выпуска, т.е. PL/I-86 R1.0.
* **DOS** относится к дисковой операционной системе персонального компьютера IBM версии 1.1.

**П1.1. DRI PL/I по сравнению с PL/I подмножества G**

DRI PL/I соответствует стандарту подмножества G с учетом следующих исключений:

DRI PL/I не реализует атрибуты:

* DEFINED
* FLOAT DECIMAL
* PICTURE (он реализован как элемент формата редактирования при выводе)

Компилятор DRI PL/I не реализует размерность (неопределенной длины) \* в массивах или строках. Язык DRI PL/I не реализует в выражениях размерности в массивах или строках. Все размерности должны быть константами.

Язык DRI PL/I не реализует

**A = scalar;**

где A - переменная типа массив.

Язык DRI PL/I в настоящее время не реализует ключевое слов PARAMETER. Будущие версии PL/I-86 реализуют это ключевое слово.

Язык DRI PL/I требует, чтобы третьим параметром SUBSTR строки битов была константа.

Язык DRI PL/I не реализует некоторые встроенные функции. *Таблица 1-1* показывает, какие встроенные функции не доступны в соответствующих реализациях.

Язык DRI PL/I реализует следующие встроенные функции как расширения стандарта подмножества G:

* ASCII
* RANK

В языке DRI PL/I, оператор %REPLACE расширен, допуская многократные замены в одном операторе.

Таблица П1-1 Нереализованные встроенные функции

|  |  |  |  |
| --- | --- | --- | --- |
| **PL/I-80**  **R1.3 и R1.4** | **PL/I-86**  **R1.0 и R1.1** | **PL/I-86**  **R1.0 (DOS)** | **PL/I-86**  **R1.2** |
| ATANH | ATANH | ATANH | ATANH |
| COPY | COPY | COPY |  |
| DATE |  |  |  |
| REVERSE | REVERSE | REVERSE |  |
| SEARCH | SEARCH | SEARCH |  |
| STRING | STRING | STRING | STRING |
| TIME |  |  |  |
| TRIM | TRIM | TRIM |  |
| VALID | VALID | VALID | VALID |

Язык DRI PL/I добавляет следующие средства ввода-вывода для обработки файла ASCII:

* Формы оператора READ VARYING и WRITE VARYING для обработки записей ASCII переменной длины.
* Оператор GET EDIT расширен до ввода полной записи в формате A.

Язык DRI PL/I допускает управляющие символы в строковых константах. Эта функция несовместима со стандартом ANSI и будет изменена в будущих выпусках компилятора PL/I-86.

Язык DRI PL/I допускает такие операторы

**declare numbers(10) character(10)**

**static initial((10)'0123456789');**

В подмножестве G, необходимо использовать коэффициент повторения строки (1)

**declare numbers(10) character(10)**

**static initial((10)(1)101234567891);**

Компилятор DRI PL/I, в ON-блоке не освобождает память от переменной, обращение к которой вызвало данную ON-ситуацию, и не закрывает файл, для которого возникло условие ввода-вывода. ON-блок должен перейти к нелокальной метке.

Язык DRI PL/I не поддерживает частично индексируемые, и/или частично описанные смешанные агрегатные ссылки, которые определяют несвязанные области памяти.

Компилятор DRI PL/I имеет нестандартную реализацию рекурсивных процедур. При входе, они копируют в стек статический фрейм, содержащий адреса используемых объектов в памяти класса AUTOMATIC. При выходе такие процедуры копируют значения, сохраненные при вводе из стека назад в статический фрейм. Если процедура RECURSIVE вызывает подпрограмму и передает переменную AUTOMATIC ссылкой, она передает адрес переменной в статическом фрейме. Если подпрограмма затем вызывает исходную процедуру RECURSIVE, могут возникнуть нестандартные результаты.

Можно избежать нестандартных результатов, если заставить компилятор всегда передавать аргумент значением при создании вызова внутри процедуры RECURSIVE. Для передачи аргумента значением, заключите аргумент в круглые скобки.

**П1.2. Различия между PL/I-80 и PL/I-86**

Компиляторы PL/I-80 и PL/I-86 R1.0 не проверяет границы точности, заданной в операторе DECLARE. Например, получая объявление

**declare x fixed binary(35);**

компилятор обеспечит максимальную точность (15), не выдавая предупреждающее сообщение.

Компиляторы PL/I-80 и PL/I-86 R1.0 не проверяет, что процедура функция содержит оператор RETURN.

Компиляторы PL/I-80 и PL/I-86 R1.0 не восстанавливают состояние ON-блока при выходе из блока BEGIN.

Компиляторы PL/I-80 и PL/I-86 R1.0 не создают фиктивную переменную для параметра константы. Вы можете вынудить компилятор создать фиктивный параметр, заключив константу в круглые скобки.

Компилятор PL/I-80 не поддерживает операции сравнения значений FIXED BINARY, сумма или разница которых больше 32767 по абсолютному значению.

Компиляторы PL/I-80 и PL/I-86 R1.0 реализует стек состояний, имеющий глубину 16 уровней. В PL/I-80 и PL/I-86 R1.0 состояния, устанавливаемые ON-блоком в любом блоке, помещаются в тот же самый стек условий. Кроме того, состояния, устанавливаемые ON-блоком во вложенном блоке, помещаются в тот же стек условий.

Компилятор PL/I-86 R1.1 реализует условия ON правильно без ограничений.

**Примечание**: PL/I-80 и PL/I-86 R1.0 позволяет установку максимуму 16 состояний ON-блоков в любом месте программы. Включение больше 16 состояний ON-блоков являются невосстанавливаемой ошибкой. Программа во время выполнения прекращает выполнение и выводит следующее сообщение:

**Condition Stack Overflow**

В языках PL/I-80 и PL/I-86 R1.0, нельзя объявить переменную на основе указателя, которая является членом структуры. Например, следующее объявление недопустимо:

**declare**

**1 my\_structure,**

**2 some\_data fixed binary(7),**

**2 p pointer,**

**x float binary based(p);**

Компилятор PL/I-80 производит перемещаемый объектный код в формате Microsoft®. Этот формат ограничивает длину внешних имен 6 символами.

Компилятор PL/I-86 производит перемещаемый объектный код в формате Intel®. В этом формате ограничения на длину внешних имен отсутствуют.

Компиляторы PL/I-80 R1.4 и PL/I-86 R1.1 реализуют защиту паролем для файлов в атрибуте ENVIRONMENT, и обеспечивает встроенные функции LOCK и UNLOCK для блокирования и разблокирования отдельных записей в файле. Компилятор PL/I-86 R1.0 не реализует эту функциональную возможность.

Компиляторы PL/I-80 R1.4 и PL/I-86 R1.1 реализуют данные двойной точности FLOAT BINARY. PL/I-86 R1.0 не обеспечивает их.

Компилятор PL/I-86 использует формат IEEE для представления данных с плавающей точкой одинарной точности. PL/I-80 не обеспечивает. Таким образом, существует существенная несовместимость формата данных между PL/I-80 и PL/I-86. Программа PL/I-86 не может считать данные с плавающей точкой, записанные в дисковые файлы с помощью PL/I-80. *Приложение 2* содержит описания каждого формата и процедуру преобразования из формата PL/I-80 в формат PL/I-86.

Язык PL/I-86 R1.1 допускает использование в идентификаторах символов @ (знак at) и # (знак номера).

В языке PL/I-86 R1.2 встроенная функция COLLATE возвращает символьную строку длиной 256 вместо 128.

**П1.3. Выполнение PL/I-86 в DOS**

Операционная система DOS не поддерживает защиту файлов паролем, или блокирование и разблокирование отдельных записей. См. [*Раздел 10.1*](#_Оператор_OPEN) или *Руководство программиста* для получения дополнительной информации.

В DOS имена физических устройств заканчиваются двоеточием. Например, системная консоль - CON: и системное устройство печати - LPT1: или PRN:. В CP/M соответствующие имена - $CON и $LST. Смотрите [*Раздел 10.1*](#_Оператор_OPEN_1) или *Руководство программиста* для получения дополнительной информации.

**П1.4. Краткое описание различий**

*Таблица П1-2* содержит список различий различных реализаций DRI PL/I.

Таблица П1-2 Краткое изложение различий в реализациях

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Функциональная**  **возможность** | **PL/I-80**  **R1.3** | **PL/I-80**  **R1.4** | **PL/I-86**  **R1.0** | **PL/I-86**  **R1.0 D** | **PL/I-86**  **R1.1** | **PL/I-86**  **R1.2** |
| Формат объектного кода | Microsoft | Microsoft | Intel | Intel | Intel | Intel |
| Формат чисел с плавающей точкой | не IEEE | не IEEE | IEEE | IEEE | IEEE | IEEE |
| Поддержка данных Float Binary | нет | есть | нет | нет | есть | есть |
| Зашита паролем | нет | есть | нет | нет | есть | есть |
| Блокирование разблокирование записей | нет | есть | нет | нет | есть | есть |
| Уровни глубины стека условия | 16 | 16 | 16 | 16 | неограничен | |
| Имена устройств | $CON  $LST | $CON  $LST | $CON  $LST | CON:  LPT1: | $CON  $LST | $CON  $LST |
| @, # допустимы в идентификаторах | нет | нет | нет | нет | да | да |
| Длинна строки возвращаемой COLLATE() | 128 | 128 | 128 | 128 | 128 | 256 |
| Переменные на основе указателя в структуре | нет | нет | нет | нет | да | нет |
| Создание фикивной переменной для константы параметра | нет | нет | нет | нет | да | да |
| Восстановление состояния ON-units при завершении блока BEGIN | нет | нет | нет | нет | да | да |
| Сравнение FIXED BINARY  > 32767 | нет | нет | да | да | да | да |
| Не поддерживаемые встроенные функции | ATANH  COPY  DATE  REVERSE  SEARCH  STRING  TIME  TRIM  VALID | ATANH  COPY  DATE  REVERSE  SEARCH  STRING  TIME  TRIM  VALID | ATANH  COPY  REVERSE  SEARCH  STRING  TRIM  VALID | ATANH  COPY  REVERSE  SEARCH  STRING  TRIM  VALID | ATANH  COPY  REVERSE  SEARCH  STRING  TRIM  VALID | ATANH  STRING  VALID |

**Конец приложения 1**

Приложение 2 Внутреннее представление данных

Это приложение описывает внутренние форматы данных PL/I. Это знание жизненно важно при использовании базированных переменных с наложением мест хранения, чтобы не уничтожить содержимое смежных мест хранения. Также знание внутреннего представления данных полезно, когда необходим интерфейс подпрограммам языка ассемблер с программами языка высокого уровня и библиотекой подпрограмм времени выполнения PL/I.

**Примечание**: в этом разделе термин PL/I применяется к PL/I-80 и PL/I-86, если не обозначено иначе.

**П2.1 Представление FIXED BINARY**

PL/I хранит данные FIXED BINARY в одной из двух форм, в зависимости от заявленной точности. Значения FIXED BINARY с точностью 1-7 хранятся в однобайтовых расположениях, и значения с точностью 8-15 в расположениях из двух байт (слове). В многобайтных ячейках PL/I сохраняет младший значащий байт по самому нижнему адресу памяти.

PL/I представляет все данные FIXED BINARY в дополнительном коде, позволяя однобайтовые значения в диапазоне от -128 до +127 и двухбайтовые значения в диапазоне от -32768 до +32767.

*Рисунок П2-1* показывает представление хранения в однобайтовых расположениях и в расположениях из двух байт для значений 0, 1, и -1. Каждое упакованное значение представляет байт памяти и показано в двоичном и шестнадцатеричном значениях.

┌─────────┐ ┌─────────┬─────────┐

│0000 0000│ │0000 0000│0000 0000│

├────┬────┘ ├─────┬───┴─────────┘

│ 00 │ │00 00│

└────┘ └─────┘

FIXED BINARY(7) FIXED BINARY(15)

┌─────────┐ ┌─────────┬─────────┐

│0000 0001│ │0000 0001│0000 0000│

├────┬────┘ ├─────┬───┴─────────┘

│ 01 │ │01 00│

└────┘ └─────┘

FIXED BINARY(7) FIXED BINARY(15)

┌─────────┐ ┌─────────┬─────────┐

│1111 1110│ │1111 1110│1111 1111│

├────┬────┘ ├─────┬───┴─────────┘

│ FE │ │FE FF│

└────┘ └─────┘

Рисунок П2-1 Представление FIXED BINARY

**П2.2 Представление FLOAT BINARY**

**П2.2.1 Одинарная точность**

**PL/I-80**

PL/I-80 хранит двоичные данные одинарной точности с плавающей точкой в формате Microsoft. Этот формат использует четыре последовательных байта с 32 битами, содержащий следующие поля: 23-разрядная мантисса, бит знака и 8-разрядный показатель степени (порядок). Младший значащий байт мантиссы находится по самому нижнему адресу памяти.

┌─────────────┬───┬──────────────┐

│ порядок │ з │ мантисса │

└─────────────┴───┴──────────────┘

↑ ↑ ↑ ↑

31 23 22 0

Рисунок П2-2 Формат числа PL/I-80 с плавающей точкой одинарной точности

Формат Microsoft нормализует числа с плавающей точкой, поэтому, старший значащий бит мантиссы всегда 1 для ненулевых чисел. Поскольку самый старший бит мантиссы должен быть равен 1 для отличных от нуля чисел, эта позиция двоичного разряда используется для знака. Это называется использованием неявного нормированного бита, и считается, что двоичная точка расположена непосредственно слева от нормализованного бита.

Для упрощения некоторых видов сравнений, байт двоичного порядка имеет смещение 128 (в десятичной системе) или 80 (в шестнадцатеричной), чтобы 81 представлял порядок 1, а 7F представлял порядок -1. Нулевая мантисса имеет байт порядка 00.

Предположим, что двоичное значение с плавающей точкой представлено в памяти следующим образом

┌──┬──┬──┬──┐

│00│00│40│81│

└──┴──┴──┴──┘

3 2 1 0

Представление битового потока имеет следующий вид:

8 1 4 0 0 0 0 0

1000 0001 0100 0000 0000 0000 0000 0000

↑ ↑ ↑

31 22 0

Когда смещение вычитается из порядка, истинный двоичный порядок равняется 1.

1000 0001

-1000 0000 (80h)

0000 0001

Мантисса представляется как

0100 0000 0000 0000 0000 0000

↑

з

Старший бит, равный нулю, означает, что знак положительный. Восстановление неявного, нормализованного бита производит поток битов

1100 0000 0000 0000 0000 0000

Так как двоичная точка находится на одну позицию влево от неявного нормализованного бита, значение мантиссы следующее

1100 0000 0000 0000 0000 0000

1100 ... представляет 2-1 + 2-2. Умножением на истинный показатель 21 мы получаем

21 (2-1 + 2-2 ) = 2(1/2 + 1/4) = 1 + 1/2 = 1.5

Таким образом, четырехбайтовое значение:

┌──┬──┬──┬──┐

│00│00│40│81│

└──┴──┴──┴──┘

является двоичным представлением с плавающей точкой десятичного числа 1.5.

**PL/I-86**

PL/I-86 хранит двоичные числа с плавающей запятой одинарной точности, используя формат IEEE. Этот формат использует четыре последовательных байта, с 32 битами, содержащими следующие поля: 23-разрядная мантисса, 8-разрядный порядок, бит знака. Наименее значащий байт мантиссы находится по адресу в памяти.

┌───┬─────────────┬──────────────┐

│ з │ порядок │ мантисса │

└───┴─────────────┴──────────────┘

↑ ↑ ↑ ↑ ↑

31 30 23 22 0

Рисунок П2-3 Формат числа IEEE с плавающей точкой одинарной точности

Формат IEEE нормализует числа с плавающей точкой, поэтому самый значащий бит мантиссы всегда 1 для ненулевых чисел. Поскольку старший значащий бит мантиссы должен быть 1 для ненулевых чисел, этот бит не сохраняется. Это называется использованием неявного нормализованного бита, и считается, что двоичная точкой располагается непосредственно справа от нормированного бита.

В двоичном формате IEEE (одинарная точность) порядок имеет смещение 127 (в десятичной системе) или 7F (в шестнадцатеричной), таким образом, 80 представляет порядок +1, в то время как 7E представляет порядок -1.

Предположим, что двоичное значение с плавающей точкой представлено в памяти следующим образом

┌──┬──┬──┬──┐

│00│00│C0│3F│

└──┴──┴──┴──┘

3 2 1 0

Представление в виде битовой строки имеет следующий вид:

3 F C 0 0 0 0 0

0011 1111 1100 0000 0000 0000 0000 0000

↑ ↑ ↑

31 22 0

Старший бит, равный нулю, указывает, что знак положителен, и порядок имеет смещение 7F, таким образом, истинный двоичный порядок 0.

0 0111 1111 1000 0000 0000 0000 0000 000

знак порядк мантисса

Восстановление неявного, нормализованного бита производит поток битов

1.100 0000 0000 0000 0000 0000

Так как двоичная точка находится на одну позиция справа от неявного нормализованного бита, значение мантиссы следующее

1 100 0000 0000 0000 0000 0000

1 1 в двоичном файле представляет 20+2-1. Умножением на истинный показатель 20 мы получаем

20 (20 + 2-1 ) = 1 (1 + 1/2) = 1 + 1/2 = 1.5

Таким образом, четырехбайтовое значение:

┌──┬──┬──┬──┐

│00│00│C0│3F│

└──┴──┴──┴──┘

является двоичным представлением значения с плавающей точкой одинарной точности десятичного числа 1.5.

Вы можете преобразовать данные, записанные в формате не-IEEE в формат IEEE при помощи процедуры, приведенной в *Листинге 1*.

**SPBOT086: procedure(f) returns(float binary(24));**

**declare**

**(f,r) float binary(24),**

**b fixed binary(7),**

**(fp,rp) pointer;**

**declare**

**1 f80 based (fp)**

**2 (word0,word1) bit(16);**

**declare**

**1 f86 based(rp),**

**2 (word0,word1) bit(16);**

**declare**

**1 f80over based(fp),**

**2 (byte0,byte1,byte2,byte3) fixed binary(7);**

**fp = addr(f);**

**rp = addr(r);**

**r = f; /\* copy the whole source to target \*/**

**/\* copy exponent and adjust bias by 2:**

**1 for 127 vs 128, 1 for 1. not .1 \*/**

**b = byte3 - 2;**

**substr(rp->f86.word1,2,8) = unspec(b);**

**/\* copy sign bit \*/**

**substr(rp->f86.word1,1,1) = substr(fp->f80.word1,9,1);**

**return(r);**

**end SP80T086;**

Листинг 1 Процедура преобразования формата с плавающей точкой

**П2.2.2 Двойная точность**

Компиляторы PL/I-80 R1.4 и PL/I-86 R1.1 хранят двоичные числа двойной точности с плавающей запятой, используя формат IEEE. Этот формат использует восемь последовательных байтов с 64 битами, содержащими следующие поля: 52-разрядная мантисса, 11-разрядная экспонента и знаковый бит.

┌───┬─────────────┬──────────────┐

│ з │ порядок │ мантисса │

└───┴─────────────┴──────────────┘

↑ ↑ ↑ ↑

63 62 51 0

Рисунок П2-4 Формат числа с плавающей точкой двойной точности

Формат IEEE нормализует числа с плавающей точкой, поэтому самый значащий бит мантиссы всегда 1 для ненулевых чисел. Поскольку старший значащий бит мантиссы должен быть 1 для ненулевых чисел, этот бит не сохраняется. Это называется использованием неявного нормализованного бита, и считается, что двоичная точкой располагается непосредственно справа от нормированного бита.

В формате IEEE (двойная точность) порядок имеет смещение 1023 (в десятичной системе) или 3FF (в шестнадцатеричной), таким образом, 400 представляет порядок +1, в то время как 3FE порядок экспоненту -1.

Для примера, предположим, что двоичное значение с плавающей точкой представлено в памяти, как показано в следующем примере:

┌────┬────┬────┬────┬────┬────┬────┬────┐

│ 00 │ 00 │ 00 │ 00 │ 00 │ C0 │ 43 │ C0 │

└────┴────┴────┴────┴────┴────┴────┴────┘

Младшие адреса Старшие адреса

В этом случае мантисса представляет собой следующий битовый поток

3 C 0

0011 1100 0000 ...

Восстановление неявного нормализованного бита приводит

1001 1110 0000 ...

Показатель вычисляется следующим образом:

C 0 4

1100 0000 0100

Старший бит равен 1, таким образом, знак отрицательный. Игнорирование знакового бита приводит к показателю

4 0 4

0100 0000 0100

которое имеет смещение 3FF, таким образом, истинный двоичный порядок

404

-3FF

5

Таким образом, двоичное число

1001 11 10 0000 ...

↑

является десятичным числом 39.5. Таким образом, восьми-байтовое значение:

┌──┬──┬──┬──┬──┬──┬──┬──┐

│00│00│00│00│00│C0│43│C0│

└──┴──┴──┴──┴──┴──┴──┴──┘

является двоичным представлением значения с плавающей точкой двойной точности десятичного числа -39.5

**П2.3 Представление FIXED DECIMAL**

PL/I хранит элементы данных FIXED DECIMAL в упакованном дополнительном десятичном коде BCD (Binary Coded Decimal). Каждая цифра BCD занимает полубайт или тетраду. PL/I хранит младшую значащую пару BCD по самому нижнему адресу памяти с зарезервированной одной позицией символа BCD для знака. Положительные числа в старшей позиции разряда содержат знак 0, а отрицательные числа - 9.

Число байтов, занимаемых числом FIXED DECIMAL, зависит от его заявленной точности. Учитывая десятичное число точности p, PL/I резервирует некоторое количество байтов, равное

**FLOOR((p + 2)/2)**

где p может изменяться от 1 и 15. Это приводит минимум к 1 байту и максимум к 8 байтам для размещения элементов данных FIXED DECIMAL.

Например, если вы объявляете число 12345 с точностью 5, тогда PL/I зарезервирует FLOOR((5 + 2)/2) = 3 байта для хранения и представит число в следующем виде:

┌──┬──┬──┐

│45│23│01│

└──┴──┴──┘

PL/I хранит отрицательные числа FIXED DECIMAL в дополнительном десятичном коде. Для получения кода дополненного до десяти, сначала получите дополнение до девяти и затем к результату добавьте 1. Например, число -2, выраженное в дополнительном десятичном коде следующее

(9 - 2) + 1 = 8

Добавление разряда знака дает

98

Если вы объявляете -2 с точностью 5, то PL/I представляет его следующим образом

┌──┬──┬──┐

│98│99│99│

└──┴──┴──┘

**П2.4 Представление CHARACTER**

PL/I хранит символьные данные в одной из двух форм, в зависимости от объявления. Он хранит строки символов фиксированной длины, объявленные как CHARACTER(n) в n смежных байтах, с первым символом расположенном в строке по самому младшему адресу области хранения.

PL/I резервирует n+1 байтов для переменных, объявленных как CHARACTER(n) VARYING с дополнительным байтом, содержащим длину символьной строки. Длина может изменяться от 0 до 254. Максимальная длина строки любого типа - 254 символа.

Для примера, предположим, что переменная A объявлена как CHARACTER(20).

Присвоение

**A = 'Walla Walla Wash';**

приводит к следующему распределению хранения:

**Walla⌴Walla⌴Wash⌴⌴⌴⌴**

где **⌴** представляет пробел. Если A объявлена как данные CHARACTER(20) VARYING, PL/I хранит ту же строку как

**10Walla⌴Walla⌴Wash????**

представляет собой неопределенные позиции символов.

**П2.5 Представление BIT**

PL/I представляет данные строки битов в двух формах, в зависимости от заявленной точности. Он хранит значение битовой строки длиной 1-8 в одном байте и битовой строки длиной 9-16, в одном слове (двух байтах). PL/I в слове сохраняет значение младшего значащего байта по самому младшему адресу памяти. Битовые значения сохраняется выравненными по левому краю, и если точность не равняется точно 8 или 16 битам, лишние биты справа игнорируются.

*Рисунок П2-5* показывает хранение для постоянных значений строки битов '1'b, 'A0'b4 и '1234'b4 в расположениях из одного и двух байтов. Каждое помещенное в прямоугольник значение представляет собой байт.

BIT(8) BIT(16)

┌─────────┐ ┌─────────┬─────────┐

│1000 0000│ │0000 0000│1000 0000│

└─────────┘ └─────────┴─────────┘

BIT(8) BIT(16)

┌─────────┐ ┌─────────┬─────────┐

│1010 0000│ │0000 0001│1010 0000│

└─────────┘ └─────────┴─────────┘

BIT(8) BIT(16)

┌─────────┬─────────┐

Н/Д │0011 0100│0001 0010│

└─────────┴─────────┘

Рисунок П2-5 Представление данных строки битов

**П2.6 Представление данных POINTER**

Компиляторы PL/I-80 и PL/I-86 R1.0 хранят переменные, которые обеспечивают доступ к адресам памяти как два непрерывных байта. Байт младшего разряда сохраняется по самому младшему адресу памяти. Элементы данных POINTER отображаются в виде

┌────┬────┐

│ LS │ MS │

└────┴────┘

где LS обозначает младший значащий байт адреса, и MS обозначает старший значащий байт.

**П2.7 Представление данных ENTRY и LABEL**

Компиляторы PL/I-80 и PL/I-86 R1.0 хранят данные ENTRY и LABEL в двух расположенных рядом байтах. Байт младшего разряда сохраняется по самому младшему адресу памяти. Элементы данных ENTRY и LABEL отображаются в виде

┌────┬────┐

│ LS │ MS │

└────┴────┘

где LS обозначает младший значащий байт адреса, и MS обозначает старший значащий байт.

Компилятор PL/I-86 R1.1 выделяет 8 байтов для элементов данных ENTRY и LABEL. Область из 8 байтов содержит следующие поля:

┌──────────┬──────────────┬─────────────┬──────┐

│ Смещение │ Сегмент кода │ Фрейм стека │ Стек │

└──────────┴──────────────┴─────────────┴──────┘

2 2 2 2

**П2.8 Представление файловых констант**

PL/I связывает каждую файловую константу с блоком параметров файла (FPB). FPB занимает 57 непрерывных байтов, содержащих различные поля, некоторые из которых зависят от реализации.

**Примечание**: каждое объявление файла вызывает статическое выделение памяти для соответствующего FPB. При открытии файла, возникают накладные расходы для операционной системы на FCB и буфер. Система времени выполнения динамично выделяет для них место из свободной области памяти.

**П2.9 Хранение агрегатов**

PL/I хранит элементы агрегированных данных непрерывно без байтов заполнителей. Битовые данные всегда хранятся не выровненными, но каждая битовая переменная начинается в новом байте. Массивы хранятся по строкам, причем крайний правый индекс изменяется быстрее.

Например, объявление:

**declare A(2,2,2);**

приводит к следующему распределения памяти:

┌─────┬─────┬─────┬─────┬─────┬─────┬─────┬─────┐

│1,1,1│1,1,2│1,2,1│1,2,2│2,1,1│2,1,2│2,2,1│2,2,2│

└─────┴─────┴─────┴─────┴─────┴─────┴─────┴─────┘

Младшие адреса Старшие адреса

**Конец приложения 2**

Приложение 3 Интерфейсные соглашения

Это приложение описывает стандартный набор соглашений интерфейса для взаимодействия программ PL/I с подпрограммами ассемблера и с программами, написанными на других языках высокого уровня.

**Примечание**: в этом разделе термин PL/I применяется к PL/I-80 и PL/I-86, если не обозначено иное.

**П3.1 Передача параметров используя блок параметров**

Параметры между программой PL/I и подпрограммой на ассемблере можно передать, загрузив регистровую пару адресом блока параметров, содержащего значения указателей. Эти указатели, в свою очередь, приводят к фактическим значениям параметров. Число параметров, длина параметра и тип должны быть неявно определены соглашением между программой вызова и вызванной подпрограммой. *Рисунок П3-1* иллюстрирует это понятие. Поля адреса произвольны.

Регистровая Блок Параметры

пара параметров

┌──────┐ ┌────────────┐

HL (8080) ┌──────> 1000: │ 2000 ├────────────> 2000: │ параметр 1 │

┌──────┐ │ ├──────┤ └────────────┘

│ 1000 ├──────┘ │ 3000 ├───────┐

└──────┘ ├──────┤ │ ┌────────────┐

BX (8086) │ 4000 ├────┐ └────> 3000: │ параметр 2 │

├──────┤ │ └────────────┘

│ . │ │

│ . │ │ ┌────────────┐

│ . │ └───────> 4000: │ параметр 3 │

├──────┤ └────────────┘

│ 5000 ├──┐ .

└──────┘ │ .

│ .

│ ┌────────────┐

└─────────> 5000: │ параметр n │

└────────────┘

Рисунок П3-1 Механизм передачи параметров PL/I

Следующий пример иллюстрирует этот механизм передачи параметров. Предположим, что программа PL/I использует большое количество операций делания с плавающей точкой, где каждое деление является степень двух. Предположим также, что итерационный цикл, где происходят деления, критический по скорости, и что полезно иметь подпрограмму на ассемблере для выполнения деления.

Подпрограмма на ассемблере просто уменьшает двоичный порядок числа с плавающей точкой для каждой степени двойки при делении. Уменьшение показателя степени эффективно выполняет операцию деления без накладных расходов на распаковку числа, выполнение общей операции деления и переупаковки результата. Во время деления подпрограмма на ассемблере может произвести потерю значимости и должна передать состояние UNDERFLOW программе PL/I, если она происходит.

Следующие три листинга показывают программы, которые демонстрируют передачу параметров. *Листинг 2* показывает программу DTEST, которая тестирует операцию деления. *Листинг 3* показывает DIV2.ASM, подпрограмму на ассемблере 8080, которая выполняет деление. В строке 8 DTEST определяет DIV2 как константу внешней точки входа с двумя параметрами: FIXED(7) и двоичное значение с плавающей точкой. *Листинг 4* содержит DIV2.A86, которая является той же подпрограммой на языке ассемблера 8086.

В каждой итерации DO-группы DTEST сохраняет тестовое значение 100 в f (строка 13) и передает его в подпрограмму DIV2 (строка 14). В каждом вызове DIV2 DTEST изменяет значение f на f/(2\*\*i) и распечатывает его, используя оператор PUT. При вызове DIV2 получает два адреса, которые соответствуют этим двум параметрам i и f.

После записи DIV2 загружает значение i в аккумулятор и устанавливает соответствующую регистровую пару указывать на поле экспоненты входного числа с плавающей точкой. Если экспонента - 0, DIV2 сразу возвращается, так как получающееся значение - 0.

В противном случае, циклы подпрограммы в метке dby2 при обратном отсчете показатель в степени 2 уменьшается до нуля. Если показатель достигает нуля во время этого процесса подсчета, DIV2 устанавливает состояние UNDERFLOW.

В DIV2, вызов ?signal демонстрирует формат параметров на ассемблере, использующих интерфейс. Подпрограмма ?signal - часть библиотеки подпрограмм времени выполнения PL/I (PLILIB).

Эта подпрограмма загружает в соответствующую регистровую пару адрес списка параметров сигнала, обозначенного siglst. Список параметров сигнала, в свою очередь, является блоком параметров из четырех адресов, приводящих к сигнальному коду sigcode, сигнальному подкоду sigsub, индикатору имени файла sigfil (здесь не используется) и вспомогательному сообщению sigaux, который является последним параметром.

Вспомогательное сообщение может предоставить дополнительную информацию при возникновении ошибки. Сигнальная подпрограмма распечатывает сообщение до исчерпания длины строки (32, в этом случае), или встречи с двоичным 00 в строке.

*Листинг 5* показывает сокращенный вывод этой тестовой программы. Счетчик цикла i становится отрицательным, когда он достигает 128, но подпрограмма DIV2 обрабатывает это значение как значение величины без знака, таким образом, UNDERFLOW происходит, когда i достигает - 123.

1 a /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

2 a /\* Эта программа тестирует подпрограмму на ассемблере,\*/

3 a /\* выполняющую деление с плавающей точкой. \*/

4 a /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

5 a dtest:

6 b procedure options(main);

7 b declare

8 b div2 entry(fixed(7),float),

9 b i fixed(7),

10 b f float;

11 b

12 c do i = 0 by 1;

13 c f = 100;

14 c call div2(i,f);

15 c put skip list('100 / 2 \*\*',i,'=',f);

16 c end;

17 b

18 b end dtest;

Листинг 2 Программа DTEST

title 'division by power of two'

public div2

extrn ?signal

; entry:

; p1 -> fixed(7) power of two

; p2 -> floating point number

; exit:

; p1 -> (unchanged)

; p2 -> p2 / (2\*\*p1)

div2: ;HL = .low(.p1)

mov e,m ;low(.p1)

inx h ;HL = .high(.p1)

mov d,m ;DE = .p1

inx h ;HL = .low(p2)

ldax d ;a = p1 (power of two)

mov e,m ;low(.p2)

inx h ;HL = .high(.p2)

mov d,m ;DE = .p2

xchg ;HL = .p2

;

; A = power of 2, HL = .low byte of fp num

inx h ;to middle of mantissa

inx h ;to high byte of mantissa

inx h ;to exponent byte

inr m

dcr m ;p2 already zero?

rz ;return if so

dby2: ;divide by two

ora a ;counted power of 2 to zero?

rz ;return if so

dcr a ;count power of two down

dcr m ;count exponent down

jnz dby2 ;loop again if no underflow

;

;underflow occurred, signal underflow condition

lxi h,siglst;signal parameter list

call ?signal ;signal underflow

ret ;normally, no return

;

dseg

siglst: dw sigcod ;address of signal code

dw sigsub ;address of subcode

dw sigfil ;address of file code

dw sigaux ;address of aux message

; end of parameter vector, start of params

sigcod: db 3 ;03 = underflow

sigsub: db 128 ;arbitrary subcode for id

sigfil: dw 0000 ;no associated file name

sigaux: dw undmsg ;0000 if no aux message

undmsg: db 32,'Underflow in Divide by Two',0

end

Листинг 3 DIV2.ASM Программа на языке ассемблера (8080)

; Routine to divide single precision float value by 2

cseg

public div2

extrn ?signal:near

; entry:

; p1 -> fixed(7) power of two

; p2 -> floating point number

; exit:

; p1 -> (unchanged)

; p2 -> p2 / (2\*\*p1)

div2: ;BX = .low(.p1)

mov si,[bx] ;SI = .p1

mov bx,2[bx] ;BX = .p2

lods al ;AL = p1 (power of 2)

; AL = power of 2, BX = .low byte of fp num

cmp byte ptr 3[bx],0 ;p2 already zero?

jz done ;exit if so

dby2: ;divide by two

test al,al ;counted power of 2 to zero?

jz done ;return if so

dec al ;count power of two down

sub word ptr 2[bx],80h ;count exponent down

test word ptr 2[bx],7f80h ;test for underflow

jnz dby2 ;loop again if no underflow

; Underflow occurred, signal underflow condition

mov bx,offset siglst;signal parameter list

call ?signal ;signal underflow

done: ret ;normally, no return

dseg

siglst dw offset sigcod ;address of signal code

dw offset sigsub ;address of subcode

dw offset sigfil ;address of file code

dw offset sigaux ;address of aux message

; end of parameter vector, start of params

sigcod db 3 ;03 = underflow

sigsub db 128 ;arbitrary subcode for id

sigfil dw 0000 ;no associated file name

sigaux dw offset undmsg ;0000 if no aux message

undmsg db 32,'Underflow in Divide by Two',0

end

Листинг 4 DIV2.A86 Программа на языке ассемблера (8086)

A>dtest

100 / 2 \*\* 0 = 1.000000E+02

100 / 2 \*\* 1 = 5.000000E+01

100 / 2 \*\* 2 = 2.500000E+01

100 / 2 \*\* 3 = 1.250000E+01

100 / 2 \*\* 4 = 0.625000E+01

100 / 2 \*\* 5 = 3.125000E+00

100 / 2 \*\* 6 = 1.562500E+00

100 / 2 \*\* 7 = 0.781250E+00

100 / 2 \*\* 8 = 3.906250E-01

100 / 2 \*\* 9 = 1.953125E-01

100 / 2 \*\* 10 = 0.976562E-01

.

.

.

.

100 / 2 \*\* 127 = 0.587747E-36

100 / 2 \*\* -128 = 2.938735E-37

100 / 2 \*\* -127 = 1.469367E-37

100 / 2 \*\* -126 = 0.734683E-37

100 / 2 \*\* -125 = 3.673419E-38

100 / 2 \*\* -124 = 1.836709E-38

100 / 2 \*\* -123 = 0.918354E-38

100 / 2 \*\* -122 = 4.591774E-39

UNDERFLOW (128), Underflow in Divide By Two

Traceback: 017F 011B

A>

Листинг 5 Вывод DTEST (Сокращенный)

**П3.2 Возврат значений в регистрах или в стеке**

В качестве альтернативы возврата значений через блок параметров, PL/I имеет подпрограммы, которые производят значения функции, которые затем возвращают непосредственно в регистрах или в стеке. В этом разделе представлены соглашения для возврата данных в качестве функциональных значений. Ссылки на регистры 8086 находятся в круглых скобках.

**П3.2.1 Возвращение данных FIXED BINARY**

Функции, которые возвращают элементы FIXED BINARY, делают это, оставляя результат в регистре или регистровой паре, в зависимости от точности элемента данных.

PL/I возвращает данные FIXED BINARY с точностью 1-7 в регистре A(AL) и данные с точностью 8-15 в регистровой паре HL(BX). Всегда безопасно возвратить значение в HL(BX) и скопировать младший байт в A(AL), таким образом, регистр A(AL), равен регистру L(BL) по возврату.

**П3.2.2 Возвращение данных FLOAT BINARY**

PL/I-80 R1.4 возвращает числа с плавающей точкой одинарной точности в стеке как четыре непрерывных байта в формате Microsoft. Младший байт мантиссы находится на вершине стека, за ним средний байт, затем старший байт. Четвертый байт является показателем степени числа. Старший бит мантиссы является знаковым битом.

Например, значение 1.5 возвращается как

┌──┬──┬──┬──┐

│00│00│40│81│ (низкий стек) ──>

└──┴──┴──┴──┘

↑

SP

PL/I-86 R1.0 возвращает числа с плавающей точкой одинарной точности в стеке как четыре непрерывных байта в формате IEEE. Младший байт мантиссы находится наверху стека, за ним средний байт, затем старший байт. Старший бит является знаковым битом, и младший бит младшего разряда экспоненты находится в старшем байте мантиссы.

Например, значение 1.5 возвращается как

┌──┬──┬──┬──┐

│00│00│C0│3F│ (низкий стек) ──>

└──┴──┴──┴──┘

↑

SP

PL/I-80 R1.4 и PL/I-86 R1.1 возвращают числа с плавающей точкой двойную точность как восемь непрерывных байтов в стеке. Младший байт мантиссы наверху стека. Экспонента занимает три полубайта: восьмой байт и старший полубайт седьмого байта.

Например, значение -39.5 возвращается как

┌──┬──┬──┬──┬──┬──┬──┬──┐

│00│00│00│00│00│C0│43│C0│ (низкий стек) ──>

└──┴──┴──┴──┴──┴──┴──┴──┘

↑

SP

**П3.2.3 Возвращение данных FIXED DECIMAL**

PL/I возвращает данные FIXED DECIMAL в стеке как 8 непрерывных байтов. Младшая пара BCD в верхней части стека. Число представляется в форме дополнения до десять и расширенное для знака символом старшей позиции с положительным знаком, обозначаемым 0, и знаком минус, обозначаемым 9.

Например, PL/I возвращает значение -2 как

┌──┬──┬──┬──┬──┬──┬──┬──┐

│98│99│99│99│99│99│99│99│ (низкий стек) ──>

└──┴──┴──┴──┴──┴──┴──┴──┘

↑

SP

**П3.2.4 Возвращение данных CHARACTER**

PL/I-80 и PL/I-86 R1.0 возвращает элементы данных CHARACTER в стеке с длиной строки в регистре A(AL). Например, строка

'Walla Walla Wash'

возвращается, как показано ниже:

A (8080)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 10 |  | W | a | l | l | a | ⌴ | W | a | l | l | a | ⌴ | W | a | s | h | (низкий стек) ─> |
| AL (8086) | | ↑  SP | | | | | | | | | | | | | | | | |

где регистр содержит длину строки 10 (шестнадцатеричное), а указатель стека SP - адрес первого символа в строке.

PL/I-86 R1.1 возвращает элементы данных CHARACTER в стеке как строку переменной длины с полем из 4 байтов длины сначала. Например, строка:

'Walla Walla Wash'

возвращается, как

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 10 | 00 | 00 | 00 | W | a | l | l | a | ⌴ | W | a | l | l | a | ⌴ | W | a | s | h | (низкий стек) ─> |

↑

SP

**П3.2.5 Возвращение данных BIT**

PL/I возвращает данные битовой строки в регистре или регистровой паре, в зависимости от точности элемента данных.

PL/I возвращает строки битов длины 1-8 в регистре A(AL) и строки битов длины 9-16 в регистровой паре HL(BX). Строки битов остаются выровненными по ширине в их полях, таким образом, BIT(1) оценивается как истина, возвращается в регистре HL(BX) как 80 (шестнадцатеричный). Безопасно возвратить битовые значения в регистровой паре HL(BX) и скопировать старший байт в A(AL), таким образом регистр A(AL), равен регистру H(BH) по возврату.

**П3.2.6 Возвращение переменных POINTER**

PL/I-80 и PL/I-86 возвратите переменные указателя в регистровой паре HL(BX). При возврате переменной метки, которая может быть целью операции GOTO, подпрограмма, содержащая метку, должна восстановить стек к надлежащему уровню, когда управление достигает метки.

**П3.2.7 Возвращение переменных ENTRY и LABEL**

PL/I-80 R1.4 и PL/I-86 R1.0 возвращают переменные ENTRY и LABEL в регистровой паре HL(BX). При возврате переменной метки, которая может быть целью операции GOTO, подпрограмма, содержащая метку, должна восстановить стек к надлежащему уровню, когда управление достигает метки.

PL/I-86 R1.1 возвращает переменные ENTRY и LABEL в стеке как 8 непрерывных байтов. Байт младшего разряда находится наверху стека.

┌──────────┬──────────────┬─────────────┬──────┐

│ Смещение │ Сегмент кода │ Фрейм стека │ Стек │ (низкий стек) ─>

└──────────┴──────────────┴─────────────┴──────┘

↑

SP

Следующие листинги программ иллюстрируют концепцию возврата значения функции. *Листинг 6* показывает программу под названием FDTEST, которая подобна предыдущему тесту деления с плавающей точкой. Однако FDTEST включает определение точки входа для подпрограммы ассемблера под названием FDIV2, которая возвращает результат в стеке. *Листинг 7* показывает FDIV2.ASM на ассемблере 8080 и *Листинг 8* показывают FDIV2.A86, ту же подпрограмма на ассемблере 8086.

FDIV2 напоминает предыдущую подпрограмму DIV2 с некоторыми незначительными изменениями. Во-первых, FDIV2 загружает входные значение с плавающей точкой в регистры BC(CX) и DE(DX) так, чтобы она могла управлять временной копией и не затрагивать исходное входное значение. FDIV2 затем уменьшает поле экспоненты в регистре B(CH) на входной счетчик и возвращает его в стек прежде, чем выполнить инструкцию PCHL.

1 a /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

2 a /\* This program tests the assembly-language routine \*/

3 a /\* called FDIV2 which returns a FLOAT BINARY value. \*/

4 a /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

5 a fdtest:

6 b procedure options(main);

7 b declare

8 b fdiv2 entry(fixed(7),float) returns(float),

9 b i fixed(7),

10 b f float;

11 b

12 c do i = 0 by 1;

13 c put skip list('100 / 2 \*\*',i,'=',fdiv2(i,100));

14 c end;

15 b

16 b end fdtest;

Листинг 6 Программа FDTEST

title 'div by power of two (function)'

public fdiv2

extrn ?signal

; entry:

; p1 -> fixed(7) power of two

; p2 -> floating point number

; exit:

; p1 -> (unchanged)

; p2 -> (unchanged)

; stack: p2 / (2 \*\* p1)

fdiv2: ;HL = .low(.p1)

mov e,m ;low(.p1)

inx h ;HL = .high(.p1)

mov d,m ;DE = .p1

inx h ;HL = .low(p2)

ldax d ;a = p1 (power of two)

mov e,m ;low(.p2)

inx h ;HL = .high(.p2)

mov d,m ;DE = .p2

xchg ;HL = .p2

;

; A = power of 2, HL = .low byte of fp num

mov e,m ;E = low mantissa

inx h ;to middle of mantissa

mov d,m ;D = middle mantissa

inx h ;to high byte of mantissa

mov c,m ;C = high mantissa

inx h ;to exponent byte

mov b,m ;B = exponent

inr b ;B = 00?

dcr b ;becomes 00 if so

jz fdret ;to return from float div

dby2: ;divide by two

ora a ;counted power of 2 to zero?

jz fdret ;return if so

dcr a ;count power of two down

dcr b ;count exponent down

jnz dby2 ;loop again if no underflow

;

;underflow occurred, signal underflow condition

lxi h,siglst;signal parameter list

call ?signal ;signal underflow

lxi b,0 ;clear to zero

lxi d,0 ;for default return

;

fdret: pop h ;recall return address

push b ;save high order fp num

push d ;save low order fp num

pchl ;return to calling routine

;

dseg

siglst: dw sigcod ;address of signal code

dw sigsub ;address of subcode

dw sigfil ;address of file code

dw sigaux ;address of aux message

; end of parameter vector, start of params

sigcod: db 3 ;03 = underflow

sigsub: db 128 ;arbitrary subcode for id

sigfil: dw 0000 ;no associated file name

sigaux: dw undmsg ;0000 if no aux message

undmsg: db 32,'Underflow in Divide by Two',0

end

Листинг 7 FDIV2.ASM Программа на языке ассемблера (8080)

; Division by power of two (function)

cseg

public fdiv2

extrn ?signal:near

; entry:

; p1 -> fixed(7) power of two

; p2 -> floating point number

; exit:

; p1 -> (unchanged)

; p2 -> (unchanged)

; stack: p2 / (2 \*\* p1)

fdiv2: ;BX = .low(.p1)

mov si,[bx] ;SI = .p1

lods al ;AL = p1 (power of 2)

mov bx,2[bx] ;BX = .p2

; AL = power of 2, BX = .low byte of fp num

mov dx,[bx] ;DX = low and middle mantissa

mov cx,2[bx] ;CL = high mantissa, CH = exponent

test cx,7f80h ;exponent zero?

jz fdret ;to return from float div

dby2: ;divide by two

test al,al ;counted power of 2 to zero?

jz fdret ;return if so

dec al ;count power of two down

sub cx,80h ;count exponent down

test cx,7f80h ;test for underflow

jnz dby2 ;loop again if no underflow

; Underflow occurred, signal underflow condition

mov bx,offset siglst;signal parameter list

call ?signal ;signal underflow

sub cx,cx ;clear result to zero for default return

mov dx,cx

fdret: pop bx ;recall return address

push cx ;save high order fp num

push dx ;save low order fp num

jmp bx ;return to calling routine

dseg

siglst dw offset sigcod ;address of signal code

dw offset sigsub ;address of subcode

dw offset sigfil ;address of file code

dw offset sigaux ;address of aux message

; end of parameter vector, start of params

sigcod db 3 ;03 = underflow

sigsub db 128 ;arbitrary subcode for id

sigfil dw 0000 ;no associated file name

sigaux dw offset undmsg ;0000 if no aux message

undmsg db 32,'Underflow in Divide by Two',0

end

Листинг 8 FDIV2.A86 Программа на языке ассемблера (8086)

**П3.3 Прямые вызовы функции операционной системы**

Вы можете иметь прямой доступ ко всем функциям операционной системы с помощью дополнительных подпрограмм на языке ассемблера, которые присутствуют в исходном виде на вашем диске с примерами программ PL/I. Диск с примерами программ также содержит файл RELNOTES.PRN, который описывает эти программы на языке ассемблера и несколько программ на языке PL/I, которые тестируют различные вызовы функций.

Подпрограммы в этих программах не включены в стандартную библиотеку PLILIB, поскольку конкретные приложения могут потребовать изменения системных функций, либо удалить некоторые операции, чтобы уменьшить пространство или изменить интерфейс к конкретной функции. Если интерфейс к функции изменяется, вы должны изменить точку входа, чтобы избежать недоразумений.

**Конец приложения 3**

Приложение 4 Опции компилятора

*Таблица П4-1* перечисляет опции компилятора и дает краткое описание их использования. В каждом случае однобуквенная опция следует за символом $ в командной строке. Вы можете определить максимум семь опций после знака доллара. Режим по умолчанию, не используя опции, компилирует программу, но не производит исходный листинг и отправляет все сообщения об ошибках на консоль.

Таблица П4-1 Опции компилятора PL/I

|  |  |
| --- | --- |
| **Опция** | **Включенное действие** |
| **A** | *Сокращенный листинг*. Отключает листинг параметров и листинг операторов %INCLUDE во время первого прохода компилятора. |
| **B** | *Встроенная трассировка подпрограммы*. Во время выполнения показывает библиотечные функции, которые вызываются вашей программой PL/I. |
| **D** | *Печать файла на диск*. Отправляет листинг файла на диск, используя тип файла PRN. |
| **I** | *Чередование исходного и машинного кода*. Декодирует машинный код, произведенный компилятором в форме псевдо-ассемблера. |
| **K** | Аналогична A (реализации 8080) |
| **L** | *Листинг исходной программы*. Производит листинг исходной программы с номерами строк и размещением машинного кода (автоматически устанавливается переключателем I). |
| **N** | *Отображение уровня вложенности*. Включает на 1 проходе трассировку, которая показывает точный баланс операторов DO, PROCEDURE и BEGIN с соответствующими им операторами END. |
| **O** | *Отмена объектного кода*. Отключает вывод перемещаемого объектного кода, обычно создаваемого компилятором. |
| **P** | *Режим печати страниц*. Вставляет символы прогона страницы через каждые 60 строк и отправляет листинг на принтер. |
| **S** | *Отображение таблицы символов*. Показывает имена переменных программы, вместе с их присвоенными, назначенными по умолчанию и расширенными атрибутами. |

**Конец приложения 4**

Приложение 5 Сообщения об ошибках и коды состояний

PL/I может обнаружить два вида ошибок: ошибки компиляции и ошибки периода выполнения. Компилятор отмечает каждую ошибку компиляции с помощью символа ? около позиции ошибки в строке и выводит сообщение об ошибке после строки, содержащей ошибку. Символ ? может отстоять от позиции фактической ошибки на несколько столбцов. В некоторых случаях ошибка в одной строке может привести к ошибкам в последующих строках.

PL/I подразделяет ошибки на восстанавливаемые и невосстанавливаемые. Большинство ошибок компиляции восстанавливаемые, и компилятор продолжает обрабатывать исходный файл. Однако некоторые ошибки компиляции невосстанавливаемые. Компилятор прекращает обработку, и управление сразу возвращается в операционную систему.

Система времени выполнения обнаруживает ошибки во время выполнения программы. Большинство ошибок периода выполнения восстанавливаемые, если перехвачены ON-блоком. Однако некоторые ошибки периода выполнения невосстанавливаемые. Программа завершается, и управление сразу возвращаются в операционную систему.

Это приложение перечисляет сообщения об ошибках, которые появляются в каждой реализации. Ошибки перечислены в следующем порядке:

* Общие ошибки
* Ошибки компиляции (при проходе)
* Времени выполнения

**Примечание**: все невосстанавливаемые ошибки отмечены звездочкой.

**П5.1 PL/I-80 R1.4 и PL/I-86 R1.0**

Таблица П5-1 Общие ошибки

|  |  |
| --- | --- |
| **Ошибка** | **Описание** |
| **DIR FULL\*** | Отсутствует доступное место в дисковом каталоге операционной системы. Вы должны стереть все ненужные файлы и попробовать еще раз. |
| **DISK FULL\*** | Отсутствует доступное место для файлов на диске. Вы должны стереть все ненужные файлы и попробовать еще раз. |
| **INVALID INCLUDE** | В операторе %INCLUDE имеется синтаксическая ошибка. Оператор %INCLUDE имеет общий формат %include 'd:filename.typ';  где d - (необязательный) диск, и filename.typ - спецификация файла. |
| **LENGTH** | Элемент превышает максимальную ширину поля для ключевого слова или элемента данных (31 символ для идентификаторов, 128 для строк). |
| **NO FILE x \*** | Файл x отсутствует на диске. Если x имеет тип PLI, убедитесь, что ваш исходный файл находится на названном диске. Если тип - OVR или OVL, то убедитесь, что все три оверлея компилятора PL/I (PLI0, PLI1, PLI2) находятся на диске по умолчанию. |
| **OUT OF MEMORY** | Размер области транзитных программ (TPA) вашей системы слишком маленький. Вы должны реконфигурировать систему. |
| **READ ONLY X \*** | PL/I не может закрыть файл, названный x. Обычно это происходит с диском, установленным в состояние только для чтения с помощью аппаратных средств. |

|  |  |
| --- | --- |
| **Ошибка** | **Описание** |
| **TERMINATED.\*** | Число ошибок компиляции превышает 255, или компиляция была завершена пользователем в консоли. |
| **TRUNC** | Длина строки превышает 120 символов, и была усечена. |
| **UNEXPECTED EOF\*** | Компилятор встретился с концом исходной программы до логического конца программы. Обычно эта проблема связана с нарушением баланса уровней блоков (перекомпилируйте с опцией $n для включения трассировки), или несбалансированными комментариями и строками (проверьте баланс недостающих \*/ или символов апострофа). |
| **VALUE** | Указывает, что преобразованное число превышает 16-разрядную емкость констант FIXED BINARY (-32768, +32767). |

Таблица П5-2 Ошибки компиляции

| **Ошибка** | **Описание** |
| --- | --- |
| **BAD VAL** | Константа, обнаруженная в формате, недопустима для этого элемента формата. |
| **BALANCE** | Левые и правые круглые скобки в выражении не сбалансированы. |
| **BLOCK AT LINE x VARIABLE v EXCEEDS STORAGE** | |
|  | Блок, начинающийся в исходной строке x, содержит переменную v, которая приводит к суммарному выделению области памяти превышающей 65535 байтов. |
| **BLOCK OVERFLOW** | Уровень вложенности блоков PROCEDURE, DO и BEGIN превышает тридцать один уровень. Вы должны упростить структуру программы и попробовать еще раз. |
| **CONFLICT** | Атрибуты данных в операторе DECLARE конфликтуют друг с другом. |
| **DUPLIC** | Обозначенная переменная в этом блоке объявлена несколько раз. |
| **LABEL** | Метка для этого оператора сформирована не правильно. Допускается только одна метка для каждого оператора, константа индексируемой метки должны иметь постоянные индекс в виде константы. |
| **LENGTH** | Длина обозначенного символа превышает максимальный размер символа. Вы должны упростить структуру и попробовать еще раз. Эта ошибка может также быть вызвана несбалансированной строкой. |
| **NESTED REP** | Оператор %REPLACE неправильно помещен в блочную структуру. Все операторы %REPLACE должны быть размещены на внешнем блочном уровне до появления вложенных внутренних блоков. |
| **NO DCL: v1, v2, ... vn** | |
|  | Перечисленные параметры процедуры присутствуют в заголовке процедуры, но не объявлены в теле процедуры. |
| **NOT BIF** | Атрибут BUILTIN применен к идентификатору, который не является встроенной функцией PL/I. |
| **NOT IMP** | Оператор использует функцию, которая не реализована в PL/I. |
| **NOT VARIABLE** | Объявленное имя обработано как переменная, но не имеет атрибута VARIABLE. |
| **NUMBER** | В этой позиции в формате требуется числовая константа. |
| **ON BODY** | Недопустимый оператор в теле ON-состояния. Вы не можете использовать оператор RETURN, для выхода из ON-блока. Операторы DO и IF требуют включение блока в BEGIN...END. |
| **PICTURE** | Есть синтаксическая ошибка в спецификации шаблона или элементе формата P. |
| **RECUR PROC** | Рекурсивная процедура содержит недопустимый вложенный блок. В рекурсивных процедурах разрешены только встроенные DO-группы. |
| **STRUCTURE** | Обозначенная структура неправильно сформирована. Уровни вложенности не могут превышать 255. |
| **SYMBOL LENGTH OVERFLOW** | |
|  | Максимальный размер символа превышен во время построения записи таблицы символов. Вы должны упростить программу и попробовать еще раз. |
| **SYMBOL TABLE OVERFLOW \*** | |
|  | Эта программа не может быть скомпилирована с текущим объемом памяти. Вы должны разбить модуль на раздельные компиляции или увеличить размер TPA в вашей системе. |
| **SYNTAX** | В указанном операторе имеется синтаксическая ошибка. Посмотрите соответствующий раздел справочника языка PL/I для уточнения правильного синтаксиса. |
| **Ошибки 2 прохода** | |
| **AGG VAL** | Фактический параметр является агрегатным значением, которое не совпадает с формальным параметром. Измените фактический или формальный параметр, для устранения несоответствия. |
| **ARG COUNT** | Происходит одна из следующих ошибок: индекс не соответствует объявлению; имеется объявленная ссылка на элемент массива; имеется больше 15 граничных пар; некоторые граничные пары не соответствуют; или количество формальных параметров не соответствует числу фактических. |
| **BASE** | Присутствует недопустимая ссылка базированной переменной. Это может произойти, когда спецификатор указателя ссылается на небазированную переменную, или когда переменная объявлена BASED(x), где x не указатель простой переменной или не вызов функции возвращающей простой указатель, как в BASED(p) или BASED(q()). |
| **BASED REQ** | В этом контексте требуется базированная переменная. |
| **BAD TYPE** | Недопустимая управляющая переменная в итеративной DO-группе. Допускаются только скалярные переменные. |
| **BAD VALUE** | Имеется недопустимый аргумент встроенной функции. |
| **BALANCE** | Левые и правые круглые скобки для этого выражения несбалансированные. |
| **BIT CON** | Константа битовой подстроки вне диапазона. Третий аргумент битовой SUBSTR должен быть константой в диапазоне 1-16. |
| **BIT REQ** | В этом контексте требуется битовое выражение. |

| **Ошибка** | **Описание** |
| --- | --- |
| **CLOSURE** | Метка после ключевого слова END не соответствует имени соответствующего блока. |
| **COMP REQ** | Используется невычислительное выражение там, где требуется вычислительное выражение. |
| **COMPILER** | Произошла ошибка компилятора. Ошибка могла произойти из-за предыдущих ошибок. |
| **CONFLICT** | Атрибуты данных находятся в конфликте, или не совместимые атрибуты в операторе OPEN. |
| **CONVERT** | Компилятор не может преобразовать константу в требуемый тип. |
| **EXPRESSION OVERFLOW \*** | |
|  | Выражение переполняет внутренние структуры компилятора. Упростите программу и попробуйте снова. |
| **ID REQ** | В этом контексте требуется идентификатор. |
| **INT REQ** | В этом контексте требуется целое (FIXED BINARY) выражение. |
| **LABEL** | Метка неправильной формы в месте, где ожидается метка. |
| **NO BUILTIN** | Встроенная функция, на которую ссылаются, не реализована в PL/I. |
| **NO DCL** | Обозначенная переменная не объявлена в области действия этой ссылки. |
| **NOT FILE** | Ссылка в опции FILE не файловая переменная или файловая константа. |
| **NOT FORMAT** | Поле формата оператора GET или PUT EDIT не ссылается на формат. |
| **NOT IMP** | Конструкция в этом операторе не реализована в PL/I. |
| **NOT KEY** | Выражение в опции KEYTO, KEYFROM или KEY не является переменной FIXED BINARY. |
| **NOT LABEL** | Цель этого оператора GOTO не является меткой. |
| **NOT PROC** | Ссылка, расположенная за ключевым словом CALL, не является процедурой. |
| **NOT SCALAR** | Нескалярное значение возникает в контексте, требующем скалярное выражение. |
| **NOT STATIC** | Предпринята попытка инициализации автоматического хранения. Необходимо объявить переменную с атрибутом STATIC и попробовать еще раз. |
| **PTR REQ** | В этом контексте требуется переменная указателя. |
| **QUALIFY** | Эта ссылка на структуру не определяет имя переменной должным образом. Обычно это происходит из-за неуникальной ссылки подструктуры. |
| **RET EXP** | Выражение в операторе RETURN не совместимо с атрибутом RETURNS соответствующей процедуры. |
| **RETURN** | Предпринята попытка вернуть значение из процедуры без атрибута RETURNS. |
| **SYNTAX** | В этом операторе имеется синтаксическая ошибка. Посмотрите соответствующий раздел справочника языка PL/I для уточнения правильного синтаксиса. |

| **Ошибка** | **Описание** |
| --- | --- |
| **SCALE GREATER THAN 0** | |
|  | Получающееся выражение FIXED BINARY производит ненулевой масштабный коэффициент. Если выражение включает в себя деление, необходимо заменить x/y на DIVIDE(x,y,0). Эта замена необходима для сохранения совместимости с полной версией языка. |
| **SYMBOL TABLE OVERFLOW \*** | |
|  | Во время компиляции исчерпана свободная память. (См. аналогичную ошибку в проходе 1.) |
| **STR REQ** | В этом контексте требуется строковая переменная. В случае встроенной функции SUBSTR, необходимо присвоить выражение временной переменной до операции с подстрокой. |
| **TYPES NOT=** | Типы бинарной операции не совместимы. Вы можете проверить все объявления и ознакомиться с правилами преобразования ([*Раздел 4*](#_Преобразования_данных_4)). Эта ошибка может произойти из-за элементов агрегированных данных, которые не совпадают по структуре. |
| **UNSPEC** | Источник или цель операции UNSPEC не 8 или 16 разрядная переменная. |
| **# VALUES** | Число элементов, определенных в операторе INITIAL, не совместимо с инициализируемой переменной. |
| **VAR REQ** | В этом контексте требуется переменная. |
| **Ошибки 3 прохода** | |
| **\*\*\*AUTOMATIC STORAGE OVERFLOW\*\*\*** | |
|  | Общее хранение, определенное в этом программном модуле, превышает 65535 байтов. |
| **BAD INT FILE** | Промежуточный файл, отправленный в проход 3, является недействительным. Как правило, это случается из-за сбоя в работе оборудования. |
| **BLOCK OVERFLOW** | Уровень вложенности превысил внутренние таблицы компилятора (максимум 32 уровня). |
| **EOF ON INT FILE** | Компилятор встречается с преждевременным концом файла при чтении промежуточного файла. Эта ошибка обычно происходит из-за отказа оборудования. |
| **EXPRESSION OVERFLOW \*** | |
|  | Превышены размеры внутренних структур компилятора. Необходимо упростить выражение и попробовать еще раз. |
| **LINE x OPERATION NOT IMPLEMENTED** | |
|  | Происходит недопустимая промежуточная операция. Эта ошибка обычно происходит из-за отказа оборудования или ошибок в предыдущих проходах. |

Таблица П5-3 Ошибки времени выполнения

| **Ошибка** | **Описание** |
| --- | --- |
| **Невосстанавливаемые ошибки времени выполнения** | |
| **FREE REQUEST OUT OF RANGE** | |
|  | Оператор FREE определяет адрес памяти вне диапазона свободной области хранения. Это обычно происходит при ссылке на неинициализированный указатель базы. |
| **FREE SPACE OVERWRITE** | Свободная область хранения перезаписана. Обычно эта ошибка возникает из-за индексной ссылки вне диапазона или переполнения стека. Если происходит переполнение стека, используйте ключевое слово STACK(n) в поле OPTIONS, чтобы увеличить размер стека и попробуйте еще раз. |
| **INSUFFICIENT MEMORY** | Загруженная программа не может работать в выделенном объеме памяти. Если возможно, увеличьте размер области транзитных программ. |
| **INVALID I/O LIST** | Во время выполнения программы перезаписан список активных файлов, и попытка закрыть все активные файлы неудачна. Обычно она возникает при значениях индексов вне диапазона. |
| **Восстанавливаемые ошибки времени выполнения** | |
| PL/I распечатывает следующие ошибки, если не включен ON-блок или, если управление возвращается из соответствующего ON-блока по невосстанавливаемому состоянию (отмеченному звездочкой). В каждом случае, выводится префикс состояния, за которым следует дополнительный подкод, идентифицирующий источник ошибки, за которым в некоторых случаях следует вспомогательное сообщение, которое дополнительно идентифицирует источник ошибки. | |
| **ERROR(1) "Conversion"** | Эта ошибка происходит каждый раз, когда система время выполнения не может выполнить требуемое преобразование между типами данных. Эта ошибка может быть сообщена во время арифметических операций, присвоений и обработки ввода-вывода операторами GET и PUT. |
| **ERROR(2) "I/O Stack Overflow"** | |
|  | Во время выполнения одновременных, вложенных операций ввода-вывода стек ввода-вывода превышает 16. Необходимо упростить программу и попробовать еще раз. |
| **ERROR(3)** | Параметр трансцендентной функции вне диапазона. |
| **ERROR(4) "I/O Conflict x"** | |
|  | Файл явно или неявно открыт с одним набором атрибутов, и впоследствии получен доступ с помощью оператора, требующего конфликтующие атрибуты. Значение x является одним из следующих:  • STREAM/RECORD • INPUT/OUTPUT  • SEQUEN/DIRECT • KEYED Access  Первый конфликт возникает, когда файлы ASCII обрабатываются, используя READ или WRITE, но опция INTO или FROM не определяет переменную символьную строку. |
| **ERROR(5) "Format Overflow"** | |
|  | Уровень вложенности встроенных форматов превышает 32. Необходимо упростить программу и попробовать еще раз. |
| **ERROR(6) "Invalid Format Item"** | |
|  | Процессор формата обнаружил элемент формата, который не может быть обработан. Формат P не реализован в PL/I. |
| **ERROR(7) "Free Space Exhausted"** | |
|  | Нет больше свободного места. Если вы перехватываете эту ошибку с помощью ON-блока, не выполняйте ALLOCATE, OPEN или рекурсию без освобождения памяти. |
| **ERROR(8) "OVERLAY, NO FILE d:filename"** | |
|  | Менеджер оверлея не может найти обозначенный файл. |
| **ERROR(9) "OVERLAY, DRIVE d:filename"** | |
|  | В качестве параметра к оверлею передан недопустимый код диска. |
| **ERROR(10) "OVERLAY, SIZE d:filename"** | |
|  | Обозначенный оверлей слишком большой и перезаписывает при загрузке стековое и/или свободное пространство PL/I. |
| **ERROR(11) "OVERLAY, NESTING d:filename"** | |
|  | Загрузка обозначенного оверлея превышает максимальную глубину вложения. |
| **ERROR(12) "OVERLAY, READ d:filename"** | |
|  | Была ошибка чтения с диска при загрузке оверлея. Это, вероятно, вызвано преждевременным EOF. |
| **ERROR(13) "Invalid OS Version"** | |
|  | Любая операция, генерирующая вызов операционной системы, не поддерживаемый текущей операционной системой, вызывает эту ошибку. |
| **ERROR(14) "Unsuccessful Write"** | |
|  | Любая неуспешная операция записи в файл из-за отсутствия места в каталоге, отсутствия места на диске, и т.д., вызывает эту ошибку. |
| **ERROR(15) "File Not Open"** | |
|  | Любая попытка заблокировать или разблокировать запись в неоткрытом файле вызывает эту ошибку. |
| **ERROR(16) "File Not Keyed"** | |
|  | Любая попытка заблокировать или разблокировать запись в файле, у которого отсутствует атрибут KEYED, вызывает эту ошибку. |
| **FIXEDOVERFLOW** | Десятичная операция дает значение более 15 десятичных цифр точности, или попытка сохранить в переменную с недостаточной точностью. |

| **Ошибка** | **Описание** |
| --- | --- |
| **OVERFLOW(1)** | Операции с плавающей точкой производит значение, слишком большое, чтобы быть представленным в формате с плавающей точкой. |
| **OVERFLOW(2)** | Значение с плавающей точкой двойной точности присвоено значению с недостаточной точностью одинарной точности. |
| **UNDERFLOW(1)** | Операция с плавающей точкой производит значение, слишком маленькое, чтобы быть представленной в формате с плавающей точкой. |
| **UNDERFLOW(2)** | Значение с плавающей точкой двойной точности присвоено значению с недостаточной точностью одинарной точности. |
| **ZERODIVIDE(1)** | Предпринята операция деления десятичных чисел или модуля с нулевым делителем. |
| **ZERODIVIDE(2)** | Предпринята операция деления или модуля с плавающей точкой с нулевым делителем. |
| **ZERODIVIDE(3)** | Предпринята операция целочисленного деления или модуля с нулевым делителем. |
| **ENDFILE** | Сделана попытка чтения после конца файла, или во время вывода возникает состояние полного диска. |
| **UNDEFINEDFILE** | Если эта ошибка происходит при вводе, система времени выполнения не может найти именованный файл на диске, или устройство ввода данных открыто для вывода. Если ошибка происходит при выводе, система времени выполнения не может создать выходной файл, или устройство вывода открыто для ввода. |
| **KEY(1)** | Обнаружен недопустимый ключ в операции вывода. |
| **KEY(2)** | Обнаружен недопустимый ключ во время операции ввода. |
| **ENDPAGE** | Обнаружено состояние конца страницы. Это состояние не вызывает завершение программы, если нет активного ON-блока. |

**П5.2 PL/I-86 R1.1 и PL/I-86 R1.0 для DOS**

В PL/I-86 R1.1 и PL/I-86 R1.0 для DOS сообщения об ошибках компиляции на 3 проходе и сообщения об ошибках периода выполнения идентичны сообщениям в PL/I-80 R1.4 и PL/I-86 R1.0. Однако существуют новые сообщения об ошибках в проходе 1 и проходе 2. Текст новых сообщений об ошибках делает их очевидными и не требует пояснений.

Механизм нахождения и создания отчетов об ошибках является аналогичным. Т.е. компилятор отмечает каждую ошибку компиляции с помощью символа ? около позиции ошибки в строке и выводит сообщение об ошибке после строки, содержащей ошибку. Символ ? может отстоять от позиции фактической ошибки на несколько столбцов. В некоторых случаях ошибка в одной строке может привести к ошибкам в последующих строках.

**П5.3 Категории состояний и коды**

Категории состояний описывают различные состояния, которые может устанавливать система времени выполнения или ваша программа, выполняя оператор SIGNAL.

Существует девять категорий основных состояний с подкодами, часть из которых определены системой, и некоторые вы можете определить самостоятельно. *Таблица П5-4* приводит предопределенные подкоды.

Таблица П5-4 Категории состояний PL/I и подкоды

|  |  |
| --- | --- |
| **Тип** | **Значение** |
| **ERROR** |  |
| ERROR(0) | Любой подкод ERROR |
| ERROR(1) | Преобразование данных |
| ERROR(2) | Переполнение стека ввода-вывода |
| ERROR(3) | Недопустимый аргумент функции |
| ERROR(4) | Конфликт ввода-вывода |
| ERROR(5) | Переполнение стека формата |
| ERROR(6) | Недопустимый элемент формата |
| ERROR(7) | Исчерпание свободной памяти |
| ERROR(8) | Ошибка оверлея, нет файла |
| ERROR(9) | Ошибка оверлея, недопустимый диск |
| ERROR(10) | Ошибка оверлея, размер |
| ERROR(11) | Ошибка оверлея, размещение |
| ERROR(12) | Ошибка оверлея, ошибка чтения диска |
| ERROR(13) | Недопустимый вызов ОС |
| ERROR(14) | Неудачная запись |
| ERROR(15) | Файл не открыт |
| ERROR(16) | Файл не с ключом |
| **FIXEDOVERFLOW** |  |
| FIXEDOVERFLOW(0) | Любой подкод FIXEDOVERFLOW |
| **OVERFLOW** |  |
| OVERFLOW(0) | Любой подкод OVERFLOW |
| OVERFLOW(1) | Операция с плавающей точкой |
| OVERFLOW(2) | Преобразование точности с плавающей точкой |
| **UNDERFLOW** |  |
| UNDERFLOW(0) | Любой подкод UNDERFLOW |
| UNDERFLOW(1) | Операция с плавающей точкой |
| UNDERFLOW(2) | Преобразование точности с плавающей точкой |
| **ZERODIVIDE** |  |
| ZERODIVIDE(0) | Любой подкод ZERODIVIDE |
| ZERODIVIDE(1) | Десятичное деление |
| ZERODIVIDE(2) | Деление с плавающей точкой |
| ZERODIVIDE(3) | Целочисленное деление |
| **ENDFILE** |  |
| **UNDEFINEDFILE** |  |
| **KEY** |  |
| **ENDPAGE** |  |

**Конец приложения 5**

Приложение 6 Коды ASCII

ASCII обозначает американский стандартный код для обмена информацией. Код содержит 96 печатаемых и 32 непечатаемых символа используемых для хранения данных на диске. *Таблица П6-1* определяет символы ASCII, далее *Таблица П6-2* перечисляет ASCII и шестнадцатеричные преобразования. Таблица содержит двоичное, десятичное, шестнадцатеричное и ASCII преобразования.

Таблица П6-1 Символы ASCII

|  |  |  |  |
| --- | --- | --- | --- |
| **Символ** | **Значение** | **Символ** | **Значение** |
| ACK | Подтверждение | FS | Разделитель файлов |
| BEL | Звонок | GS | Разделитель групп |
| BS | Возврат на один символ | HT | Горизонтальная табуляция |
| CAN | Отмена | LF | Перевод строки |
| CR | Возврат каретки | NAK | Отсутствие подтверждения |
| DC | Контроль устройства | NUL | Нет данных |
| DEL | Удаление | RS | Разделитель записей |
| DLE | Переключение | SI | Режим обычного ASCII |
| EM | Конец носителя | SO | Режим национальных символов |
| ENQ | Запрос | SOH | Начало заголовка |
| EOT | Конец передачи | SP | Пробел |
| ESC | Управляющая последовательность | STX | Начало текста |
| ETB | Конец блока передачи | SUB | Подставить |
| ETX | Конец текста | SYN | Синхронизация |
| FF | Перевод страницы | US | Разделитель элементов |
|  |  | VT | Вертикальная табуляция |

Таблица П6-2 Таблица преобразования ASCII

| **Binary** | **Decimal** | **Hexadecimal** | **ASCII** | **Binary** | **Decimal** | **Hexadecimal** | **ASCII** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0000000 | 0 | 00 | NUL | 1000000 | 64 | 40 | @ |
| 0000001 | 1 | 01 | SOH (CTRL-A) | 1000001 | 65 | 41 | A |
| 0000010 | 2 | 02 | STX (CTRL-B) | 1000010 | 66 | 42 | B |
| 0000011 | 3 | 03 | ETX (CTRL-C) | 1000011 | 67 | 43 | C |
| 0000100 | 4 | 04 | EOT (CTRL-D) | 1000100 | 68 | 44 | D |
| 0000101 | 5 | 05 | ENQ (CTRL-E) | 1000101 | 69 | 45 | E |
| 0000110 | 6 | 06 | ACK (CTRL-F) | 1000110 | 70 | 46 | F |
| 0000111 | 7 | 07 | BEL (CTRL-G) | 1000111 | 71 | 47 | G |
| 0001000 | 8 | 08 | BS (CTRL-H) | 1001000 | 72 | 48 | H |
| 0001001 | 9 | 09 | HT (CTRL-I) | 1001001 | 73 | 49 | I |
| 0001010 | 10 | 0A | LF (CTRL-J) | 1001010 | 74 | 4A | J |
| 0001011 | 11 | 0B | VT (CTRL-K) | 1001011 | 75 | 4B | K |
| 0001100 | 12 | 0C | FF (CTRL-L) | 1001100 | 76 | 4C | L |
| 0001101 | 13 | 0D | CR (CTRL-M) | 1001101 | 77 | 4D | M |
| 0001110 | 14 | 0E | SO (CTRL-N) | 1001110 | 78 | 4E | N |
| 0001111 | 15 | 0F | SI (CTRL-0) | 1001111 | 79 | 4F | O |
| 0010000 | 16 | 10 | DLE (CTRL-P) | 1010000 | 80 | 50 | P |
| 0010001 | 17 | 11 | DCI (CTRL-Q) | 1010001 | 81 | 51 | Q |
| 0010010 | 18 | 12 | DC2 (CTRL-R) | 1010010 | 82 | 52 | R |
| 0010011 | 19 | 13 | DC3 (CTRL-S) | 1010011 | 83 | 53 | S |
| 0010100 | 20 | 14 | DC4 (CTRL-T) | 1010100 | 84 | 54 | T |
| 0010101 | 21 | 15 | NAK (CTRL-U) | 1010101 | 85 | 55 | U |
| 0010110 | 22 | 16 | SYN (CTRL-V) | 1010110 | 86 | 56 | V |
| 0010111 | 23 | 17 | ETB (CTRL-W) | 1010111 | 87 | 57 | W |
| 0011000 | 24 | 18 | CAN (CTRL-X) | 1011000 | 88 | 58 | X |
| 0011001 | 25 | 19 | EM (CTRL-Y) | 1011001 | 89 | 59 | Y |
| 0011010 | 26 | 1A | SUB (CTRL-Z) | 1011010 | 90 | 5A | Z |
| 0011011 | 27 | 1B | ESC (CTRL-[) | 1011011 | 91 | 5B | [ |
| 0011100 | 28 | 1C | FS (CTRL-\) | 1011100 | 92 | 5C | \ |
| 0011101 | 29 | 1D | GS (CTRL-]) | 1011101 | 93 | 5D | ] |
| 0011110 | 30 | 1E | RS (CTRL-^) | 1011110 | 94 | 5E | ^ |
| 0011111 | 31 | 1F | US (CTRL-\_) | 1011111 | 95 | 5F | \_ |
| 0100000 | 32 | 20 | (Пробел) | 1100000 | 96 | 60 | ` |
| 0100001 | 33 | 21 | ! | 1100001 | 97 | 61 | a |
| 0100010 | 34 | 22 | " | 1100010 | 98 | 62 | b |
| 0100011 | 35 | 23 | # | 1100011 | 99 | 63 | c |
| 0100100 | 36 | 24 | $ | 1100100 | 100 | 64 | d |
| 0100101 | 37 | 25 | % | 1100101 | 101 | 65 | e |
| 0100110 | 38 | 26 | & | 1100110 | 102 | 66 | f |
| 0100111 | 39 | 27 | ' | 1100111 | 103 | 67 | g |
| 0101000 | 40 | 28 | ( | 1101000 | 104 | 68 | h |
| 0101001 | 41 | 29 | ) | 1101001 | 105 | 69 | i |
| 0101010 | 42 | 2A | \* | 1101010 | 106 | 6A | j |
| 0101011 | 43 | 2B | + | 1101011 | 107 | 6B | k |
| 0101100 | 44 | 2C | , | 1101100 | 108 | 6C | l |
| 0101101 | 45 | 2D | - | 1101101 | 109 | 6D | m |
| 0101110 | 46 | 2E | . | 1101110 | 110 | 6E | n |
| 0101111 | 47 | 2F | / | 1101111 | 111 | 6F | o |
| 0110000 | 48 | 30 | 0 | 1110000 | 112 | 70 | p |
| 0110001 | 49 | 31 | 1 | 1110001 | 113 | 71 | q |
| 0110010 | 50 | 32 | 2 | 1110010 | 114 | 72 | r |
| 0110011 | 51 | 33 | 3 | 1110011 | 115 | 73 | s |
| 0110100 | 52 | 34 | 4 | 1110100 | 116 | 74 | t |
| 0110101 | 53 | 35 | 5 | 1110101 | 117 | 75 | u |
| 0110110 | 54 | 36 | 6 | 1110110 | 118 | 76 | v |
| 0110111 | 55 | 37 | 7 | 1110111 | 119 | 77 | w |
| 0111000 | 56 | 38 | 8 | 1111000 | 120 | 78 | x |
| 0111001 | 57 | 39 | 9 | 1111001 | 121 | 79 | y |
| 0111010 | 58 | 3A | : | 1111010 | 122 | 7A | z |
| 0111011 | 59 | 3B | ; | 1111011 | 123 | 7B | { |
| 0111100 | 60 | 3C | < | 1111100 | 124 | 7C | | |
| 0111101 | 61 | 3D | = | 1111101 | 125 | 7D | } |
| 0111110 | 62 | 3E | > | 1111110 | 126 | 7E | ~ |
| 0111111 | 63 | 3F | ? | 1111111 | 127 | 7F | Rubout |

**Конец приложения 6**

Приложение 7 Библиография PL/l

Это приложение перечисляет несколько справочников по программированию на языке PL/I. Некоторые из них являются учебниками начального уровня для использования в классе, в то время как другие - руководства повышенного уровня сложности. Каждая ссылка сопровождается кратким описанием общего содержания. Вы можете приобрести эти книги через свой местный книжный магазин или заказать их непосредственно у издателя.

Несмотря на то, что теперь есть книги, специально написанные для PL/I подмножества G, книги, перечисленные здесь, описывают такие подмножества, как PL/C и SP/k**™** или полные реализации PL/I IBM**™**. Формы операторов PL/C и SP/k обычно включены в определения подмножества G, в то время как полный язык PL/I содержит много средств, исключенных из подмножества. Поэтому следует помнить, что могут возникнуть различия, несмотря на то, что примеры программ и определения по существу одинаковы.

Ваша личная библиотека справочников может состоять из книги Линча {Lynch} Компьютеры, их влияние и использование [12], которая охватывает очень общие аспекты вычислений с вводной информацией о языке, предоставленной книгой Ксенакиса {Xenakis) [14]. Формулировка структурного программирования и программ представлена одной из книг Конвей {Conway}, например, Учебник для начинающих по структурному программированию [6]. Дополнительные сведения о прикладном программировании приведены в книге Хьюза {Hughes} [9]. Подробная информация о более сложных структурах данных приведена в книге Augenstein [1].

Читателям предлагается критиковать отдельные книги и любые дополнительные справочные руководства, которые они считают полезными. Digital Research ценит ваши комментарии и предложения так, что мы можем обновить этот список.

1. **Augenstein, M., and A. Tenenbaum. Data Structures and PL/I Programming. Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1979 (643p, Hardback, Typeset).**

Расширенное представление полного PL/I. Это - учебник для вуза, представляющий язык PL/I через ряд последовательных примеров, охватывающих рекурсию, обработку списков, деревья и графы, сортировку, поиск, хеширование и управление выделением памяти. Включена обширная библиография. Акцент делается на реализации структур данных, используя подмножество полного PL/I, который почти соответствует подмножеству G. Структурное программирование не выделено.

1. **Bates, F., and M. Douglas. Programming Language/One. Prentice Hall, Inc., Englewood Cliffs, New Jersey, 1970 (419p, Paperback, Hand Typed).**

Простое введение в PL/I. В этой книге представлены основные элементы полного PL/I, с некоторым акцентом на коммерческую обработку, включая структуры, записи, форматирование и обработку ошибок. Объяснения подкреплены примерами. Структурное программирование не акцентируется.

1. **Cassel, D. PL/I: A Structured Approach. Reston Publishing, Inc. , Reston, Virginia, 1978 (219p, Paperback, Typeset).**

Введение в PL/I среднего уровня. Представлена часть полного PL/I, описывая пакетную обработку и коммерческое применение. Элементы языка представлены четко, но не уделено особе внимание на разработку программ или правильное структурирование, как подразумевает название.

1. **Clark, F. J. Introduction to PL/I Programming. Allyn and Bacon, Inc., Boston 1971 (243p, Paperback, Typeset).**

Начальное введение в самостоятельное изучение PL/I посредством упражнений. Этот текст описывает часть полного PL/I от традиционного ориентированного на карточки подхода, начинающегося с обсуждения двоичных чисел и продолжающегося через типы основных операторов к простому вводу-выводу на основе STREAM и RECORD. Структурное программирование не выделено, несмотря на то, что представлены примеры коммерческой обработки.

1. **Conway, R. A Primer on Disciplined Programming. Winthrop Publishers, Cambridge, Mass., 1978 (419p, Paperback, Computer Typed).**

Учебник, предназначенный для PL/C, диалекта PL/I Корнельского университета. Один из трех университетских учебников Конвея {Conway} и др. авторов, охватывающий введение в программирование, с акцентом на методы, используемые для формулирования, разработки и тестирования программ. Включает краткие обсуждения поиска и упорядочивания списков, коммерческих отчетов, строковых операций и интерактивных систем. Делается акцент на методы структурного программирования и механизмы программирования, а не обширные примеры рабочих программ.

1. **Conway, R., and D. Gries. Primer on Structured Programming. Winthrop Publishers, Cambridge, Mass., 1976 (397p, Paperback, Computer Typed).**

Книга о структурного программирования на PL/C. По существу то же содержание, что и в предыдущей книге Конвея, возможно, с большим акцентом на работу в системе программирования PL/C Корнельского университета.

1. **Conway, R., D. Gries, and D. Wortman. Introduction to Structured Programming. Winthrop Publishers, Cambridge, Mass., 1977 (420p, Paperback, Computer Typed).**

Книга о структурном программировании с использованием PL/C Корнельского университета и системы SP/k университета Торонто. Снова, подобна первой книге Конвея с добавлением разделов по обработке файлов и трансляции языка, используя компиляторы и интерпретаторы.

1. **Groner, G. PL/I Programming in Technological Applications. John Wiley & Sons, New York, 1971 (230p, Paperback, Typeset).**

Введение в программирование инженерных приложений в PL/I. Эта книга описывает полный PL/I с примерами, полученными из пакетной обработки при реализации на IBM. Представлена разработка программ с помощью блок-схем, с многочисленными полными примерами научных приложений. Представлены несколько примеров вывода и генерации диаграмм. Делается акцент на объяснении вычислений FLOAT BINARY посредством полных примеров. Программы не очень хорошо структурированы.

1. **Hughes, J. K. PL/I Structured Programming. Second edition, John Wiley & Sons, New York, 1979 (825p, Hardback, Typeset).**

Подробное руководство по общему программированию в PL/I. Это - одно из наиболее полных описаний полного языка PL/I. Темы включают структурное программирование, обработку простых элементов данных, запись и обработку файлов и работа со списками. Акцент делается на коммерческом программирование с использованием IBM PL/I.

1. **Hume, J. N. P., and R. C. Holt. Structured Programming Usinq PL/I and SP/k. Reston Publishing, Inc., Reston, Virginia 1975 (340p, Paperback, Computer Typed).**

Введение в структурное программирование PL/I. Этот учебник знакомит с PL/I через ряд подмножеств называемых от SP/1 до SP/8. Каждое следующее подмножество включает более полный язык PL/I. Текст начинается с основных понятий программирования и продвигается через различные конструкции языка PL/I. Примеры программ включают обработку строк и массивов, обработку списков и обработку файлов. Также описаны машинный язык, ассемблер и компиляция. Акцент делается на структурном программировании.

1. **Kennedy, M., and M. B. Solomon. Structured PL/Zero Plus PL/One. Prentice-Hall, Englewood Cliffs, New Jersey, 1977 (695p, Paperback, Computer Typed).**

Довольно всестороннее введение в PL/I. Эта книга охватывает основные элементы PL/I в некоторых деталях, используя PL/C для примеров. Кратко рассматривается язык PL/I IBM уровня F. Большинство объектов языка хорошо проиллюстрированы на простых примерах.

1. **Lynch, R. E. , and J. R. Rice. Computers, Their Impact and Use. Holt, Rhinehart and Winston, New York, 1978 (440p, Paperback, Typeset).**

Основная вводная книга по компьютерам и PL/I. Учебное пособие для вузов, предназначенное для обучения компьютерам нетехнических специальностей. Половина книги дает обзор компьютеров, их истории, влиянию на общество и как они используются. Описываются операционные системы, языки и типы языков. В остальной части обсуждается использование множество приложений IBM PL/I, включая простую обработку файлов. Структурное программирование не выделяется.

1. **Ruston, H. Programming with PL/I. McGraw-Hill, New York, 1978 (541p, Paperback, Typeset).**

Всесторонний учебник знакомящий с PL/I. Эта книга описывает PL/I с точки зрения пакетной обработки, используя полный язык PL/I для примеров. Отражено составление программы посредством составления блок-схем. Описаны элементы PL/I, включая простые операторы, управляющие структуры, массивы, строки, процедуры и обработка файлов. Примеры имеют научную ориентацию. Описаны основы обработки ошибок. Структурное программирование не выделено.

1. **Xenakis, J. J. Structured PL/I Programming. Duxbury Press, North Scituate, Mass., 1979 (413p, Paperback, Typeset).**

Всестороннее введение в PL/I, близкое к подмножеству G. Описаны основы программирования с краткой историей языков программирования. Показаны элементы полного PL/I, включая преобразование типов данных, массивы, строки и процедуры. Включен небольшой раздел по программированию, затем раздел игр, который содержит программу крестики-нолики. Книга простая по содержанию и легко читается.

**Конец приложения 7**

Приложение 8 Перечень терминов

**агрегат**: Набор связанных элементов данных, на которые можно сослаться вместе или индивидуально.

**алгоритм**: Любая процедура, состоящая из конечного числа однозначных, повторяемых шагов, которые характеризуют решение проблемы.

**аргумент**: Значение, передаваемое в подпрограммы или функции.

**атрибут**: Любая характеристика элемента данных, такая как с фиксированной или с плавающей точкой, десятичный или двоичный, степень, и т.д.

**базированная переменная**: Переменная, которая описывает хранение, доступ к которому осуществляется с использованием указателя.

**библиотека подпрограмм времени выполнения**: Библиотека процедур, которые поддерживают выполнение программы PL/I.

**библиотека**: Файл, содержащий объектные модули и каталог внешних имен в объектных модулях.

**битовая строка**: Ноль или больше двоичных цифр (0 или 1).

**блок процедуры**: Последовательность операторов, ограниченных операторами PROCEDURE и END. Поток управления переходит в процедуру, в точке определения ее имени в операторе CALL или функциональной ссылки, в которой PL/I выполняет активацию ее блока и внутренних переменных, объявленных в нем.

**блок BEGIN**: Один или несколько операторов, ограниченных оператором BEGIN и соответствующим ему оператором END. Блок активируется, когда управление достигает оператора BEGIN. Когда управление передается в блок BEGIN, PL/I выполняет активацию блока и для переменных, объявленных в нем.

**блок**: Любая последовательность операторов PL/I, ограниченных одной из пар операторов PROCEDURE и END или BEGIN и END.

**верхняя граница**: Верхний предел размерности массива.

**внешняя переменная**: Переменная, которая известна в любом блоке, в котором она объявлена с атрибутом EXTERNAL.

**внешняя процедура**: Процедура, которая не содержится в какой-либо другой процедуре.

**внутренняя переменная**: переменная, на значение которой можно ссылаться из блока, который объявляет ее и любых блоков, расположенных внутри блока, объявляющего ее.

**внутренняя процедура**: Процедура, расположенная в некоторой другой процедуре.

**возвращаемое значение**: Значение, возвращенное функцией, которое заменяет функцию в ее ссылке.

**встроенная функция**: Любая функция, обеспечиваемая как часть языка PL/I.

**выделение**: процесс получения А) памяти для переменной, или Б) определенной области памяти, выделяемой для базированной переменной.

**выражение**: Любая допустимая комбинация операндов и операторов, которая уменьшается до единственного значения.

**вычислительный**: Тип данных, с которыми можно выполнять операции. Вычислительными типами данных являются арифметические и строковые.

**главная процедура**: Процедура, которая получает управление, когда программа начинает выполняться. Главная процедура всегда - внешняя процедура.

**граничная пара**: Выражение, которое устанавливает число элементов в каждой размерности массива.

**записеориентированный ввод-вывод**: Передача данных, сгруппированных в определяемых пользователем блоках, называемых записями.

**запись**: Организованный набор данных, который PL/I передает, используя операторы записеориентированного ввода-вывода.

**идентификатор**: Имя, состоящее из 1-31 символов, определенное для переменной, метки оператора, точки входа или файловой константы.

**имя состояния**: Ключевое слово PL/I, связанное с конкретным состоянием.

**индекс**: Целочисленное выражение, определяющее отдельный элемент массива.

**категория ввода-вывода**: Общий метод, используемый при чтение или записи элементов данных в файл. Категории ввода-вывода - потоковый (STREAM) ввод-вывод и записеориентированный (RECORD) ввод-вывод.

**класс памяти**: Атрибут переменной, который описывает, как для нее PL/I выделяет и освобождает память. Классы памяти бывают AUTOMATIC, BASED, PARAMETER и STATIC.

**ключ**: (А) любое значение, которое используется для указания определенной записи в файле, или (Б) элемент данных, который является частью записи в индексно-последовательном файле, или (В) относительный номер записи в записи в записеориентированном файле.

**ключевое слово**: Любой идентификатор PL/I, имеющий определенное значение при использовании его в соответствующем контексте.

**комментарий**: Любая последовательность символов, расположенная между составными парами /\* и \*/. Комментарии обеспечивают документальный текст и игнорируются компилятором.

**компилятор**: Программа, которая переводит исходные операторы высокоуровневого языка программирования в объектный модуль. Объектный модуль состоит из инструкций процессора и определенной информации о перемещении, используемой редактором связей для формирования командного файла.

**константа**: А) любое литеральное значение, определенное для представления вычислительных элементов данных, или Б) любая точка входа или имя метки, объявленные неявно в контексте, или В) любой идентификатор, объявленный с одним из атрибутов ENTRY или FILE, но без атрибута VARIABLE.

**коэффициент повторения**: Целочисленная константа, заключенная в круглые скобки, указывающая число повторения значений при инициализации элементов массива или число повторений использования элемента формата в операторе редактируемого ввода-вывода.

**листинг**: Выходной файл, создаваемый компилятором, который перечисляет операторы в исходной программе с соответствующими номерами строки и дополнительной информацией.

**логический оператор**: Оператор, который выполняет логическую операцию со значениями строки битов.

**массив**: Именованный набор элементов данных с одинаковыми атрибутами, с доступом к отдельным элементам при помощи индексов.

**масштабный коэффициент**: Число дробных цифр, определяемое для элемента данных FIXED DECIMAL.

**метка**: Любой идентификатор PL/I, завершенный двоеточием, который можно использовать для идентификации оператора.

**младшая структура**: Структура, которая является элементом структуры (подструктура).

**набор символов ASCII**: Набор числовых значений, которые представляют символы и управляющую информацию, установленную американским стандартным кодом обмена информацией.

**нелокальный GOTO**: Оператор перехода, который передает программное управление оператору в охватывающем блоке.

**нижняя граница**: Нижний предел размерности массива.

**номер уровня**: Целочисленная константа, которая определяет иерархическое отношение имени в структуре относительно других имен в структуре.

**область действия**: Набор блоков в программе, в которых известно объявление идентификатора.

**объектный модуль**: Вывод из компилятора или ассемблера, который можно соединить с другими модулями для формирования командного файла.

**объявление**: Явная или неявная спецификация идентификатора и его типа данных.

**оператор конкатенации**: Оператор ||, который соединяет два строковых значения для формирования одной строки.

**оператор присваивания**: Исполняемый оператор, который присваивает значение переменной.

**оператор сравнения**: смотрите реляционный оператор.

**оператор**: Допустимая последовательность ключевых слов PL/I, идентификаторов и специальных символов, которая определяет исполняемую инструкцию или объявление данных.

**операция**: Символ, который указывает PL/I, выполнить определенную функцию.

**описатель указателя**: ссылка указателя и символ пунктуации, который связывает определенное место хранения с базированной переменной.

**память**: Любое адресуемое расположение, которое хранит код или данные.

**параметр**: Переменная, которую PL/I сопоставляет с аргументом, когда программа вызывает процедуру.

**пароль**: определенное пользователями расширение имени файла, обеспечивающее безопасность.

**переменная с автоматическим выделением памяти**: Переменная, для которой компилятор выделяет память, при активации блока в котором она объявлена. Память освобождается при деактивации блока.

**переменная**: Элемент данных, значение которого может измениться во время выполнения программы.

**процедура**: Подпрограмма, получающая управление при вызове ее с помощью оператора CALL.

**последовательный доступ**: Метод доступа, который позволяет получить доступ к записям в записеориентированном файле последовательно.

**последовательный файл**: записеориентированный файл, в котором записи расположены последовательно. Вы можете только добавлять новые записи в конец файла и считать записи одну за другой.

**построчный порядок**: Порядок, в котором PL/I хранит или присваивает значения элементам массива. При построчном расположении самый правый индекс изменяется первым.

**поток управления**: Последовательность, в которой процессор выполняет отдельные инструкции в программе.

**потоковый ввод-вывод**: Передача и интерпретация данных с точки зрения последовательностей символов ASCII, разграниченных пробелами, символами табуляции, запятыми или полями, определяемыми элементами формата.

**преобразование**: Процесс преобразования значения их одного типа данных в другой.

**префиксный оператор**: Оператор, который предшествует переменной или константе для указания или изменения их знака.

**приоритет**: Приоритет оператора, используемый PL/I при выполнении операций в выражении. PL/I выполняет операцию с более высоким приоритетом перед операцией с более низким приоритетом.

**произвольный доступ**: операция ввода-вывода в записеориентированный файл, где для доступа к отдельным записям файла, используются значения FIXED BINARY, называемые ключами.

**протяженность**: Диапазон между нижней границей и верхней границей одной размерности массива.

**процедура**: Последовательность операторов, разграниченных оператором PROCEDURE и оператором END. Процедура может быть подпрограммой, которая вызывается с помощью оператора CALL или функцией, которая вызывается функциональной ссылкой.

**псевдопеременная**: Имя встроенной функции, которую можно использовать в левой части оператора присваивания, чтобы задать особое значение присвоению.

**размерность**: Набор границ, которые определяют одно измерение массива.

**редактор связей**: Программа, которая объединяет перемещаемые объектные модули в командный файл и разрешает ссылки между внешними переменными, объявленными в модулях.

**рекурсивная процедура**: Процедура, которая может вызвать себя.

**реляционный оператор**: Оператор, определяющий отношение между двумя выражениями и приводящий к бинарному значению, указывающему, является ли отношение истиной или ложью.

**с плавающей точкой**: Тип данных, который представляет очень маленькие или очень большие числа. У числа с плавающей точкой есть мантисса и дополнительно экспонента целого числа со знаком.

**связанное хранение**: непрерывное размещение данных.

**сигнал**: Механизм, которым PL/I указывает, что произошло событие установившее состояние.

**символьная строка**: Ноль или более символов ASCII.

**скаляр**: Элемент данных, который не является агрегатом.

**состояние**: Любое событие, которое прерывает нормальное выполнение программы и инициирует ответ, определяемый пользователем или системой по умолчанию.

**спецификатор указателя**: ссылка указателя и символ пунктуации, который связывает определенное место хранения с базированной переменной.

**список аргументов**: Ноль или больше параметров, определенных при вызове процедуры или встроенной функции.

**список параметров**: Список имен переменных, значения которых определены при вызове процедуры. Оператор PROCEDURE для точки входа процедуры определяет список параметров.

**список формата**: Список элементов формата, соответствующих элементам данных для редактируемого ввода-вывода.

**ссылка на переменную**: Любая ссылка на переменную, включая с указанием индексов и имен членов.

**ссылка на структуру**: Переменная ссылка на всю структуру (в отличие от члена структуры).

**ссылка**: Появление идентификатора в любом контексте, кроме его объявления.

**старшая структура**: Имя всей структуры, с помощью которого можно указать все элементы структуры в одной ссылке. Главная структура всегда имеет номер уровня 1.

**статическая переменная**: Переменная, для которой компилятор выделяет память на все время выполнения программы.

**строковые данные**: данные в виде строки битов или символьной строки.

**структура**: Иерархическая организация логически связанных элементов данных, названных членами, которые не обязаны иметь одинаковый тип данных.

**тип данных**: Класс, к которому принадлежит элемент данных, и который определяет операции, которые можно с ним выполнить.

**тип файла**: часть спецификации файла от нуля до трех символов, которая обычно описывает использование файла.

**точка входа**: Оператор или инструкция, с которого начинается выполнение процедуры.

**точность**: Количество цифр, связанных с арифметическим элементом данных.

**указатель**: Элемент данных, значение которого - адрес места хранения.

**управляемый списком ввод-вывод**: Любая передача данных между программой и внешним устройством, для которого PL/I обеспечивает автоматическое преобразование данных и форматирование.

**управляющая переменная**: Переменная, значение которой изменяется на каждой итерации DO-группы и она может быть протестирована, чтобы определить, продолжать ли выполнение операторов DO-группы.

**управляющие данные**: Элемент данных, который не является строковым или арифметическим. Управляющие типы данных - ENTRY, FILE и LABEL.

**уточненная ссылка указателя**: Спецификация базированной переменной с точки зрения значения указателя, которое указывает расположение переменной.

**уточненная ссылка**: Однозначная ссылка на член структуры, которая определяет каждое имя более высокого уровня в структуре и разделяющая имена точками.

**файл %INCLUDE**: Внешний файл, из которого компилятор читает исходный текст при компиляции программы PL/I.

**файл PRINT**: потоковый выходной файл, для которого PL/I выравнивает определенные данные по предопределенным позициям табуляции и управляет выводом с указанным размером страницы и размером строки. В файл PRINT PL/I не включает строки в апострофы.

**файл RECORD**: Файл, содержащий двоичные данные, которые PL/I передает без преобразования.

**файл**: А) в PL/I, входной источник или выходная цель, определяемая в операторе ввода-вывода, или Б) набор данных на запоминающем устройстве.

**файловая константа**: Любой идентификатор, который объявляется с атрибутом FILE, но не с атрибутом VARIABLE.

**функциональная ссылка**: Любая ссылка на имя встроенной или написанной пользователем функции в операторе PL/I.

**функция**: процедура, которая выполняется, когда ее имя используется в выражении, и которая возвращает значение в точку вызова.

**хранение**: Любая область памяти, которая связана с определенной переменной.

**целочисленная константа**: Любая необязательно со знаком строка десятичных цифр.

**целочисленные данные**: Данные, представленные как FIXED BINARY или FIXED DECIMAL с нулевым масштабным коэффициентом.

**член**: Элемент данных в структуре. Член может быть скалярным элементом данных, массивом или структурой.

**шаблон (picture)**: символьное представление арифметического значения, состоящее из символьной строковой константы, определяющей позицию десятичной точки, подавление нулей и знаков.

**элемент формата**: Значение, указывающее представление данных и форматирующую информацию, используемую при редактируемом вводе-выводе.

**элемент**: Любой отдельный элемент данных в массиве, на который можно сослаться с помощью индексов.

**DO-группа**: Любая последовательность исполняемых операторов, ограниченных оператором DO и соответствующим оператором END.

**FIXED BINARY**: Тип данных, который представляет целочисленные значения.

**FIXED DECIMAL**: Тип данных, который представляет десятичные значения с десятичной точкой и постоянным числом дробных цифр.

**ON-блок**: операторы PL/I, определяющие выполняемое действие, когда программа устанавливает конкретное ON-состояние.

**ON-состояние**: Любое из нескольких именованных состояний, которые могут прервать программу и генерировать сигнал.

**Примечание**: материал в этом приложении был частично адаптирован из публикаций Digital Equipment Corporation®. Материал, опубликованный здесь, является исключительной ответственностью Digital Research Inc.

**Конец приложения 8**

Предметный указатель

%

%INCLUDE, оператор, 27

вложенный, 27

%REPLACE, оператор, 28

А

Агрегат, 29, 48

массив, 48

смешанный, 55

структура, 48

Адрес переменной, 36

Апостроф

как часть строки, 32, 94

Аргументы, 19

передача значением, 137

Арифметические данные, 29

Арифметические преобразования, 40

Атрибут, 29

класса памяти, 63

неявное присвоение, 29

по умолчанию, 82

явное присвоение, 29

ALIGNED, 132

BINARY, 132

BIT(n), 32, 132

BUILTIN, 132

CHARACTER(n), 133

DECIMAL, 133

DEFINED, не реализован, 136

DIRECT, 83

ENTRY, 35

ENTRY, 133

ENVIRONMENT, 84, 133

параметр Buff, 85

параметр Fixed, 85

параметр Password, 85

EXTERNAL, 16, 35, 133

FILE, 133

FIXED, 134

FLOAT, 134

FLOAT DECIMAL, не реализован, 136

INITIAL, 50, 63, 65, 134

INPUT, 83

KEYED, 83

LABEL, 33, 134

LINESIZE, 83

OUTPUT, 83

PAGESIZE, 83

PICTURE, не реализован, 136

POINTER, 134

PRINT, 83

RECORD, 83

RECURSIVE, 23

RETURNS, 23, 35, 134

в главной процедуре, 19

SEQUENTIAL, 83

STREAM, 83

TITLE, 83

UPDATE, 83

VARIABLE, 33, 35, 82, 135

VARYING, строки, 32, 58, 135

Атрибуты файла, 85

Б

Базированная переменная, 63

спецификатор указателя, 64

Битовая строка, 32

Блок, 13

внешний, 15

внутренний, 15

параметров, 148

BEGIN, 13

PROCEDURE, 13

Блок параметров файла, 88, 146

В

Ввод-вывод

записеориентированный, 91

потоковый, 91

строчно-ориентированный, 94

управляемый редактированием, 95

управляемый списком, 93

Внутренние форматы данных, 140

файловых констант, 146

BIT, 145

CHARACTER, 145

ENTRY и LABEL, 146

FIXED BINARY, 140

FIXED DECIMAL, 144

FLOAT BINARY

двойной точности, 143

одинарной точности, 141

POINTER, 146

Встроенная функция, 108

ABS, арифметическая, 110

ACOS, математическая, 110

ADDR, вспомогательная, 68, 110

ASCII, преобразования, 110

ASIN, математическая, 111

ATAN, математическая, 111

ATAND, математическая, 111

BINARY, преобразования, 42

BINARY, преобразования, 46, 111

BIT, преобразования, 44, 47, 112

BOOL, строковая, 60, 112

CEIL, арифметическая, 112

CHARACTER, преобразования, 45, 46, 113

COLLATE, строковая, 113

COPY, строковая, 113

COS, математическая, 113

COSD, математическая, 114

COSH, математическая, 114

DATE, вспомогательная, 114

DECIMAL, преобразования, 43, 114

DIMENSION, вспомогательная, 115

DIVIDE, арифметическая, 43, 115

EXP, математическая, 60, 115

FIXED, преобразования, 43, 115

FLOAT, преобразования, 44, 116

FLOOR, арифметическая, 116

HBOUND, строковая, 116

INDEX, преобразования, 116

LBOUND, вспомогательная, 117

LENGTH, строковая, 117

LINENO, вспомогательная, 90, 117

LOCK, вспомогательная, 84, 117

LOG, математическая, 60, 118

LOG10, математическая, 118

LOG2, математическая, 118

MAX, арифметическая, 118

MIN, арифметическая, 119

MOD, арифметическая, 119

NULL, вспомогательная, 67, 119

ONCODE, состояния, 80, 119

ONFILE, состояния, 90, 119

ONKEY, состояния, 90, 120

PAGENO, вспомогательная, 90, 120

RANK, преобразования, 120

REVERSE, строковая, 120

ROUND, арифметическая, 120

SEARCH, строковая, 121

SIGN, арифметическая, 121

SIN, математическая, 121

SIND, математическая, 121

SINH, математическая, 121

SQRT, математическая, 122

SUBSTR, строковая, 61, 122

TAN, математическая, 122

TAND, математическая, 122

TANH, математическая, 123

TIME, вспомогательная, 123

TRANSLATE, строковая, 123

TRIM, строковая, 123

TRUNC, арифметическая, 124

UNLOCK, вспомогательная, 84, 124

UNSPEC, вспомогательная, 62, 124

VERIFY, строковая, 125

Выражение, 57

инфиксное, 57

операнд, 57

префиксное, 57

Г

Главная программа, 21

Д

Длина записи, 84

Длина строки, 32

Доступ к системным процедурам, 24

И

Идентификатор, 24

Иерархическая структура программы, 12

Имя

внешнего устройства, 83

файла, 84

Индексная ссылка

на элемент массива, 49

Источник, 39

Исходная программа, 23

К

Категории состояний, 77

Класс памяти, 63

AUTOMATIC, 63, 132

BASED, 63, 132

PARAMETER, 65, 134

STATIC, 65, 135

Ключевое слово, 11, 24

Комментарии, 27

Конец строки, 83

Константа, 25, 29

битовой строки, 33

формат, 33

десятичная, 31

метки, 33, 76

неявное объявление, 34

преобразование, 31

символьной строки, 32

файловая, 36, 82, 88

SYSIN, 90, 93, 94, 104

SYSPRINT, 90, 93, 95, 105

entry, 35

FIXED DECIMAL, 31

FLOAT BINARY, 30

Коэффициент повторения

в атрибуте INITIAL, 50, 65

элемента формата, 95

Л

Логические блоки программы, 12

Логические данные, 32

М

Маркер

страницы, 83

строки, 83

Массив, 48

индекс, 48

инициализация, 65

меток-констант, 34

общее количество элементов, 48

отдельные элементы, 52

структур, 55

Масштабный коэффициент, 29

по умолчанию, 29

Метка

в операторе END, 13

константа, 33

переменная, 33

Методы доступа к файлу, 91

Н

Набор атрибутов файла, 85

Набор данных, 82

Набор символов PL/I, 23

Непрерывное хранение, 52, 56

О

Область действия

меток констант и переменных, 34

переменной, 16

Объектный код в формате

Intel, 138

Microsoft, 138

Объявление файла, 82

Ограничители, 25

Оператор

%INCLUDE, 27, 129

вложенный, 27

%REPLACE, 28, 130

присваивания, 57, 126

ALLOCATE, 66, 69, 126

BEGIN, 126

CALL, 18, 126

CLOSE, 87, 126

DECLARE, 37, 126

атрибуты по умолчанию, 38

DO, 71, 127

управляемый, 71

DO BY WHILE, 74

DO REPEAT, 73

DO REPEAT WHILE, 73

DO WHILE, 72

END, 13, 21, 71, 128

FORMAT, 99, 128

FREE, 66, 67, 128

GET EDIT, 104, 128

GET LIST, 93, 128

GOTO, 34, 75, 76, 128

нелокальный, 76

IF, 75, 128

вложенный, 75

NULL, пустой, 129

ON, 77, 129

OPEN, 82, 129

PROCEDURE, 21, 129

опция

EXTERNAL, 21

MAIN, 21

STACK(b), 21

PUT EDIT, 105, 129

PUT LIST, 93, 130

READ, 106, 130

READ с опцией KEY, 106, 130

READ с опцией KEYTO, 106, 130

READ VARYING, 94, 130

RETURN, 18, 130

REVERT, 78, 131

SIGNAL, 78, 131, 166

STOP, 75, 131

WRITE, 107, 131

WRITE с опцией KEYFROM, 107, 131

WRITE VARYING, 95, 131

Оператор PL/I

выполняемый, 12

объявления, 12

пустой, 12

сравнения, 59

структурный, 12

управления, 71

Операторы препроцессора, 27

Операция

||, 58

арифметическая, 26

битовых строк, 26

инфиксная, 57

конкатенации, 26

приоритет, 58

составная, 25

сравнения, 26

Очередность выполнения операций, 58

П

Параметры

фактические, 19

формальные, 19

Передача параметра

значением, 19

ссылкой, 19

Переменная, 29

базированная, 63

в скобках, 20

внешняя, 16

локальная, 16

массива, 48

метки, 75

структуры, 48

файловая, 36, 82

entry, 35

Подпрограмма, 18

Подструктура, 53

Правила форматирования, 23

Преобразование данных, 39

Программа PL/I, 15

Процедура

внешняя, 15

внутренняя, 15

вызов, 18

главная, 15

определение, 21

подпрограмма, 18

точка выхода, 21

функция, 18

Псевдопеременная

SUBSTR, 60, 61

битовая, 61

UNSPEC, 60, 62

Пустое пространство, 25

Р

Разделители, 25, 93

Размер внутреннего буфера, 84

Размерность массива, 48

Режим открытия файла, 84

Locked, 84

Readonly, 84

Shared, 84

Результат, 39

Рекурсивная процедура, 132, 161, 176

нестандартная реализация, 137

Рекурсия, 63

С

Свободный формат программы, 23

Связный список, 67

Символ конца-файла, 88

Символы ASCII, 59, 168

Символьная строка, 32

переменной длины, 32, 135

пустая, 32

Скаляр, 29

Совместное использование памяти, 68

Состояние, 77

восстанавливаемое, 77

невосстанавливаемое, 77

подкоды, 79

ENDFILE, 88

ENDPAGE, 89, 94

ERROR, 79

ERROR(1), 47, 79, 102

ERROR(14), 88

ERROR(3), 123

ERROR(7), 66

FIXEDOVERFLOW, 31

KEY, 89

UNDEFINEDFILE, 89

Специальные символы, 11, 26

Спецификация шаблона, 99

выходные символы, 100

дрейфующие символы, 100

символы

вставки, 101

кредита и дебета, 102

позиции десятичной точки, V, 101

условные цифровые, \* и Z, 101

цифровые, 9, 101

статические символы, 100

точность и масштабный коэффициент, 102

Список ключевых слов, 24

Список форматов, 95

Среда логического блока, 12, 71

Ссылка

верхнего уровня, 16

на отдельный элемент массива, 49

на член структуры, 54

определенная с помощью указателя, 64

уточненная на структуру, 48

функции, 18

Стек состояний, 138

Строковые данные, 32

Структура, 48, 53

главная, 53

младшая, 53

определение, 53

полностью определенные имена, 55

составные имена, 54

старная, 53

уточненная ссылка, 48

Т

Тело блока, 13

Тип файла

PLI, 23

PRN, 158

Точка входа в процедуру, 21

Точность, 29

двойная, 30

одинарная, 30

по умолчанию

FIXED BINARY, 30

FLOAT BINARY, 30

У

Указатель, 63

на адрес блока управления файлом, 88

нулевое значение, 68

Управляемый оператор DO, 71

Управляющие символы

в строковых константах, 32, 95

Уровень защиты паролем, 84

параметр Password, 85

Delete, 85

Read, 85

Write, 85

Ф

Файл, 82

заблокированный, 84

записеориентированный, 106

DIRECT, прямого доступа, 106

SEQUENTIAL, последовательного доступа, 106

прямой доступ, 88

совместно используемый, 84

только для чтения, 84

STREAM, 91

Фактический параметр

Преобразование, 20

Функция, 18

вызов, 18

значение, 18

обработки состояний, 109

Ц

Цель, 39

Ч

Член структуры, 48, 53

инициализация, 65

неоднозначность, 54

Э

Экспоненциальное представление чисел, 30

Элемент удаленного формата, 99

Элементы данных, 29

управления программой, 33

ENTRY, 35

FILE, 36

LABEL, 34

POINTER, 36

Элементы формата данных, 96

формат A, 96

формат B, 96

формат E, 97

формат F, 97

формат P, шаблон, 99

Элементы формата управления, 97

**COLUMN**, 97

**LINE**, 98

**PAGE**, 98

**SKIP**, 98

**X**, 98

D

DECLARE, оператор, 37

dimension, 48

DO-группа, 71

E

extent, 48

F

FIXED BINARY, тип данных, 29

деление, 43

FIXED DECIMAL, тип данных, 31

FLOAT BINARY, тип данных, 30

L

LABEL, тип данных, 33

O

ON-блок, 77

ON-состояние, 178

Open List, 88

1. Оригинал руководства на английском языке [↑](#footnote-ref-1)
2. FPB Сокращение от английского File Parameter Block [↑](#footnote-ref-2)
3. FCB Сокращение от английского от английского File Control Block [↑](#footnote-ref-3)