Г. А. ЛЕПИН-ДМИТРЮКОВ

ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ ПЛ/I

(для ДОС ЕС ЭВМ)

МОСКВА «СОВЕТСКОЕ РАДИО» 1978

6Ф7.3
Л48
УДК 681.3.06

Лепин-Дмитрюков Г. А.

Л48 Программирование на языке ПЛ/1 (для ДОС ЕС ЭВМ). — М.: Советское радио, 1978. — 288 с., ил.

В книге описан наиболее универсальный алгоритмический язык ПЛ/1, показана связь программы, написанной на языке, с дисковой операционной системой, а также даны необходимые для практической работы сведения о трансляторе.

Книга рассчитана на специалистов — пользователей машин ЕС ЭВМ, системных программистов, разработчиков АСУ.

Л $\frac{\text{30502-040}}{\text{046(01)-78}}$ **61-77**

6Ф7.3
32.973

Редакция кибернетической литературы

ИБ № 221

Генрих Артурович Лепин-Дмитрюков

ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ ПЛ/1
(ДЛЯ ДОС ЕС ЭВМ)

Научный редактор **В. М. Козуб**
Редактор **Т. М. Любимова**
Художественный редактор **Н. С. Шеин**
Обложка художника **В. В. Волкова**
Технический редактор **Т. П. Сафонова**
Корректор **Л. А. Максимова**
Сдано в набор 29.07.77. Подписано в печать 15.03.78
Формат 60×90/16 Бумага типографская № 2
Литературная гарн. Высокая печать
Объем 18 усл. п. л., 18,57 уч.-изд. л.
Тираж 54 000 экз. Зак. 1939 Цена 1 р. 10 к.
Издательство «Советское радио», Москва, Главпочтамт, а/я 693
Московская типография № 4 Союзполиграфпрома
при Государственном комитете Совета Министров СССР
по делам издательств, полиграфии и книжной торговли
Москва, И-41, Б. Переяславская, 46.

Введение

Наименование алгоритмического языка ПЛ/1 образовано из начальных букв английских слов Programming Language/One. Язык был разработан в период 1963—1966 гг. сотрудниками американской фирмы IBM и ассоциацией пользователей машин IBM. При создании языка была учтена необходимость обеспечения как можно большей простоты и удобства написания программы. За время своего существования язык претерпел большие изменения. Первые версии сильно отличались друг от друга. Однако постепенно язык стабилизировался, и новые публикации отличаются от предыдущих, как правило, лишь редакционными поправками, устранением неточностей или усовершенствованием отдельных элементов.

Язык ПЛ/1 появился после целого ряда весьма совершенных языков, и, конечно, эти языки-предшественники оказали существенное влияние на его структуру. Так, в нем сохранены от АЛГОЛа блочная структура программы (правда, слегка видоизмененная), возможность динамического распределения памяти, имеется аппарат вызова процедур (в том числе и рекурсивных), способ задания форматов аналогичен используемому в ФОРТРАНе. В то же время язык ПЛ/1 имеет много новых свойств. Это позволяет существенно расширить области его применения, например, считается, что язык удобен для моделирования, решения логических задач, исследования логических схем, решения задач в реальном масштабе времени и даже для разработки системы математического обеспечения. Особенности языка позволяют повысить эффективность выполнения рабочих программ и более рационально использовать имеющееся на машине оборудование.

Можно отметить и такие свойства этого языка, как использование разного типа данных (двоичных, десятичных, символьных, комплексных чисел, матриц и т. д.), возможность весьма сложной организации данных (массивы, таблицы, тексты, картотеки), возможность программной реакции на прерывание, несколько способов распределения памяти, гибкое использование вспомогательных запоминающих устройств, большой набор стандартных функций и процедур. Обобщая сказанное, следует указать, что ПЛ/1 относится к проблемно-ориентированному языку высокого уровня.

В последние годы для решения экономических задач в качестве проблемно-ориентированного языка применялся КОБОЛ, а для решения научно-технических задач — АЛГОЛ-60 и ФОРТРАН. В настоящее время благодаря слиянию экономических и научно-технических задач и проникновению математики в экономику (напри-

мер, методы исследования операций), целесообразно использовать универсальный язык высокого уровня ПЛ/1, который значительно облегчает описание задач любого класса, а также предоставляет возможность применения общих принципов обработки данных, как например: ввод — вывод больших объемов данных, подготовка различных отчетов, обработка произвольных строк символов и т. д.

Этот язык удовлетворяет запросам и требованиям большинства потребителей и способствует возможностям вычислительных машин третьего поколения. Он содержит основные элементы и свойства таких проблемно-ориентированных языков, как АЛГОЛ-60, ФОРТРАН и КОБОЛ. Наряду с этим он допускает некоторые операции, которые свойственны машинно-ориентированным языкам, а также позволяет работать с внутренним представлением данных.

Язык ПЛ/1 построен так, что программист легко может использовать его на уровне, соответствующем своей квалификации. Фактически сложность используемых средств языка соответствует сложности программ, для которых этот язык применяют.

Одной из основных целевых установок при разработке языка было достижение модулярности, т. е. возможности выделения различных уровней языка для различных применений и различных степеней сложности.

В возможности образовывать простые специализированные подмножества языка путем почти чисто механического выбрасывания ненужных для данных приложений свойств состоит одна из основных особенностей языка ПЛ/1. Программисту, пользующемуся языком некоторого уровня, нет надобности знать о неиспользуемых им средствах других уровней.

В языке ПЛ/1 каждому описателю переменной, каждой уточняющей конструкции и каждой спецификации придана «интерпретация (принцип) умолчания». Это означает, что всюду, где язык представляет несколько возможностей, а программист не указал никакой, транслятор применяет интерпретацию умолчания, т. е. подразумевается некоторая возможность, предусмотренная в языке на этот случай. В качестве таких подразумеваемых для каждой конструкции возможностей в языке выбраны те, которые вероятнее всего потребуются программисту.

Обычно в системах обработки данных не реализуется полный объем теоретически определенного языка программирования ПЛ/1. Реализуемый язык представляет собой подмножество полного языка.

Объем реализации языка зависит также от параметров электронно-вычислительной машины (например, от размеров основной памяти, быстродействия, метода обращения к операционной системе). В рамках дисковой операционной системы (ДОС) реализуется подмножество языка ПЛ/1, куда не включены концепции полного языка, которые не поддерживаются работой ДОС (например, асинхронная обработка, передача данных на расстояние и др.). Таким образом, рассматриваемое в книге подмножество языка ПЛ/1 соответствует его реализации в ДОС ЕС версии 2.1.

Глава 1

ОБЩИЕ СВЕДЕНИЯ О ЯЗЫКЕ

1.1. Основные символы языка

Для записи программы на языке ПЛ/1 можно пользоваться одним из двух алфавитов: либо из 60 символов, либо из 48.

60-символьный алфавит состоит из букв, цифр и специальных знаков. Всего имеется 29 букв, а именно от *A* до *Z* (26 букв) и три дополнительных символа, которые по определению считаются буквами: знак доллара $*, знак цены (коммерческое *at*) @, знак номера #; 10 десятичных цифр от 0 до 9 и 21 специальный символ:

— пробел;
= знак равенства в логических выражениях или символ присваивания;
+ плюс;
— минус;
* звездочка;
/ наклонная черта;
(открывающая скобка;
) закрывающая скобка;
, запятая;
. точка;
' кавычка (апостроф);
% процент;
; точка с запятой;
: двоеточие;
¬ символ отрицания;
& символ логического умножения (И);
| вертикальная черта—символ логического сложения (ИЛИ)**);
> знак больше;
< знак меньше;

---

*) На некоторых устройствах ЕС ЭВМ символу доллара соответствует денежный символ ⌾

**) В данной реализации языка при набивке программы на перфокарты вместо вертикальной черты используется символ восклицательного знака (!)

— символ разбивки (черта под строкой);

? вопросительный знак.

48-символьный алфавит является подмножеством 60-символьного алфавита с некоторыми ограничениями. В него не включаются следующие символы:

% процент; > знак больше;

: двоеточие; < знак меньше;

¬ символ отрицания; $ знак доллара;

| символ ИЛИ; # знак номера;

& символ И; ; точка с запятой;

однако некоторые комбинации символов 60-символьного алфавита заменяются следующими сочетаниями:

60 символьный алфавит	*48 символьный алфавит*
:	..
;	. ,
%	//
>=	GT
<>	NG
>=	GE
<>	NE
<=	LE
<	LT
¬	NOT
|	OR
&	AND
||	CAT
->	PT

Отметим следующие особенности их использования:

— перед двумя точками, заменяющими двоеточее, должен быть пробел, если предыдущий символ есть точка;

— перед (вслед за) двумя наклонными чертами, заменяющими знак процента, должен быть пробел, если предыдущий (последующий) символ есть звездочка;

— последовательность «точка, запятая» заменяет точку с запятой, если она не стоит внутри примечания или строки символов или непосредственно за ней не стоит цифра;

— все слова GT, NG, ..., являются в 48-символьном алфавите зарезервированными, т. е. программист не может использовать их в качестве идентификаторов;

— в каждом случае, когда буквенному обозначению знака операции предшествует буква или цифра, между ними необходимо поставить не менее одного пробела; аналогично когда за буквенным обозначением знака операции следует буква или цифра, между ними нужно поставить хотя бы один пробел.

Например, сравнение переменных AG и BO2Y на неравенство нужно записать так:

AG NE BO2Y

а не AGNEBO2Y и не AGNE BO2Y или AG NEBO2Y.

Ограничители. Некоторые символы языка используются в качестве ограничителей, которые делятся на три класса:

— знаки операций;

— скобки;

— разделители.

Знаки операций, используемые в языке, подразделяются на четыре типа:

— знаки арифметических операций;

— знаки операций сравнения;

— знаки операций над строками битов (знаки логических операций);

— знаки операций над строками.

Имеются следующие знаки арифметических операций:

\+ сложение или одноместная операция плюс;

— вычитание или одноместная операция минус;

* умножение;

/ деление;

** возведение в степень.

Имеются следующие знаки операции сравнения:

> больше;

¬> не больше;

>= больше или равно;

= равно;

¬= не равно;

<= меньше или равно;

¬< не меньше;

< меньше.

Для логических операций над строками битов используются знаки:

¬ отрицание;

& логическое умножение;

| логическое сложение.

В качестве операции над строками используется знак ||, означающий сцепление.

Скобки. В языке используются только круглые скобки. Они употребляются в выражениях, в них заключают списки, а также указывается информация, связанная с тем или иным служебным словом.

Разделители:

, запятая, с помощью которой отделяются друг от друга элементы списка, используется в элементах формата и некоторых режимах;

. точка разделяет элементы уточненных имен, применяется в качестве десятичной или двоичной точки в константах;

; точка с запятой используется для отделения операторов друг от друга;

= знак присвоения или знак равенства в логических выражениях;

: двоеточие следует за метками и приставками-ситуациями, а также используется для разделения имен точек входа в процедуры;

␣ пробел разделяет элементы предложений (может добавляться в любом количестве);

' кавычки (апостроф), в которые заключают строковые константы и спецификации шаблона.

1.2. Идентификаторы

Идентификатором называется последовательность буквенно-цифровых символов и символа разбивки (знак подчеркивания), начинающаяся с буквы.

Максимальное количество символов, которые входят в состав идентификатора, равно 31. Исключение составляют лишь идентификаторы имен точек входов, которые могут состоять максимум из 6 символов.

Следует напомнить, что символы $, @ и # в языке отнесены к буквам и с них может начинаться идентификатор.

Знак подчеркивания используется прежде всего для лучшей читабельности в том случае, если идентификатор состоит из нескольких отдельных частей (слов). Другие символы, в том числе пробел, не могут использоваться в идентификаторах.

Хотя, учитывая приведенные ограничения, идентификаторы могут выбираться произвольно, они должны быть по возможности выразительны и символически описывать величину, которую они представляют. Например:

VAR, BCD32, XR20A, #32_45, $L32

Идентификаторы в языке ПЛ/1 служат для изображения:
— имен скалярных переменных;
— имен массивов;
— имен структур;
— операторных меток;
— имен входов;
— имен файлов;
— служебных (ключевых) слов;
— имен ситуаций.

Ключевое слово — это идентификатор, который при использовании в особом контексте имеет для транслятора смысловое значение. В 48-символьном алфавите ключевые слова запрещены для использования их в качестве идентификаторов, в 60-символьном алфавите они не резервируются.

Ключевые слова подразделяются: на идентификаторы операторов; описатели, разделяющие ключевые слова; имена встроенных функций; ситуации.

Идентификатором оператора называется одно слово или последовательность из нескольких ключевых слов, которая определяет функцию оператора. Например:

GOTO, DECLARE, GET.

Описателями (атрибутами) называются ключевые слова, указывающие свойства данных, процедур и других элементов языка. Например:

FLOAT — плавающий;
SEQUENTIAL — последовательный.

Разделяющие ключевые слова используются для разделения отдельных частей операторов IF (если) и DO (выполнить). Они состоят из пяти ключевых слов: THEN (тогда), ELSE (иначе), BY (шаг), TO (до), WHILE (пока).

Именем встроенной функции называется ключевое слово, являющееся названием соответствующей функции, которая имеется в языке и доступна для использования программистом.

Ситуация — это ключевое слово, которое используется в операторе ON (при), а также в качестве приставки к другим операторам. Программист может задать специальные действия, которые должны быть выполнены при возникновении той или иной ситуации. Например:

OVERFLOW — переполнение;
ZERODIVIDE — деление на нуль.

Использование пробелов. Пробелы достаточно широко могут использоваться при написании программы на языке ПЛ/1. Они могут ставиться по обе стороны от знаков операций и большинства ограничителей. При этом там, где может использоваться один пробел, разрешается устанавливать любое количество пробелов.

Пример.

B + C эквивалентно B _ + _ C или B __ + __ C.
(A + B) эквивалентно (_ A __ + B __).

Однако идентификаторы, константы, спецификации шаблоном, составные знаки операций (например, —> или <=) не могут содержать пробелов. Пробелы допускаются внутри констант типа строки символов.

Идентификаторы, константы и спецификации шаблоном не могут непосредственно примыкать друг к другу. Они должны отделяться каким-нибудь знаком операции, разделителем, пробелом или примечанием.

Комментарии. Комментарии служат для пояснения и документирования программы и могут быть вставлены в любом месте программы, где стоят пробелы, но не в константу типа строки символов. Комментарии игнорируются транслятором и поэтому не оказывают никакого влияния на решение задачи.

ЕС ЭВМ ПЛ-1	Программа	Программист Дата_____ Лист_____ Листов_____

Операторы исходной программы

1 5 10 15 20 25 30 35 40 45 50 55 60 65 70 72

Идентификация

73 80

Рис. 1.1. Форма бланка для программы, написанной на языке ПЛ/1.

Комментарии используются для лучшей читабельности программы, отделения одной части программы от другой и описания их содержания, а также для пояснения используемых в программе величин. Общий формат комментария представляет собой:

/*строка символов*/.

Например, в операторе A2 = M + /* производится сложение двух переменных */B;

1.3. Правила написания программы на бланке

Содержание операторов программы записывается на специальном бланке, затем набивается на перфокарты, а с них вводится в память машины. Стандартный бланк для программы содержит аналогично перфокарте 80 позиций, при этом каждая строка бланка соответствует одной перфокарте. Положение некоторого символа внутри строки соответствует положению его в колонке перфокарты (рис. 1.1).

В пределах строки бланка для написания операторов программисту предоставляется почти полная свобода. Однако должны соблюдаться следующие правила:

— первая колонка бланка не должна заполняться;

— колонка 2—72 могут использоваться для написания операторов программы;

— колонки 73—80 транслятором игнорируются и могут содержать любую информацию или замечания.

Совершенно безразлично, на каком месте начнется или окончится написание оператора, однако, выбрав наиболее удобное расположение операторов и меток, программист может улучшить читабельность программы.

Глава 2

ПРАВИЛА ОПИСАНИЯ И ИСПОЛЬЗОВАНИЯ ЭЛЕМЕНТОВ ДАННЫХ В ЯЗЫКЕ

Информация, обрабатываемая во время выполнения рабочей программы, называется данными. Каждый элемент данных имеет определенный тип и изображение.

По способу организации данные делятся на скалярные величины (отдельные элементы данных) и агрегаты элементов данных (группы элементов данных).

Элемент данных может быть или константой, или скалярной переменной. Константы и скалярные переменные называются скалярными величинами.

Константа есть элемент данных, который обозначает самого себя, т. е. изображение константы является как ее именем, так и ее значением; таким образом, она не может изменяться во время выполнения программы.

Скалярная переменная, подобно константе, обозначает элемент данных. Однако в отличие от константы переменная может принимать более чем одно значение за время выполнения программы. Совокупность значений, которые может принимать переменная, называется областью значений этой переменной. Область значений переменной всегда ограничена одним типом данных (а в случае арифметического типа — одними и теми же основанием системы счисления, способом представления и разрядностью). Если никаких дополнительных ограничений на область значения не наложено, то переменная может принимать значения всей совокупности данных соответствующего типа. Обращение к скалярной переменной осуществляется с помощью ее имени, которое может быть простым именем, именем с индексами, составным именем или составным именем с индексами.

В языке ПЛ/1 элементы данных могут быть объединены в группы — массивы или структуры.

Массив есть n-мерная упорядоченная совокупность элементов с одинаковыми характеристиками (атрибутами).

Если элементы массива принадлежат к арифметическому типу, они должны иметь одни и те же основания системы счисления, способ представления и разрядность или один и тот же шаблон.

Если элементы массива являются строками битов или строками символов, они должны иметь одну и ту же длину.

Структура есть иерархически упорядоченная совокупность скалярных переменных, массивов и структур. Элементы структуры могут не принадлежать к одному и тому же типу данных или иметь одинаковые характеристики.

2.1. Описание арифметических данных

Допустимые в языке данные делятся на два типа:

— проблемные данные;

— данные управления программой.

Проблемные данные представляют собой величины, обрабатываемые в программе, и разделяются на арифметические и строковые данные.

Обращение к элементу данных производится с помощью переменной или константы. Обращение к элементу данных с помощью переменной требует присвоения этой переменной некоторых характеристик.

Присвоение характеристик переменной производится оператором DECLARE (сокращенно DCL) с помощью атрибутов. Формат оператора:

```
DECLARE идентификатор [атрибут [атрибут] ...]
        [, идентификатор [атрибут [атрибут] ...] ...];*)
```

где идентификатор — это имя объявляемой переменной.

Если в операторе объявлены не все атрибуты, то транслятор осуществляет дополнение атрибутов. Эти дополняемые атрибуты называются атрибутами по умолчанию.

Арифметические данные. К ним относятся:

— десятичные данные с фиксированной точкой;

— десятичные данные с плавающей точкой;

— двоичные данные с фиксированной точкой;

— двоичные данные с плавающей точкой.

Арифметические данные имеют числовое значение, характеризуемое основанием системы счисления, способом представления и разрядностью. Элемент данных может быть представлен либо в форме числового поля, либо в кодовой форме (т. е. во внутреннем коде).

Числовое поле — это цифровая строка знаков, которая получает числовую интерпретацию с помощью атрибута PICTURE (шаблон). Основание системы счисления, способ представления и разрядность определяются спецификацией шаблона числового поля.

---

*) Квадратные скобки и многоточие не являются символами языка ПЛ/1 и используются в форматах со специальными целями. Квадратные скобки указывают, что конструкция, помещенная в них, может быть опущена. Многоточие указывает, что стоящая перед ними конструкция может повторяться произвольное число раз.

Элемент данных в *кодовой форме* характеризуется атрибутами, определяющими основание системы счисления, способ представления и разрядность.

Основание. Арифметическому данному может быть присвоено десятичное основание системы счисления с помощью атрибута DECIMAL (сокращенно DEC) или двоичное основание системы счисления с помощью атрибута BINARY (сокращенно BIN). Если при объявлении данного ничего не указано, то предполагается атрибут по умолчанию DECIMAL.

Способ представления. Арифметические данные могут быть представлены либо с фиксированной, либо с плавающей точкой с помощью атрибутов соответственно FIXED или FLOAT. В случае отсутствия атрибута предполагается FLOAT. Элементы данных, представленные с фиксированной точкой, являются рациональными числами, для которых определено количество десятичных или двоичных цифр; элементы данных с плавающей точкой являются рациональными числами, представленными в виде мантиссы и порядка. Например:

```
DECLARE A FIXED BINARY (3);
```

Этим оператором DECLARE программист объявляет, что идентификатор A имеет атрибуты FIXED, BINARY и (3), т. е. идентификатор A представляет собой имя двоичной переменной с фиксированной точкой, величина которой не превышает трех двоичных цифр.

Разрядность. Если для атрибутов основания и способа представления установлены ключевые слова, то для атрибута разрядности их нет. Атрибут разрядности имеет следующую общую форму:

(p, q) — для десятичных переменных с фиксированной точкой,

где p — указывает общее количество десятичных цифр, допустимое для переменной;

q — масштаб, который дает возможность программисту установить нужное положение десятичной точки. Он определяет количество цифр дробной части значения (т. е. справа от десятичной точки);

(p) — для двоичной переменной с фиксированной точкой, а также для десятичных и двоичных чисел с плавающей точкой.

Двоичные данные могут быть только целыми, поэтому нет необходимости указывать положение десятичной точки (т. е. q всегда равно нулю).

Для десятичных и двоичных чисел с плавающей точкой этим атрибутом программист определяет максимальное количество значащих цифр мантиссы, участвующих в обработке.

В табл. 2.1 для элементов атрибута разрядности показаны максимально допустимые значения и значения по умолчанию.

Таблица 2.1

Способ представления	Основание	Общая форма атрибута разрядности	Атрибут разрядности	
			по умолчанию	максимальное значение
FIXED	DECIMAL	(*p*, *q*)	(5,0)	(15,*q*)
FIXED	BINARY	(*p*)	(15)	(31)
FLOAT	DECIMAL	(*p*)	(6)	(16)
FLOAT	BINARY	(*p*)	(21)	(53)

При объявлении элементов арифметических данных последовательность атрибутов основания, способа представления и разрядности может быть любой за исключением того, что атрибут разрядности не может быть первым атрибутом после объявленного идентификатора.

Пример. Необходимо объявить три переменные VAR1, VAR2, VAR3.
VAR1 — десятичная переменная с фиксированной точкой, значение которой не превышает четырех цифр в целой части и двух в дробной;
VAR2 — целая десятичная переменная с фиксированной точкой, значение которой не превышает пяти цифр;
VAR3 — двоичная переменная с фиксированной точкой, значение которой не превышает десяти двоичных цифр.

```
DECLARE VAR1 FIXED DECIMAL (6, 2);
DECLARE VAR2 FIXED DECIMAL (5,0);
DECLARE VAR3 FIXED BINARY (10);
```

Переменная VAR1, объявленная таким образом, может принимать значение в форме ± *dddd.dd*, где *d* — любая десятичная цифра. Область значений для VAR1:

$$-9999.99 \leqslant \text{VAR1} \leqslant +9999.99$$

Объявленная переменная VAR2 может принимать значение в форме ±*ddddd*. Область значения для VAR2:

$$-99999 \leqslant \text{VAR2} \leqslant +99999$$

При этом видно, что переменная VAR2 с помощью атрибута разрядности (5,0) определена как целое число. В этом случае *q* можно опустить, и атрибут разрядности будет содержать только элемент *p*. Тогда оператор будет иметь вид

```
DECLARE VAR2 FIXED DECIMAL (5);
```

Объявленная переменная VAR3 является двоичной переменной с фиксированной точкой, в которой может размещаться двоичное число вида ± *bbbbbbbbbb*, где *b* — двоичная цифра (0 или 1). Область значений VAR3

$$-1111111111 \leqslant \text{VAR3} \leqslant +1111111111.$$

Можно все объявления переменных записать в одном операторе:

```
DCL VAR1 FIXED (6,2),
    VAR2FIXED (5),
    VAR3 FIXED BIN (10);
```

В данном случае использован принцип умолчания и возможные сокращения ключевых слов.

Программа может содержать любое количество операторов DECLARE. Однако все атрибуты, относящиеся к одной переменной, должны содержаться в одном операторе. В то же время с помощью одного оператора DECLARE может быть описано любое число переменных, даже если какие-нибудь их атрибуты отличаются друг от друга. Отдельные описания в этом случае отделяются друг от друга запятыми.

Пример.

```
DCL  A BIN FIXED (16),
     B DEC (12),
     Z FIXED DEC (8,4);
```

Если совпадают атрибуты у нескольких переменных, то можно избежать повторения описаний. Все идентификаторы переменных, имеющих одинаковые атрибуты, заключаются в скобки, а друг от друга отделяются запятыми. Объединение описаний возможно также и в том случае, когда совпадают не все атрибуты.

Пример.

```
DCL (A, B, C, D) BIN FIXED;
DCL (X DEC (6,2), Y BIN (12)) FIXED;
DCL ((A, B) FIXED (8), K FLOAT (6)) DEC;
```

Если в операторе DECLARE для переменной не указаны никакие атрибуты, то транслятор присваивает их на основании первого символа имени переменной. Если имя переменной начинается с букв I, J, K, L, M, N, то ей присваиваются атрибуты FIXED BINARY (15), во всех остальных случаях FLOAT DECIMAL (6).

Аналогично транслятор поступает и в том случае, если переменная не объявляется с помощью оператора DECLARE, а используется в операторах программы.

Пример.

```
DCL ALPHA, LAMBDA;
```

Здесь переменной ALPHA будут присвоены атрибуты по умолчанию FLOAT DECIMAL (6), а переменной LAMBDA—FIXED BINARY (15)

Арифметические константы сами определяют значение и поэтому не нуждаются в имени. В отличие от переменных, начальное значение которым должно быть присвоено в процессе выполнения программы, константы получают свое значение уже при трансляции программы. Транслятор присваивает им атрибуты, которые вытекают из формы записи констант. Кроме того, константе присваивается внутреннее имя, которое используется для обращения к ней во время выполнения программы.

Десятичная константа с фиксированной точкой может состоять из одной или нескольких десятичных цифр с необязательной десятичной точкой и необязательным знаком.

Если в константе нет десятичной точки, то она предполагается стоящей справа после последней цифры числа и в этом случае константа — целое число. Если в константе нет знака, то она считается положительной.

Для чисел, меньших единицы, цифра нуль перед десятичной точкой может опускаться.

Пример

Значение константы	Атрибуты, присваиваемые транслятором
+31 0	FIXED DEC (3,1)
678	FIXED DEC (3)
—008	FIXED DEC (3)
+ 00625	FIXED DEC (5,5)
—2.18805	FIXED DEC (6,5)

Двоичная константа с фиксированной точкой может состоять из одной или нескольких двоичных цифр, за которыми непосредственно следует буква B.

Пример

Значение константы	Атрибуты, присваиваемые транслятором
+1000B	FIXED BIN (4)
—10B	FIXED BIN (2)
+1B	FIXED BIN (1)
—1010101B	FIXED BIN (7)

Использование в константе двоичной точки не допускается. Она предполагается стоящей справа от самой правой двоичной цифры. Двоичная константа может иметь знак.

Десятичная константа с плавающей точкой состоит из двух частей—мантиссы и порядка. Мантисса представляет собой десятичную константу с фиксированной точкой. Порядок состоит из одной или двух десятичных цифр и присоединяется к мантиссе с помощью символа E (без пробелов). Как мантисса, так и порядок могут иметь знак.

Пример

Значение константы	Атрибуты, присваиваемые транслятором
26E6	FLOAT DEC (2)
—0.1E—02	FLOAT DEC (2)
4895.0E0	FLOAT DEC (5)
+005192E5	FLOAT DEC (6)

Двоичная константа с плавающей точкой также состоит из мантиссы и порядка. Мантисса выражается двоичной константой с фиксированной точкой, в которой может присутствовать двоичная точка. Порядок начинается с буквы E и может состоять из одной, двух или трех десятичных цифр, являющихся целой степенью двух. Оканчивается константа буквой B. Как мантисса, так и порядок могут иметь знак.

Пример

Значение константы	Атрибуты, присваиваемые транслятором
1010E5B	FLOAT BIN (5)
10.101E—6B	FLOAT BIN (5)
1110010E+28B	FLOAT BIN (7)

2.2. Внутреннее представление арифметических данных

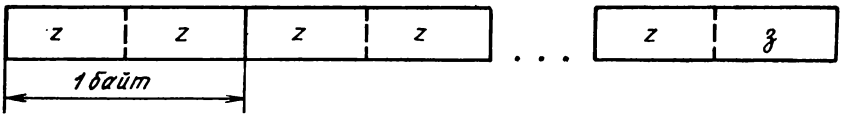
При описании задач на языке ПЛ/1 программист должен представлять, какой объем памяти будет выделяться для размещения объявленных в программе арифметических данных. Это позволяет ему более рационально использовать имеющуюся в его распоряжении основную память, а также рассчитать длину записи при создании наборов данных на внешних носителях. С этой целью рассмотрим более подробно внутреннюю форму представления арифметических данных.

Для десятичной переменной с фиксированной точкой внутренняя кодированная форма упакованная десятичная, т. е. запоминается по две десятичные цифры в байте, кроме самого правого байта, в котором содержится десятичная цифра и знак.

При этом каждая десятичная цифра, а также знак числа изображаются двоичной комбинацией из четырех бит. Для цифр и знака эти комбинации имеют вид

0	0000	5	0101
1	0001	6	0110
2	0010	7	0111
3	0011	8	1000
4	0100	9	1001
+	1100	(шестнадцатиричное C)	
—	1101	(шестнадцатиричное D)	

Схематически внутреннее представление десятичного числа с фиксированной точкой можно изобразить так:



где z — цифра; з — знак числа.

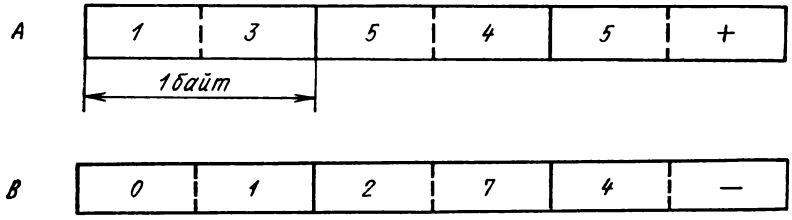
Число в память записывается справа налево, а если остается лишний полубайт, то он заполняется нулем. Отрицательные числа представляются так же, как положительные, отличаясь только знаком.

Для дробных чисел представляются лишь цифры без десятичной точки. Положение точки записывается транслятором в специальное поле описания данного и учитывается при каждом обращении к нему. Например, числа 124, 12.4 и 1.24 имеют одинаковое внутреннее представление.

Пример. Переменные A и B описаны в программе так:

```
DCL A FIXED (5,1),
    B FIXED (4,2);
A = 1354.5;     B = —12.74;
```

Эти переменные расположатся в памяти следующим образом:

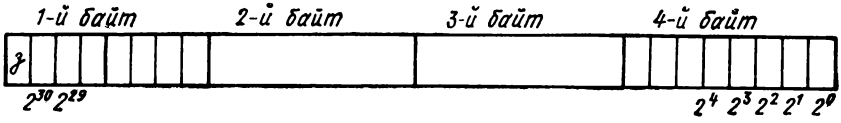


Для двоичной переменной с фиксированной точкой и любым атрибутом разрядности внутренняя кодированная форма есть полное двоичное слово (четыре байта), эти данные должны быть всегда целыми.

Положение каждой двоичной цифры справа налево соответствует степеням 2^0, 2^1, 2^2, ..., 2^{30}. Следовательно, позиции под число

отводятся также справа налево. Оставшиеся позиции для положительного числа содержат нули.

Схематически внутреннее представление двоичного числа с фиксированной точкой можно изобразить так:



В знаковой позиции для положительного числа содержится нуль. Отрицательное число обычно представляется в форме дополнения. Дополнение получают путем замены нулей на единицы и наоборот, а затем к результату прибавляется единица. В знаковой позиции, как и в позициях, занятых нулями, для отрицательного числа появляется единица.

На основании изложенного самое большое положительное число $2^{31} - 1 = 2147483647$, а отрицательное $2^{31} = -2147483648$.

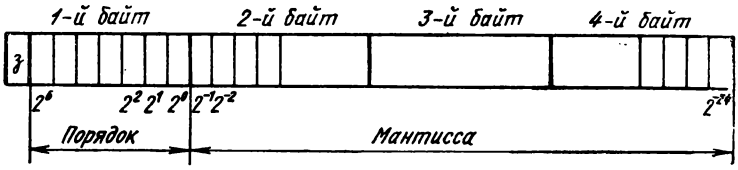
Для двоичной и десятичной переменных с плавающей точкой внутренняя кодированная форма является нормализованной шестнадцатиричной формой с плавающей точкой. Всякое число в форме с плавающей точкой состоит из двух частей — мантиссы и порядка. Мантисса содержит значащие цифры, а с помощью порядка указывается положение десятичной точки. Обе части числа с плавающей точкой хранятся вместе в четырех последовательных байтах (полное слово).

Для хранения порядка выделяется 7 правых битов 1-го байта. Порядок числа представляется в двоичной системе счисления. Для того чтобы наряду со знаком числа не требовалось хранить знак порядка, он всегда представляется на 64 больше, чем на самом деле. Ввиду того, что семью битами можно описать числа от 0 до 127, в действительности порядок может принимать значение от —64 до +63.

Мантисса также представляется в двоичной системе счисления и хранится в 24 битах последующих трех байтов. Точка всегда подразумевается перед самым левым битом. Соответственно этому значения цифр в двоичных позициях мантиссы слева направо: 2^{-1}, 2^{-2}, ..., 2^{-24}. В шестнадцатиричной системе счисления мантисса выразится числом позиций 16^{-1}, 16^{-2}, ..., 16^{-6}.

Отрицательное число не представляется в форме дополнения, а для знака мантиссы (т. е. знака числа) используется самый левый бит 1-го байта. Для положительного числа он содержит 0, для отрицательного 1.

Схематически внутреннюю форму представления числа с плавающей точкой можно изобразить следующим образом:



Значение числа с плавающей точкой равно

$$\pm\text{мантисса}\ 16^{\text{порядок}-64}.$$

Следовательно, значения чисел охватывают

$$5{,}4 \cdot 10^{-78} \leqslant z \leqslant 7{,}2 \cdot 10^{75}.$$

Таким образом, с помощью 24 битов, предназначенных для мантиссы, может быть представлено лишь 6-значное десятичное число. Для получения повышенной точности допустимо в случае необходимости использовать еще четыре байта, 32 бита которых служат для продолжения мантиссы и соответствуют позициям 2^{-25}, 2^{-26}, ... 2^{-56}. С помощью этих дополнительных битов можно представить 16-значное десятичное число.

Если у переменной с атрибутами FLOAT DECIMAL объявленная разрядность $p \leqslant 6$, применяется короткий формат числа с плавающей точкой (одно полное слово); если же объявленная разрядность $p > 6$, применяется длинный формат числа с плавающей точкой (двойное слово).

Если у переменной с атрибутами FLOAT BINARY объявленная разрядность $p \leqslant 21$, то применяется короткий формат числа с плавающей точкой; если же разрядность $p > 21$, применяется длинный формат числа с плавающей точкой.

Пример. Если переменная B с плавающей точкой объявляется как DCL B DECIMAL (3); то для этой переменной транслятором резервируется одно слово, т.е. 4 байта.

2.3. Арифметические операции

При обработке данных в программе они могут использоваться в арифметических выражениях. Простое (или скалярное) арифметическое выражение строится из констант, переменных и функций с помощью знаков арифметических операций и круглых скобок. Оно задает порядок выполнения действий над элементами данных для получения некоторого конкретного числового значения. Отдельные данные, входящие в выражение, называются операндами.

В языке ПЛ/1 используются следующие арифметические операции:

+ сложение, — вычитание, * умножение, /деление и ** возведение в степень.

Арифметические операции бывают одно- и двухместные. Одноместная операция (+ или —) действует на один операнд, записанный непосредственно за ней. Двухместная операция связывает два операнда, между которыми она записана.

Все арифметические операции выполняются над кодовыми данными. Поэтому, если необходимо, данные преобразуются в кодовый арифметический тип перед выполнением операции.

Особо следует отметить, что в языке ПЛ/1 в арифметических выражениях в качестве операнда можно использовать строку битов, которая интерпретируется транслятором как двоичное число с фиксированной точкой с атрибутами FIXED BINARY (31).

При отсутствии круглых скобок установлен следующий порядок старшинства арифметических операций: первыми выполняются операции возведения в степень (**), затем все умножения и деления (*, /) и, наконец, все сложения и вычитания (+, —).

Операции в выражении выполняются в последовательности указанных приоритетов. Если друг за другом следуют операции с одинаковым приоритетом, то операция возведения в степень выполняется справа налево, все остальные операции выполняются последовательно слева направо. Выражения в круглых скобках вычисляются в первую очередь.

Два операнда в арифметической операции могут отличаться по типу, основанию системы счисления, способу представления и разрядности. Перед выполнением операции атрибуты операндов проверяются и, если это необходимо, автоматически преобразуются к некоторому единому представлению. Однако каждое преобразование данного требует дополнительного времени и памяти (для включения в рабочую программу соответствующих специальных подпрограмм библиотеки ПЛ/1). Кроме того, многочисленные преобразования могут привести к потере значащих разрядов и получению неверных результатов вычислений. Поэтому при объявлении переменных, используемых в программе, программист должен обратить внимание на то, чтобы такие преобразования при вычислении выражений выполнялись как можно реже. Преобразование данных в арифметических операциях происходит по следующим правилам.

Тип. Цифровые знаковые данные и строки битов преобразуются в арифметический тип. Результат арифметической операции получается всегда в арифметической кодовой форме.

Основание. Если основания системы счисления у двух операндов различны, то операнд с десятичным основанием переводится в двоичную систему счисления.

Способ представления Если операнды различаются способом представления, то операнд с фиксированной точкой преобразуется в форму представления с плавающей точкой.

Исключением из этого правила является случай возведения в степень, если первый операнд представлен в форме с плавающей точкой, а второй — в форме с фиксированной точкой с количеством

дробных разрядов, равным нулю, т. е. его значение является целым числом. В этом случае никаких преобразований не производится, а результат будет с плавающей точкой.

Если оба операнда операции возведения в степень представлены в форме с фиксированной точкой, то преобразование осуществляется следующим образом:

— оба операнда преобразуются в форму с плавающей точкой, если второй операнд имеет разрядность, отличную от (p, 0);

— первый операнд преобразуется в форму с плавающей точкой, исключая случай, когда второй операнд — целая константа без знака;

— первый операнд преобразуется в форму с плавающей точкой, если разрядности операндов таковы, что разрядность результата возведения в степень с фиксированной точкой превышает максимально допустимое число разрядов (15 десятичных, 31 двоичных).

Пример. В некоторой программе объявлены следующие переменные:

```
DCL A FIXED (2),
    B FIXED (3,2),
    C FLOAT (5),
    D FLOAT (7),
    F FIXED (15);]
```

При выполнении операции возведения в степень могут быть следующие случаи:

D**C — преобразования не будет, так как оба операнда представлены в форме с плавающей точкой.

A**4 — преобразования не будет, так как второй операнд — целая константа без знака и результат не превысит 15 разрядов.

D**5 — преобразования не будет, так как первый операнд представлен в форме с плавающей точкой, а второй — с фиксированной точкой и разрядностью (p, 0).

F**A — первый операнд преобразуется в число с плавающей точкой, так как разрядность результата операции превышает максимально допустимую разрядность для десятичных чисел с фиксированной точкой.

D**B — второй операнд преобразуется в форму с плавающей точкой, потому что его атрибут разрядности не равен (p, 0).

Разрядность. Если операнды отличаются только разрядностью, то никакого преобразования не производится. Разрядность результата определяется на основании разрядности операндов.

Арифметическая операция выполняется только после того, как произведены определенные преобразования. Если в процессе выполнения операции произойдет превышение максимально допустимой разрядности, то все усечения делаются за счет отбрасывания младших дробных разрядов, участвующих в операции операндов независимо от основания и способа их представления. Однако в некоторых случаях для данных с фиксированной точкой могут усекаться и старшие разряды целой части (в этом случае происходит

прерывание программы, если включена ситуация SIZE). Прерывания при выполнении программы будут рассмотрены в гл. 10.

Основание, способ представления и разрядность результата зависят от операндов и вида операции следующим образом.

Одноместные операции плюс и минус дают результат, имеющий основание, способ представления и разрядность операнда.

Плавающая точка. Если операнды в двухместной операции оба с плавающей точкой, то результат будет тоже с плавающей точкой, а основанием результата является основание, общее для операндов. Разрядностью результата является наибольшая из разрядностей двух операндов.

Фиксированная точка. Если операнды двухместной операции представлены с фиксированной точкой и операция не является возведением в степень, то результат имеет фиксированную точку, а основанием результата является основание, общее для операндов. Разрядность результата зависит от операции и разрядности операндов и вычисляется в соответствии с правилами, указанными в приложении 1. Рассмотрим на примерах, как используются приведенные таблицы.

В формулах для вычисления разрядности используются следующие обозначения:

p — число цифр результата;
q — число дробных разрядов результата;
p_1 — число цифр в первом операнде;
q_1 — число дробных разрядов в первом операнде;
p_2 — число цифр во втором операнде;
q_2 — число дробных разрядов во втором операнде.

Пример. Сложить две десятичные переменные с фиксированной точкой, значения которых изменяются в следующих пределах:

$$980.0000 \leqslant X \leqslant 12500.000$$
$$3.00000 \leqslant Y \leqslant 218.00000$$

Как нужно объявить переменные X и Y и результат сложения Z?

Переменная X должна объявляться с атрибутами FIXED DECIMAL (9,4). Переменная Y — с атрибутами FIXED DECIMAL (8,5).

Атрибут разрядности переменной Z вычисляется по формулам (приложение 1, табл. 1)

$$p = 1 + \max(p_1 - q_1,\ p_2 - q_2) + \max(q_1,\ q_2);$$
$$q = \max(q_1,\ q_2).$$

В примере $p_1 = 9$, $q_1 = 4$, $p_2 = 8$, $q_2 = 5$ и отсюда $p = 11$, $q = 5$.

Переменную Z можно объявить с атрибутами FIXED (11,5).

При вычитании десятичных операндов с фиксированной точкой используются те же формулы.

Пример. Умножить две десятичные переменные с фиксированной точкой, значения которых изменяются в следующих пределах:

$$80.000 \leqslant M1 \leqslant 600.000$$
$$1.00 \leqslant M2 \leqslant 80.00$$

Какая должна быть разрядность у M1, M2 и результата умножения M?

M1 объявляется с атрибутами FIXED DECIMAL (6,3). M2 объявляется с атрибутами FIXED DECIMAL (4,2).

Атрибут разрядности переменной M можно вычислить с помощью формул (приложение 1, табл. 2)

$$p = p_1 + p_2 + 1;$$
$$q = q_1 + q_2.$$

В примере $p_1 = 6$, $q_1 = 3$, $p_2 = 4$, $q_2 = 2$ и отсюда $p = 11$, $q = 5$.

Переменную M можно объявить с атрибутами FIXED (11,5).

Пример. Десятичную переменную с фиксированной точкой X, значение которой изменяется от 10.000 до 500.000, нужно разделить на 2. Какова разрядность результата Y?

Переменная X объявляется с атрибутами: FIXED DECIMAL (6,3). Атрибуты константы 2—FIXED DECIMAL (1).

Атрибуты разрядности результата вычисляются по формулам (приложение 1, табл. 3)

$$p = \begin{cases}15 \text{ для десятичного основания}\\31 \text{ для двоичного основания}\end{cases}$$
$$q = \begin{cases}15 \text{ для десятичного основания}\\31 \text{ для двоичного основания}\end{cases}$$

Для нашего примера $p = 15$, $q = 15 - (6 - 3) + 0 = 12$.

Атрибут результата Y будет FIXED (15,12).

Пример. Возвести в степень число с фиксированной точкой X, атрибуты которого FIXED DECIMAL (2,0). Показатель степени — константа (3) с атрибутами FIXED DECIMAL (1).

Разрядность результата вычисляется по формулам (приложение 1, табл. 4)

$$p = (p_1 + 1) * n - 1;$$
$$q = q_1 * n.$$

В нашем примере $p_1 = 2$ и $n = 3$, а результат

$$p = (2 + 1) * 3 - 1 = 8;$$
$$q = 0 * 3 = 0.$$

Атрибут результата FIXED (8, 0).

Для операции возведения в степень X**Y укажем несколько специальных случаев, а именно:

а) $X = 0$, $Y > 0$, результат равен 0;

б) $X = 0$, $Y \leqslant 0$, возникает исключительная ситуация ERROR (ошибка) и происходит прерывание программы;

в) $X \neq 0$, $Y = 0$, результат равен 1;

г) X = 0, Y не является переменной с фиксированной точкой с разрядностью (p, q), возникает исключительная ситуация ERROR, и происходит прерывание программы.

2.4. Строковые данные

К проблемным данным, кроме арифметических, относятся строковые данные. Строковые данные — это константы и переменные, не имеющие цифровых значений и характеризующиеся лишь последовательностью (строкой) отдельных символов или двоичных

цифр. Строковые данные, как правило, используются при программировании логических, информационных и текстовых алгоритмов. Они позволяют достаточно просто сформировать для выдачи на печать любую форму необходимого документа.

Строковые данные разделяются на два типа:

— символьные строковые данные;

— битовые строковые данные.

Длина элемента строковых данных равна числу символов (для строки символов) или числу двоичных цифр (для строки битов) в элементе. Элемент строковых данных с нулевой длиной называется пустой строкой.

Символьные строковые данные (или данные типа строки символов) представляют собой непрерывную последовательность любых символов, реализованных на машине ЕС ЭВМ (код ДКОИ), в том числе и заглавных букв русского алфавита. Символьные строковые данные используются обычно при выдаче на печать текста. Однако они могут также участвовать во многих случаях обработки алфавитной информации.

Переменные типа строки символов объявляются следующим оператором:

DECLARE идентификатор CHARACTER (положительное число);

Ключевое слово CHARACTER (сокращенно CHAR) указывает, что тип данного — строка символов.

Атрибут символьной строки состоит из ключевого слова CHARACTER с указанием в круглых скобках длины строки в символах. В памяти каждому символу в строке соответствует один байт. Максимальная длина символьной строки 255 символов.

Пример.

```
DCL B CHARACTER (8);
```

Транслятор резервирует для переменной B память, как для символьной строки длиной в 8 байтов.

Если значение символьной переменной содержит меньше символов, чем в указанном размере данного, то оно занимает место слева в зарезервированном поле, а лишние байты справа заполняются пробелами. Если значение символьной переменной оказалось большим, чем указано размером данного, то «лишние» правые символы отсекаются. Необходимо отметить, что прерывания программы при отсечении «лишних» символов не будет.

Пример.

```
DCL (A,B) CHAR (6);
A = 'ABC'; B = 'АВТОМОБИЛИСТ'
```

В первом случае в шести байтах памяти, зарезервированных для переменной A, будет содержаться 'ABC___'. Во втором случае в памяти переменной B будет содержаться 'АВТОМО'.

Символьная строковая константа представляет собой строку, содержащую любые допустимые в языке символы и заключенную в кавычки. Открывающая и закрывающая кавычки служат лишь для идентификации константы. Они не входят в состав константы, и место в памяти для них не резервируется. Если желательно представить кавычку (как символ из алфавита для представления обрабатываемой информации), то это делается двумя кавычками, непосредственно следующими одна за другой, при подсчете длины строки они рассматриваются как один символ.

Пример

Значение константы	Атрибуты, присваиваемые транслятором
'ЛЕНИНГРАД'	CHARACTER (9)
'КИНОТЕАТР "ЗЕНИТ"'	CHARACTER (16)
'DYHON—KAR'	CHARACTER (9)

Иногда символьная строковая константа может состоять из повторяющейся комбинации символов. При помощи коэффициента повторения запись строки может быть сокращена. Коэффициент повторения представляется целым положительным числом, заключенным в круглые скобки и стоящим непосредственно перед самой символьной строковой константой.

Пример.

(3)'M' соответствует 'MMM'

(2)'ABCD' соответствует 'ABCDABCD'.

Битовые строковые данные (или данные типа строки битов) представляют собой строки, состоящие из последовательности двоичных цифр (0 или 1). Битовые строковые данные используются чаще всего в логических операциях.

Переменные типа строки битов объявляются следующим образом:

DECLARE идентификатор BIT (положительное число);

Положительное число, стоящее в круглых скобках, задает количество двоичных цифр в битовой строке. Максимальная длина битовой строки 64.

Пример.

DECLARE L BIT (10);

Битовая строковая константа представляет собой заключенную в кавычки строку, содержащую двоичные цифры (0 и 1), за которой непосредственно следует буква B. Перед константой может стоять

в круглых скобках десятичное целое число, обозначающее число повторений.

Пример

Значение константы	Атрибут, присваиваемый транслятором
'1'B	BIT (1)
'101110011100'B	BIT (12)
(64)'0'B	BIT (64)

Битовая строка при хранении в памяти выравнивается на границу байта и занимает целое число байтов.

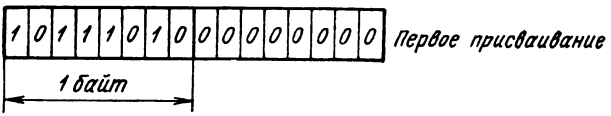
Так, в нашем примере для переменной L, которая имеет длину 10 битов, транслятор зарезервирует 16 битов, т. е. два байта памяти.

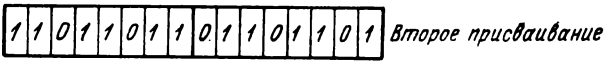
Как и в случае строки символов, заполнение строки битов происходит слева направо. Если длина строки больше, чем длина переменной, то отсекается значение правых битов, если меньше, происходит дополнение строки нулями.

Пример.

```
DCL M BIT (10);
    M = '1011101'B;
    M = '1101101101101101111'B;
```

На основании объявления DECLARE для переменной M транслятор резервирует 2 байта памяти. После первого присваивания свободные биты справа заполняются нулями. При втором присваивании строка битов отсекается справа





Операция сцепления. Во многих случаях при программировании задач обработки текста бывает необходимо соединять отдельные символы или группы символов. Для этого предназначена операция сцепления, с помощью которой осуществляется присоединение двух строк без промежуточного пробела (для строк символов) или промежуточного нуля (для строк битов).

Общий вид операции сцепления:

операнд || операнд [|| операнд] ...

Операндами операции сцепления могут быть строки символов, строки битов и цифровые строки знаков. Данные арифметического типа в качестве операндов операции сцепления участвовать не могут. Исключение составляют двоичные данные с фиксированной точкой, которые могут быть сцеплены со строками битов.

Возможные случаи и результаты операции сцепления в зависимости от типа операндов показаны в табл. 2.2.

Таблица 2.2

Первый операнд	Второй операнд		
	Строка символов	Строка битов	Цифровая строка знаков
Строка символов	Результат есть строка символов. Длина равна сумме длин операндов (максимальная длина 255 символов)	Перед сцеплением производится преобразование строки битов в строку символов. Результат тот же, что и при сцеплении двух строк символов	Перед сцеплением производится преобразование цифровой строки знаков в строку символов. Результат тот же, что и при сцеплении двух строк символов
Строка битов	Перед сцеплением производится преобразование строки битов в строку символов. Результат тот же, что и при сцеплении двух строк символов	Результат есть строка битов. Длина равна сумме длин операндов (максимальная длина равна 64 битам)	Перед сцеплением производится преобразование цифровой строки знаков. Результат тот же, что и при сцеплении двух строк битов
Цифровая строка знаков	Перед сцеплением производится преобразование цифровой строки знаков в строку символов. Результат тот же, что и при сцеплении двух строк символов	Перед сцеплением производится преобразование цифровой строки знаков в строку битов. Результат тот же, что и при сцеплении двух строк битов	Цифровая строка знаков преобразуется в строку символов. Результат тот же, что и при сцеплении двух строк символов

Примеры.

1) A и B объявлены как две строки символов длиной 5 и 12, а C — строка символов длиной 17.

```
DCL A CHAR (5),
    B CHAR (12),
    C CHAR (17);
```

После выполнения операторов:

```
A = 'AUTOR';
B = 'CONSTRUCTION';
C = A || B;
```

переменная типа строка символов C будет иметь значение

'AUTORCONSTRUCTION'

Операция сцепления	Результат
'1101'B || '10'B	'110110'B
'ABC' || 'DEF'	'ABCDEF'
'1101'B || '10'B	'10XYZ'

3) В программе объявлена переменная NR:

DCL NR PICTURE 'ZZ9';

и выполняется следующая операция сцепления:

'ОШИБОЧНОЙ__ЯВЛЯЕТСЯ_' || NR || '_ПЕРФОКАРТА'.

Если в процессе выполнения программы переменная примет значение 27 то результатом операции сцепления будет следующая строка символов:

'ОШИБОЧНОЙ__ЯВЛЯЕТСЯ__27_ПЕРФОКАРТА'.

2.5. Операции сравнения и логические операции

Как при описании алгоритмов на обычном математическом языке, так и в языке ПЛ/1 часто используются выражения типа сравнения. Выражение типа сравнения связывает два операнда при помощи знака операции сравнения, производит сравнение этих двух операндов и определяет, истинно значение выражения или ложно.

Результат операции сравнения всегда строка битов единичной длины со значением '1'B, если выражение типа сравнения истинно, или '0'B, если оно ложно.

В языке ПЛ/1 используются следующие операции сравнения.

$>$ больше;
$\geq$ больше или равно;
$=$ равно;
$\neg=$ не равно;
$\leq$ меньше или равно;
$<$ меньше;
$\neg>$ не больше;
$\neg<$ не меньше.

В зависимости от операндов, которые сравниваются, различают три типа сравнения:

— алгебраическое, означающее сравнение числовых величин в кодовой арифметической форме с учетом знаков. Цифровые знаковые данные преобразуются в десятичные данные с фиксированной или плавающей точкой, а данные типа строки битов — в двоичные данные с фиксированной точкой;

— символьное, означающее последовательное попарное сравнение слева направо символов в соответствии с упорядоченностью внутренних представлений символов в коде ДКОИ. Если операнды имеют различную длину, то более короткий дополняется справа пробелами;

— битовое, означающее сравнение двоичных цифр слева направо. Если длины операндов различны, то более короткий дополняется справа нулями.

Если в выражении типа сравнение участвуют операнды различных типов, то перед выполнением операции сравнения происходит преобразование типа. При этом учитывается приоритет типов данных. Данные с меньшим приоритетом преобразуются в данные с большим приоритетом. Приоритеты данных:

Арифметический тип	Высший приоритет
Цифровая строка знаков	
Строка символов	
Строка битов	Низший приоритет

Если в операции сравнения оба операнда являются цифровыми строками знаков, то перед выполнением операции происходит преобразование их в арифметический тип. Однако следует заметить, что строки символов нельзя сравнивать с арифметическими данными, т. е. преобразование строки символов в арифметическую форму не допускается.

Ввиду того, что значение результата выражения типа сравнения — строка битов, его можно использовать как двоичную константу при вычислении любых арифметических выражений.

Пример.

```
DCL X, Y, A, B;

X = (A < B) * Y;
```

Если значение результата при вычислении выражения (A < B) истинно (т. е. '1'B), то переменной X будет присвоено значение Y, в противном случае — значение нуль.

В языке ПЛ/1 допускаются логические (булевы) выражения, в которых могут использоваться следующие три логические операции:

— отрицание ¬;

— логическое умножение (конъюнкция) &;

— логическое сложение (дизъюнкция) |;

Остальные логические операции (тождество, импликация и др.) могут быть реализованы с помощью встроенной функции языка BOOL (описанной в § 7.3).

Операция отрицания выполняется всегда над одним операндом. Остальные операции являются бинарными, т. е. выполняются над двумя операндами.

Операндами логической операции могут быть любые проблемные данные, которые перед выполнением операции преобразуются в строки битов. Если при преобразовании операндов они получаются различной длины, то более короткий дополняется справа нулями. Логическая операция выполняется последовательно слева направо над каждой парой битов. Таким образом, результатом логической операции является строка битов длиной, равной длине большего операнда.

Для определения значений различных логических выражений необходимо уметь определять результат применения каждой логической операции к конкретным наборам операндов, участвующих в операциях.

Результатом операции отрицания является изменение значения битов операнда на противоположное:

A	¬ A
0	1
1	0

Логическое сложение (дизъюнкция):

A	B	A|B
0	0	0
0	1	1
1	0	1
1	1	1

Логическое умножение (конъюнкция):

A	B	A&B
0	0	0
0	1	0
1	0	0
1	1	1

Пример. Предположим, что мы имеем переменные типа строки битов:

```
DCL BA BIT (4),
    BB BIT (4),
    BC BIT (8);
```

и были заданы следующие операторы:

```
BA = '0011'B;  BB = '0001'B;
```

В результате выполнения операции

```
BC = BA & BB;
```

переменной BC присваивается значение '00010000'B, а в результате выполнения операции

```
BC = ¬ BA;
```

переменная BC оказывается равной '1100000'B. Заметим, что переменная BA есть четырехбитовая строка, которая расширяется до восьми битов путем добавления справа четырех нулей.

При выполнении операций логического сложения

BC = BA | BB;

переменной BC присваивается значение '00110000'B.

Обобщая все сказанное, можно определить при вычислении любых выражений следующий приоритет операций:

1) **, одноместный +, одноместный —, ¬
2) *, /
3) двухместный +, двухместный —
4) ||
5) >, ¬>, >=, ¬=, =, <=, ¬<, <
6) &
7) |

Пример.
При вычислении выражения

—4 * A * (B + C) ** 2 <= 0

сначала будут раскрыты скобки (B + C), затем проделана операция возведения в квадрат, выполнена операция одиночного минуса — 4, умножения его на A, затем на (B + C)** 2 и, наконец, операция сравнения <=. Так как последней выполняемой операцией является <=, данное выражение будет относиться к выражению типа сравнения.

2.6. Цифровые знаковые данные

Цифровые знаковые данные объявляются с помощью атрибута PICTURE (сокращенно PIC). Эти данные могут использоваться в программе только как переменные и объявляются следующим образом:

DECLARE идентификатор 'спецификация шаблона';

Под спецификацией шаблона понимается строка специальных символов, характеризующих значение переменной. Цифровые знаковые данные могут иметь две формы представления: цифровую и строки символов.

Описание данных с помощью шаблона расширяет возможности редактирования цифровых данных при выводе их на печать. Используя возможности отдельных символов шаблона, можно привести выводимые данные к виду, удобному для их чтения (например, подавить ведущие нули, ввести в значение данного пробелы или некоторые дополнительные символы и т. д.).

Вместе с тем возможно использование данных, описанных шаблоном, в арифметических операциях. Однако в этом случае приходится осуществлять их преобразование в цифровую форму. Хотя это преобразование осуществляется автоматически, оно требует

много машинного времени и является неэффективным, поэтому целесообразно избегать вычислений с участием данных этого типа. Удобнее осуществлять все вычислительные операции над переменными, объявленными в качестве цифровых, и лишь непосредственно перед выводом преобразовывать их в форму переменной, описанной шаблоном.

Все данные, описанные шаблоном, занимают в памяти 1 байт на символ (исключение составляют знаки V и K, которые места в памяти не занимают). Элементы арифметических данных, которые объявляются с атрибутом PICTURE, с точки зрения их внутреннего представления в памяти являются символьными строками. Их можно использовать в арифметических и логических операциях, в операциях сравнения, а также в операциях сцепления, причем в зависимости от выполняемой операции рассматривается арифметическое значение цифрового знакового данного или значение типа строки символов.

Арифметическое значение цифрового знакового данного может участвовать в арифметических и логических операциях, в операциях сравнения и в операциях присваивания переменным арифметического типа, а также в операциях ввода—вывода с элементами формата F и E. При этом происходит автоматическое преобразование переменной типа цифровой строки знаков из распакованного формата в упакованный десятичный формат или в десятичный формат с плавающей точкой.

При цифровой форме представления с помощью шаблона можно описывать десятичные данные с фиксированной и плавающей точкой.

При описании цифрового знакового данного в форме десятичного числа с фиксированной точкой в спецификации шаблона можно указывать не более 15 цифровых позиций. Максимальная длина спецификации шаблона, включая символ шаблона V, равна 32.

При описании цифрового знакового данного в форме десятичного числа с плавающей точкой спецификация шаблона должна иметь следующий формат:

$$\text{'мантисса} \left\{ \begin{matrix} E \\ K \end{matrix} \right\} \text{порядок'}$$

Мантисса может содержать не более 16 цифровых позиций, а порядок не более двух. Символ шаблона V может использоваться только в мантиссе. Рассмотрим функции отдельных символов шаблона, используемых при описании цифровых данных.

Для обозначения цифр, точки и порядка могут использоваться следующие символы:

9 — указывает, что в этой позиции элемента данных должна располагаться десятичная цифра;

V — указывает положение неявной (подразумеваемой) десятичной точки.

Неявная десятичная точка не занимает позицию во внутреннем представлении вычислительного данного (т. е. не запоминается машиной), однако ее положение учитывается при использовании данного в соответствующих преобразованиях;

E — указывает начало порядка для чисел с плавающей точкой; этот символ присутствует явно в цифровом значении знаковой строки;

K — указывает предполагаемое начало порядка, но не является символом в цифровом значении знаковой строки.

При описании данных с плавающей точкой последовательность символов шаблона должна представлять мантиссу и порядок переменной. Слева от символов E или K располагаются позиции, занимаемые мантиссой числового значения переменной, справа — порядком.

Обычно число с плавающей точкой допускает много вариантов представления в зависимости от выбираемого значения порядка. Однако в данном, описанном с помощью шаблона, оно нормализуется в том смысле, что первая значащая цифра мантиссы всегда занимает первую цифровую позицию в символах шаблона, относящихся к мантиссе.

Пример. Числа с плавающей и фиксированной точкой могут быть описаны следующим образом:

Значение данного	Спецификация шаблона	Результат редактирования
1567	'9999'	1567
12	'(6)9'	000012
22.785	'99V999'	22785
22.785	'99999'	00022
.84976E05	'V99999E99'	84976E05
007.77E04	'999V99E99'	77700E02
004.31E08	'99V9999K99'	4310007

Для того чтобы значения данных, выводимые на печать, были более читабельны, их необходимо отредактировать. С этой целью вводятся соответствующие знаки и символы. Они разделяются на замещающие и вставляемые символы.

К замещающим символам относятся:

Z — подавление ведущих нулей. В позициях данного, где помещаются символы Z, ведущие нули заменяются пробелами, чтобы полученные результаты были более удобными для чтения. Символы Z и 9 могут использоваться в шаблоне одновременно. Однако символ Z не должен стоять правее символа 9;

* — звездочка. В позициях данного, где стоят эти символы, ведущие нули заменяются звездочкой. Символы * и Z одновременно не могут появляться в спецификации шаблона;

++ ... — плюс плавающий. Если знак плюс появляется в спецификации шаблона более одного раза подряд, то он воспринимается транслятором как плавающий плюс. Подобно символу Z, он служит для подавления ведущих нулей. В позициях данного, где помещены плюсы, ведущие нули заменяются пробелами, а в последней из подавляемых позиций ставится знак плюс, если арифметическое значение данного больше 0, или пробел в противном случае;

— — ... — минус плавающий. Используется аналогично плавающему плюсу, но если арифметическое значение данного меньше 0, то в последней из подавляемых позиций ставится знак минус, в противном случае — пробел;

S — плавающий символ. Используется аналогично плавающему плюсу, но если арифметическое значение данного больше или равно нулю, то в последней из подавляемых позиций ставится знак плюс, в противном случае знак минус.

Для цифровых знаковых данных с фиксированной точкой в спецификации шаблона может содержаться только одна строка плавающих знаков. Для цифровых знаковых данных с плавающей точкой разрешается указывать строку плавающих знаков в части для мантиссы и в части для порядка.

Внутри последовательности знаков S, «+» или «—» недопустимо появление других символов, кроме V и вставляемых символов. В этом случае позиция, занятая вставляемым символом, может содержать:

— соответствующий символ, если слева от него находится значащая цифра;

— знак числа, если в первой справа позиции стоит первая значащая цифра;

— пробел, если первая значащая цифра находится справа дальше, чем в первой позиции.

Пример

Значение данного	Спецификация шаблона	Результат редактирования
153789	'ZZZ999'	153789
001200	'ZZZ999'	␣␣1200
0019	'***9'	**19
000.087	'***V***'	***087
004.2E03	'ZZZV9E99'	4200E00
00018	'SSSS9'	␣␣+18
—002.084	'----V999'	␣—2084
—003.02	'+++V99'	␣␣302

В качестве вставляемых символов используются:

. , B — символы, вставляемые для удобства чтения;

+ —S — знаки шаблона для знака числа;

T, L, R — знаки дополнительной пробивки.

Все вставляемые символы не влияют на числовое значение данного, но играют роль при выводе строки символов на печать. Они служат для лучшей формы представления чисел в выходных документах. Перед символами вставки должен стоять хотя бы один символ шаблона, представляющий цифровую позицию.

Рассмотрим функции вставляемых символов.

.— символ точка. Она используется для указания места вставки символа точки, т. е. в значение знаковой строки вставляется точка, если не встречается подавление нулей. При подавлении нулей точка ставится в следующих случаях:

— если слева перед позицией точки есть хотя бы одна значащая цифра;

— если слева перед точкой стоит символ V и дробная часть содержит по крайней мере одну цифру, отличную от нуля.

Этот символ шаблона не вызывает выравнивания элементов данных на десятичную точку. Он является частью знаковой строки, а не арифметического данного. Следует заметить, что в шаблоне данного может быть несколько символов точки и что присутствие этого символа ни в коей мере не влияет на место десятичной точки в значении переменной. Последнее определяется лишь символом V, если же его нет, то предполагается положение точки справа за последней цифровой позицией. По желанию программист может ввести символ «точка» в том месте, где символом V задано положение десятичной точки;

, — запятая ставится в тех же случаях, что и точка;

B — в значение знаковой строки в этой позиции вставляется пробел;

S — одиночный символ. В позиции данного, где стоит символ S, если арифметическое значение переменной больше или равно нулю, ставится знак плюс, в противном случае — минус;

+ — одиночный плюс указывает плюс, если арифметическое значение данного больше или равно 0, иначе — пробел;

— — одиночный минус указывает минус, если арифметическое значение данного меньше 0, иначе — пробел.

Символы (S, +, —), представляющие знак числа в шаблоне, могут находиться как левее, так и правее цифровых позиций по желанию программиста (см. пример на стр. 38).

В особых случаях оформления и выдачи знака числа используются некоторые специальные знаки — знаки дополнительных пробивок. Они применяются в том случае, если значение переменных вводится или выводится на перфокарты и знак числа указывается не как отдельный символ, а связан с последней цифрой числа.

Знаки дополнительных пробивок T, I, R в «спецификации шаблона» могут находиться только в одной последней цифровой по-

Пример

Значение данного	Спецификация шаблона	Результат редактирования
4550000	'9.999.999'	4.550.000
003.45	'zzzV.99'	⎵⎵3.45
003.45	'z99,V99'	⎵03,45
485	'9B9BB9'	4⎵8⎵⎵5
325	'999S'	325+
—58.22	'99.V9—'	58.2—
—13.375	'+99V.999'	⎵13.375

зиции, указывая, что в колонке с последней цифрой необходимо сделать дополнительную пробивку в 12-й строке для знака плюс и в 11-й для знака минус. Значение знаков дополнительных пробивок следующее:

T — этот знак, как и символ 9, представляет цифровую позицию. Если значение переменной положительно (или нуль), то в этой позиции формируется символ, изображение которого соответствует коду данной цифры совместно с пробивкой в 12-й строке, иначе формируется символ, соответствующий коду цифры и пробивки в 11-й строке (например, когда последняя цифра 5, то,если данное положительно, формируется символ 12—5, т. е. E).

I — указывает, что соответствующая позиция будет содержать символ, формируемый из цифры со знаком дополнительной пробивки в 12-й строке, если значение элемента данных больше или равно 0; в противном случае она содержит цифру;

R — указывает, что соответствующая позиция будет содержать символ, формируемый из цифры со знаком дополнительной пробивки в 11-й строке, если значение элемента данных меньше 0; в противном случае она содержит цифру.

Пример

Значение данного	Спецификация шаблона	Результат редактирования
+2034	'999T'	203D
—2034	'999T'	203M
+2034	'999I'	203D
—2034	'999I'	2034
+2034	'999R'	2034

В первой строке последняя цифра 4 и значение данного положительно; в спецификации шаблона правый символ Т. Следовательно, в результате редактирования сформируется символ, соответствующий пробивкам 12—4, а это символ D. Во второй строке формируется символ, соответствующий пробивкам 11—4, т. е. М.

Для сокращения записи спецификации шаблона в атрибуте PICTURE можно использовать коэффициент повторения. Коэффициент повторения представляет собой целое десятичное число, стоящее в круглых скобках, которое указывает, сколько раз повторяется следующий непосредственно за скобками символ шаблона.

Пример.

```
DCL A PIC '9999999.V99';
DCL A PIC '(7)9.V99';
```

Эти два оператора DECLARE эквивалентны.

Описание переменной шаблоном возможно и в том случае, если эта переменная строковая. Тогда спецификация шаблона должна состоять только из символов X. Их число указывает длину переменной, в соответствующих позициях которой размещаются любые символы. Максимальная длина переменной в этом случае не должна превышать 255.

Пример. Оператор

```
DECLARE A PICTURE 'XXXXXX';
```

полностью эквивалентен оператору

```
DECLARE A CHARACTER (6);
```

2.7. Данные управления программой

Данные управления программой используются для управления ходом выполнения программы. К ним относятся:

— данные типа метки;

— данные типа указателя.

Данные типа указателя будут рассмотрены в гл. 9. Остановимся более подробно на первом типе данных управления программой.

Элемент данных типа метки может быть переменной типа метки или константой типа метки.

Переменная типа метки должна быть явно объявлена с помощью атрибута LABEL. Она используется для управления ходом выполнения программы. Однако прежде чем выполнить оператор перехода, содержащий переменную типа метки, ей необходимо присвоить значение константы типа метки. С помощью переменной типа метки программист может организовывать многоуровневый разветвляющийся вычислительный процесс.

Пример.

```
DECLARE M11 LABEL;
```

Идентификатор M11 объявляется как переменная типа метки. Этой переменной в процессе выполнения программы присваивается значение константы типа метки. Следует заметить, что с данными типа метки невозможны другие операции, кроме операции присваивания.

Константа типа метки представляет собой идентификатор, стоящий перед оператором (или блоком) и отделенный от него двоеточием. С помощью этого идентификатора (метки) может быть осуществлена передача управления на помеченный оператор и тем самым изменен естественный ход выполнения программы.

Пример.

```
M1 : A = B + C;
```

В этом примере идентификатор M1 является меткой оператора и на эту метку может быть передано управление с помощью оператора перехода GOTO (см. § 3.2).

Глава 3

ОСНОВНЫЕ ОПЕРАТОРЫ ЯЗЫКА

3.1. Оператор присваивания

В процессе вычисления переменные приобретают конкретные значения. В языке ПЛ/1 факт придания некоторой переменной конкретного значения изображается специальной записью, которая называется оператором присваивания.

Оператор присваивания состоит из левой и правой частей, разделенных символом =. Левая часть — это обязательно идентификатор переменной. Правая часть — это некоторое выражение, значение которого нужно присваивать переменной, стоящей в левой части.

Если необходимо, значение «выражения» вычисляется, преобразуется и присваивается переменной, стоящей слева от знака присваивания, в форме, которая соответствует атрибутам этой переменной.

Оператор присваивания имеет вид

[метка:] ... идентификатор=выражение;

В отличие от обычных математических формул, отражающих статику вычислительного процесса, оператор присваивания показывает динамику вычислений. Дело в том, что символ присваивания имеет смысл «заменить на», поэтому в языке ПЛ/1 допустимы такие операторы присваивания, как M = M + 1. Эта конструкция означает: вычислить значение M + 1, используя текущее значение переменной M; затем текущее значение M заменить значением только что вычисленного выражения.

Наличие метки не обязательно, но она дает возможность обращаться к данному оператору по имени. Для обозначения одного и того же оператора может быть использовано несколько меток; они должны быть разделены двоеточием.

Например:

M1 : M10 : N = N + 1;

Пример

Обычная запись	Оператор присваивания
$M=\dfrac{a+bx}{c+dx^3}$	M = (A+B * X)/(C+D * X **3);
$F_y=x\dfrac{x^2-y^2}{x^2+y^2}$	FY = X * (X * X − Y * Y)/(X **2+Y **2);

3.2. Оператор перехода и пустой оператор

Оператор перехода используется в тех случаях, когда после некоторого оператора следует выполнить не следующий по порядку оператор, а какой-либо другой, расположенный в любом месте данного блока или некоторого активного блока.

Для того чтобы обеспечить переход к определенному оператору, его необходимо пометить меткой и в операторе перехода указать идентификатор этой метки.

Формат оператора перехода имеет вид

[метка:] GOTO { константа типа метка / переменная типа метка };

Сам оператор GOTO может быть помечен. Оператор в первом варианте обеспечивает безусловную передачу управления на оператор с указанной меткой, начиная с которого выполняется новая последовательность операторов.

Пример.

```
. . .
GOTO L1;
. . .
L1 : A = B + C/D;
```

При выполнении оператора перехода управление будет передано на оператор, помеченный меткой L1.

Если указан второй вариант (переменная типа метки), то оператор GOTO работает как переключатель. В этом случае значение переменной является меткой оператора, которому передается управление. Так как переменная может принимать различные значения при каждом выполнении данного оператора GOTO, то управление не всегда будет передаваться одному и тому же оператору.

Пустой оператор никаких действий не выполняет и не изменяет последовательность выполнения операторов. Понятие пустого оператора введено только для того, чтобы можно было ставить метку

в конце составного оператора или блока и таким образом переходить к концу его, пропуская какие-то другие операторы.

Общий вид пустого оператора

[метка:];

3.3. Условный оператор

Изменение естественного порядка выполнения операторов при записи разветвляющихся алгоритмов с помощью оператора перехода приводит к довольно громоздким конструкциям. Для упрощения записи программ в подобных случаях вводится условный оператор, с помощью которого можно обеспечить выполнение или невыполнение некоторого оператора (группы операторов или блока) в зависимости от заданных условий. Конструкция условного оператора имеет вид

IF выражение THEN оператор 1; [ELSE оператор 2;]

Условный оператор обеспечивает выполнение одного из операторов (оператора 1 или 2) в зависимости от значения выражения, стоящего за ключевым словом IF.

Если используется полная форма оператора (вариант с фразой ELSE) и значение выражения истинно, то выполняется оператор 1, в противном случае — оператор 2. Если используется неполная форма оператора (вариант без фразы ELSE) и значение выражения истинно, то выполняется оператор 1, в противном случае — оператор программы, непосредственно следующий за условным оператором.

В качестве выражения могут использоваться: выражение типа сравнения или скалярное выражение.

Если в условии дается скалярное выражение, то оно вычисляется и преобразуется в строку битов, длина которой зависит от значения выражения. Если значение какой-нибудь позиции этой строки битов равно 1, то выполняется оператор, следующий за ключевым словом THEN. Если значения всех позиций равны нулю, то в случае полного условного оператора выполняется оператор 2, а в случае неполного условного оператора — оператор, непосредственно следующий за условным оператором.

Условные операторы могут быть вложены один в другой, т. е. либо «оператор 1», либо «оператор 2», либо оба оператора могут быть условными. Уровень вложенности операторов не должен превышать 100. Кроме того, любой оператор, входящий в состав условного оператора, может быть помечен и ему может быть передано управление.

Примеры.

```
1)  IF X > Y THEN
      IF Z = W THEN
        IF W < P THEN Y = 1; ELSE P = Q;
      ELSE;
    ELSE X = 4;
M1: Z = 5;
```

В данном примере приведен вложенный условный оператор. При условии Z ≠ W использование пустого оператора (точка с запятой после ключевого слова ELSE) приведет к выходу из условного оператора, т. е. передаче управления на метку M1.

2) Требуется описать вычисление функции y по формуле

$$y = 24{,}2x - 1{,}5x^2 + 1{,}005x^3$$

при значении x от 1 до 9,9 включительно с шагом 0,1.

Этот алгоритм запишется на ПЛ/1 следующим образом:

```
    X = 1;
M:  Y = 24.2 * X - 1.5 * X ** 2 + 1.005 * X ** 3;
    X = X + 0.1;
    IF X < 10 THEN GOTO M;
```

В данном примере неполный условный оператор использован для организации циклического вычислительного процесса. В его условии указано выражение типа сравнения, которое обеспечивает окончание вычислений функций Y при значении X = 10, т. е. последним будет вычислено значение Y при X = 9,9.

3.4. Простейшие операторы ввода и вывода данных

Простейшими операторами ввода с перфокарт и вывода на печать являются операторы GET и PUT соответственно.

Оператор GET осуществляет ввод данных с внешнего носителя в память, отведенную для проблемных переменных, идентификаторы которых указаны в списке данных этого оператора. Формат оператора GET имеет вид

GET EDIT (список данных) (список форматов);

Вводимые данные при этом рассматриваются как непрерывная строка (поток) символов, причем границы отдельных перфокарт игнорируются.

Данные, значения которых вводятся, называются элементами данных оператора GET. Элементами данных в частном случае могут быть идентификаторы переменных, значения которых вводятся с перфокарт. При записи они отделяются друг от друга запятыми и образуют список данных этого оператора.

Форма ввода описывается с помощью списка форматов данных, в котором запятыми разделены перечисляемые элементы формата. С помощью списка элементов формата программист управляет преобразованием данных при вводе их с внешнего носителя (перфокарт).

При выполнении операторов ввода и вывода список данных и список форматов обрабатываются слева направо. При этом каждое значение элемента списка данных вводится (или выводится) согласно соответствующему элементу формата из списка форматов. Количество элементов формата данных может совпадать или не совпадать с количеством элементов в списке данных. Если элементов формата данных задано больше, чем необходимо для обработки списка данных, то лишние элементы формата игнорируются. Если элементов формата задано недостаточно, то список элементов формата исполь-

зуется повторно, начиная с первого элемента, и так будет повторяться до тех пор, пока не будет введено (или выведено) значение каждого данного из списка данных.

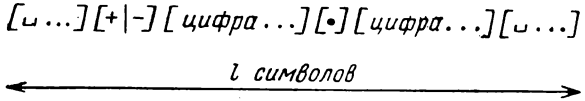
Рассмотрим использование оператора GET для чтения информации с перфокарт. При этом будем считать, что нам необходимо обрабатывать только десятичные числа с фиксированной точкой и строковые данные.

Для считывания десятичных чисел с фиксированной точкой используется элемент формата данного в следующей форме:

$$F\ (l,\ d),$$

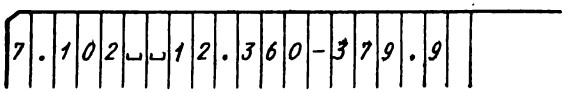
где F указывает, что l символов, находящихся на перфокарте, должны рассматриваться как десятичное число с фиксированной точкой; l — количество символов на внешнем носителе, которые должны быть взяты из потока; d указывает, что число обрабатывается так, как будто десятичная точка стоит перед последними d цифрами. Если же десятичная точка встретится в последовательности символов, взятых из потока, то ее положение отменяет параметр d.

Данные, которые читаются с элементом формата F, должны иметь следующее внешнее представление:



Если данные, прочитанные в соответствии с элементами формата F, имеют внешнее представление, отличное от указанного, то возникает прерывание выполнения программы. Если же последовательность символов содержит только пробелы, то соответствующему элементу списка данных присваивается нуль.

Пример. На перфокарте набита последовательность символов



С помощью оператора ввода GET переменным будут присвоены следующие значения:

```
GET EDIT (A, B, C, D, E, F) (F (2,0), F (3,2),
          F (6), F (2,2), F (4,3), F(2));
A = 7; B = 1.02; C = 12.3; D = 0.60; E = — 0.379; F = 0.9;
```

В качестве управляющего элемента формата при вводе данных используется элемент X, который позволяет пропускать один или несколько символов из вводимого потока. Он имеет формат

$$X\ (l),$$

где l указывает количество пропускаемых символов ($1 \leqslant l \leqslant 255$).

Пример. При вводе данных предыдущего примера с помощью оператора

GET EDIT (M, N, P) (F (3), X (5), F (4, 1), X (4), F (3, 1));

Переменным будут присвоены значения

M = 7.1; N = 2.36; P = 9.9;

При вводе строковых символьных данных используется элемент формата данных A, который имеет следующий вид:

A (l),

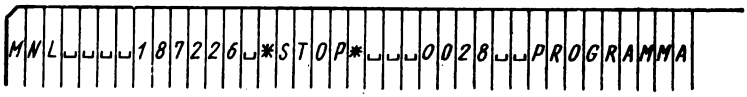
где l указывает количество символов, взятых из входного потока ($1 \leqslant l \leqslant 255$), которое может не совпадать с длиной переменной, объявленной в операторе DECLARE. Если l меньше длины объявленного данного, то оно дополняется справа пробелами, иначе последние «лишние» символы теряются.

Данные, находящиеся на внешнем носителе (перфокартах), представляются в форме строки символов, поэтому во время передачи их в буфер ввода, а затем в область памяти проблемных данных, объявленных в программе, оператор ввода GET производит двойное преобразование их: вначале в некоторую промежуточную форму (в соответствии со значением элемента формата), а затем во внутреннее представление данного, определяемого атрибутами, указанными в операторе DECLARE.

Пример. В программе объявлены переменные

```
DCL A FIXED (5,3),
    B CHAR (8),
    C FIXED (3,1),
    D CHAR (3);
```

На перфокарте набита последовательность символов



Рассмотрим, какие значения получают переменные A, B, C, D при выполнении следующего оператора ввода:

GET EDIT (A, B, C, D) (X (4), F (10, 4), X (1), A (5), F (9,3), A (9));

Действие оператора ввода будет осуществляться в следующем порядке:

— поток символов, расположенный на перфокарте, вводится в буфер ввода основной памяти;

— в буфере ввода производится первое преобразование в промежуточную форму:

а) согласно элементам списка форматов непрерывный поток символов разбивается на отдельные группы:

MNL␣	␣␣␣187226␣	*	STOP*	␣␣␣0028␣␣	PROGRAMMA
X (4)	F (10,4)	X (1)	A (5)	F (9,3)	A (9)

б) группы символов, определяемые управляющим форматом X (это MNL— и*), будут пропущены (игнорируются);

в) переменные A, B, C, D, определяемые в соответствии с элементами формата данных, получат в буфере ввода значения:

A = 18.7226; B = 'STOP*'; C = 0.028; D = 'PROGRAMMA';

— при передаче данных в область памяти производится вторичное преобразование (в соответствии с их атрибутами, указанными в операторе DECLARE), в результате которого они примут следующие значения:

A = 18.723; B = 'STOP* —— ';

C = 00.0; D = 'PRO';

Следует отметить, что переменная A согласно формату F (10,4) ввелась в буфер со значением 18.7226, которое в дальнейшем было усечено с округлением из-за объявленного атрибута переменной FIXED (5,3). То же произошло с переменной C.

Значение переменной B (введенное по формату A (5) как строка символов 'STOP*'), наоборот, было расширено с помощью пробелов до объявленной величины (атрибут переменной CHAR (8)). Усечение произошло при передаче значения переменной D.

Оператор вывода PUT подготавливает и выводит на печать значения данных, указанных в списке, в соответствии с элементами списка форматов. Он имеет следующий вид:

PUT EDIT (список данных) (список форматов);

Подготовка выводимого значения производится в буфере вывода, который имеется в распоряжении ДОС. Буфер вывода является прототипом строки печати и занимает область в 120 байтов.

При выводе десятичного числа с фиксированной точкой используется элемент формата

F (l, d),

где l — количество знаковых позиций буфера вывода, отводимых под поле выводимого значения; d — положение символа десятичной точки.

Значение параметра l выбирается, как правило, таким, чтобы обеспечить место как минимум для

— знака выводимого значения;

— целой части числа;

— десятичной точки;

— необходимого количества цифр дробной части.

Если значение этого параметра больше указанного минимума, то выводимое значение данного в буфере вывода слева будет заполняться пробелами.

При выводе на печать нескольких данных желательно, чтобы отдельные значения их в буфере вывода и на печати были отделены друг от друга пробелами. Для этого используется управляющий элемент формата

X (l),

где l — количество вставляемых пробелов.

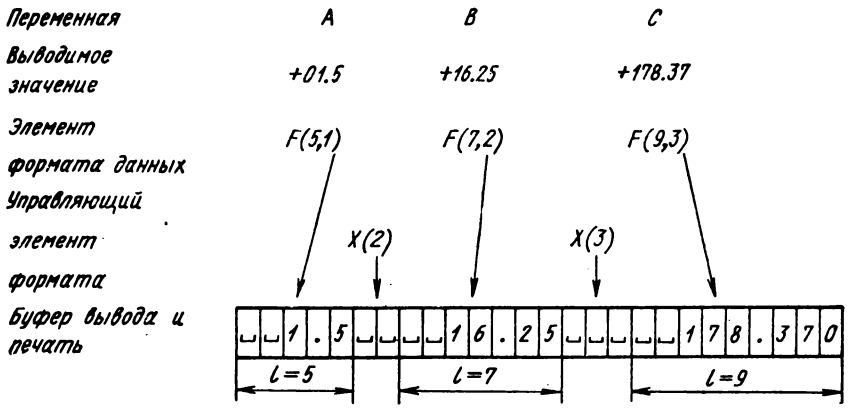
Кроме того, если мы хотим, чтобы буфер был выведен после того, как в нем будут подготовлены несколько данных (при этом не имеет значения, полностью ли он заполнен), то необходимо задать управляющий элемент формата SKIP, который означает пропуск строки и вывод на печать содержимого буфера. Для вывода содержимого буфера на одну и ту же строку можно использовать управляющий элемент SKIP в форме SKIP (0). Этот формат позволяет печатать одну строку символов на другую несколько раз, получая «жирную печать», или подчеркивать выдаваемый на печать текст.

Пример. Необходимо вывести на печать переменные A, B, C, которые могут принимать значения 10,0; 100,00; 9000,000. При этом между двумя первыми значениями необходимо вставить два пробела, а между вторыми — три.

Оператор PUT, который реализует эти требования, имеет вид

```
PUT EDIT (A, B, C) (SKIP, F (5,1), X (2), F (7,2), X (3), F (9,3));
```

Ниже показано заполнение буфера вывода и печать результатов.



При выводе значения, подготовленного элементом формата F, знак плюс не печатается, а заменяется пробелом. Ведущие нули также заменяются пробелами.

Чтобы во время выполнения программы напечатать некоторую последовательность символов (в частности, заголовок таблицы или документа), можно использовать элемент формата A. Он указывает, что последовательность символов, содержащаяся в списке элементов данных, должна быть полностью перенесена в область вывода. В этом случае используется оператор PUT в виде

```
PUT EDIT ('последовательность символов') (A);
```

Пример. Необходимо напечатать заголовок в следующей форме:

```
ТАБЛИЦА ЗНАЧЕНИЙ ТРИГОНОМЕТРИЧЕСКИХ
          Ф У Н К Ц И Й
СИНУС                          КОСИНУС
```

При выводе на печать нужно отступить от края листа на 10 позиций. Оператор вывода будет иметь вид

```
PUT EDIT ('ТАБЛИЦА ЗНАЧЕНИЙ ТРИГОНОМЕТРИЧЕСКИХ',
    'ФУНКЦИЙ', (13)'_', 'СИНУС',
    'КОСИНУС') (SKIP, X (10), A, SKIP, X (20), A,
    SKIP (0), X (20), A, SKIP, X (10), A, X (23), A);
```

3.5. Операторы начала и конца программы

Первым оператором каждой программы, написанной на языке ПЛ/1, должен быть оператора PROCEDURE (сокращенно PROC) с описателем OPTIONS (MAIN), указывающий, что выполнение программы должно начаться с первого выполняемого оператора этой процедуры. Таким образом, программа должна начинаться так:

имя процедуры: PROCEDURE OPTIONS (MAIN);

Этот оператор должен обязательно начинаться с метки, которая здесь обозначена как «имя процедуры». Для этой метки программист может выбрать идентификатор, состоящий максимум из 6 символов.

Конец программы должен быть указан оператором:

END; или END имя процедуры;

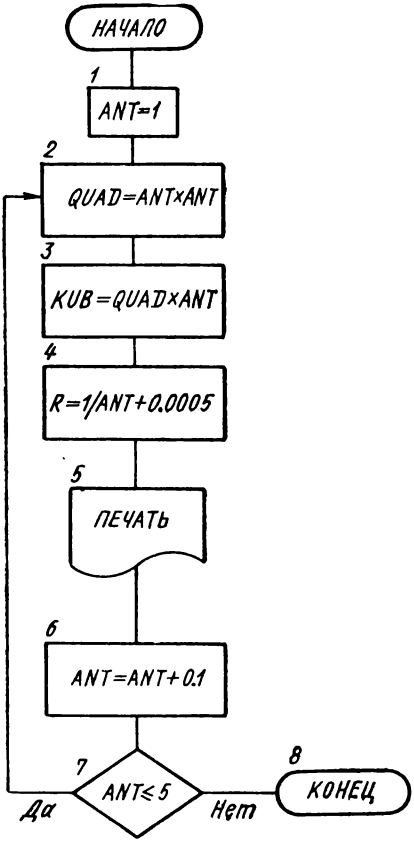
Он должен быть последним оператором программы, который сообщает транслятору, что в программе операторов больше нет.

В заключение следует отметить, что в данной главе рассмотрены основные операторы языка ПЛ/1, с помощью которых программист может описать любой вычислительный алгоритм.

Прмиер. Для последовательности чисел от 1 до 5 с шагом 0,1 вычислить значения второй и третьей степени и обратное значение и построить таблицу, в которой будут даны исходные значения и все соответствующие им вычисленные значения.

Для переменных примем следующие идентификаторы: начальное значение ANT, вторая степень QUAD, третья степень KUB, обратное значение R.

Схема задачи



Чтобы объявить выбранные переменные, необходимо решить, какое количество десятичных разрядов и общее количество разрядов необходимо иметь в памяти для значений каждой переменной. Выберем эти значения в следующих переделах:

Переменная	Максимальное значение	Требуется число дробных разрядов	p	q
ANT	5.0	1	2	1
QUAD	25.00	2	4	2
KUB	625.000	3	6	3
R	1.0000	4	5	4

Возьмем в качестве процедуры (имени точки входа в нашу программу) идентификатор TAB.

Тогда текст программы, написанной на языке PL/I, будет имет вид

```
TAB:PROC OPTIONS (MAIN);
     DCL  ANT FIXED (2,1),
          QUAD FIXED (4,2),
          KUB FIXED (6,3),
          R FIXED (5,4);
    PUT  EDIT ('ТАБЛИЦА 2-х и 3-х СТЕПЕНЕЙ',
          'И ОБРАТНЫХ ЗНАЧЕНИЙ ANT', 'ANT',
          'QUAD', 'KUB', 'R') (SKIP, X (10),
          2A, SKIP, X (10), A, 3 (X (15), A));
          ANT = 1;
     M1: QUAD = ANT * ANT ;
          KUB = QUAD * ANT;
          R = 1/ANT + 0.0005;
          PUT EDIT (ANT, QUAD, KUB, R) (SKIP, X (10),
                    F (4,1), X (15), F (6,2), X (15), F (8,3),
                    X (15), F (7,4));
        ANT = ANT + 0.1;
        IF ANT <= 5 THEN GOTO M1;
   END TAB;
```

3.6. Оператор DISPLAY

В некоторых случаях во время выполнения программы необходимо осуществить связь между оператором, который работает на ЭВМ, и программой. Такими случаями могут быть, например, выдача сообщений оператору о ходе выполнения программы; указания оператору, что́ ему необходимо сделать или какую величину ввести в программу в процессе ее выполнения. Для этого в языке имеется оператор DISPLAY, с помощью которого на пишущую машинку оператора может быть выдана требуемая информация. Необходимо отметить, что скорость вывода на пишущую машинку относительно невысока, поэтому вывод на нее должен ограничиваться короткими сообщениями.

Оператор DISPLAY имеет следующий формат:

DISPLAY (скалярное выражение)

[REPLY (переменная типа строка символов)];

При выполнении оператора вычисляется «скалярное выражение» и значение его преобразуется в строку символов (максимальная длина которой равна 72). Эта строка символов представляет собой сообщение, которое и выдается на пишущую машинку. В качестве этого выражения может использоваться строковая константа. Если используется первый вариант оператора (без режима REPLY), то после выдачи сообщения продолжается выполнение программы.

Если в операторе DISPLAY указан режим REPLY, то после выдачи сообщения на пишущую машинку в программу можно передать некоторую информацию. Если оператор используется с режимом REPLY, то процесс выполнения программы прерывается. Это состояние снимается сразу же после набора на пишущей машинке информации, передаваемой в программу.

Символы, заданные оператором с пишущей машинки, будут помещены в «переменную типа строка символов».

В программе можно, например, с помощью условного оператора сравнить значение, переданное с машинки, с некоторым заранее заданным значением, а результат этого сравнения использовать для организации разветвления в программе.

Примеры.

1) В программе производится проверка цифровой информации, вводимой с перфокарт. В случае обнаружения ошибки выдается сообщение с номером неправильной перфокарты.

Фрагмент программы:

```
        DCL A CHAR (80),
            B (80) CHAR (1) DEFINED A,
            I  PIC '99';
        I = 0;
M:      GET EDIT (A)(A(80));
            I = I + 1;
            DO J = 1 TO 80;
                IF B (J) < '0' THEN
                DISPLAY ('В ПЕРФОКАРТЕ НОМЕР —' || I ||
                ' ДОПУЩЕНА ОШИБКА');
                GOTO M;
                END;
```

Атрибут DEFINED позволяет наложить на переменную A массив B. Подробно его использование будет описано позже.

В операторе DISPLAY производится вычисление «выражения», заключающееся в выполнении цепочки операций сцепления.

2) В процессе решения задачи оператор должен указать порядок обработки некоторой группы операторов. Для реализации используется второй вариант оператора DISPLAY

```
        DCL Z CHAR (2);
M:  DISPLAY ('УКАЖИТЕ ПОРЯДОК ОБРАБОТКИ');
MM:   DISPLAY ('ДАЙТЕ ОТВЕТ МЕТКА M1 ИЛИ M2')
        REPLY (Z);
IF      Z = 'M1' THEN GOTO M1;
          ELSE IF Z = 'M2' THEN GOTO M2;
            ELSE GOTO M3;
M1:   ОПЕРАТОРЫ; ...
M2:   ОПЕРАТОРЫ;...
M3:   DISPLAY ('НЕПРАВИЛЬНЫЙ ОТВЕТ');
      GOTO MM;
```

Глава 4

АГРЕГАТЫ ДАННЫХ

До сих пор данные рассматривались как отдельные величины — константы или скалярные переменные. Однако переменные могут образовывать некоторые группы данных (агрегаты данных), объединенные единым математическим содержанием (например, векторы и матрицы) или связанные между собой по смыслу. В программе в некоторых случаях желательно иметь возможность ссылаться как на отдельные элементы таких групповых данных, так и на всю группу в целом.

В языке ПЛ/1 имеется два способа группирования данных:

— в форме массивов;

— в форме структур.

Оба способа сходны в том, что они позволяют объединять отдельные элементы данных в группы. Существенное отличие их состоит в следующем:

— элементы, объединенные в массивы, должны иметь одинаковые атрибуты;

— элементы, объединенные в структуры, могут иметь различные атрибуты.

4.1. Массивы

Массивом называется n-мерная упорядоченная совокупность элементов с одинаковыми свойствами (атрибутами).

Если элементы массива принадлежат к арифметическому типу, все они должны иметь одни и те же основания системы счисления, способы представления и разрядность или один и тот же шаблон. Если элементы массива являются строками битов или строками символов, все они должны иметь одну и ту же длину.

Для объявления массивов существует определенная форма оператора DECLARE, а именно:

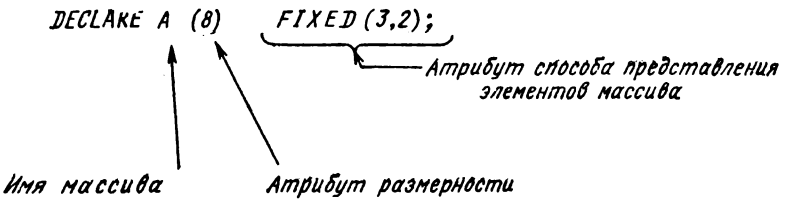
DECLARE идентификатор (атрибут размерности)
[атрибуты способа представления элементов];

Атрибут размерности указывает число измерений массива и границы по каждому измерению. В данном подмножестве языка могут использоваться одно-, двух- и трехмерные массивы. Границы по каждому измерению задаются с помощью положительных чисел, указывающих верхнюю границу измерения. Нижняя граница всегда подразумевается равной единице.

В качестве примера рассмотрим следующую таблицу:

Индекс I	1	2	3	4	5	6	7	8
A (I)	0,25	0,50	0,75	1,00	1,25	1,50	1,75	2,00

Такую таблицу можно включить в программу на ПЛ/1 как массив путем использования оператора DECLARE в такой форме:



Этим оператором DECLARE идентификатор A объявляется как имя одномерного массива с 8 элементами. Все элементы данных имеют одинаковый арифметический тип, что показывает атрибут FIXED (3,2).

Для обращения к отдельным элементам массива используется индексированная переменная; она представляет собой имя массива; за которым в круглых скобках указывается положение данного элемента внутри массива. Переменная с индексами для данного подмножества языка ПЛ/1 может иметь один, два или три индекса, т. е. определять положение элемента для одно-, двух- или трехмерного массива. Для обращения ко всему массиву используется неиндексированное имя этого массива.

Прежде чем смогут обрабатываться элементы массива, каждому из них должны быть присвоены какие-либо значения. Это может быть сделано несколькими операторами присваивания:

```
A(1) = 0.25;
A(2) = 0.50;
. . . . . .
A(8) = 2.00;
```

В качестве другого примера, рассмотрим следующую матрицу:

1	3	5	8
7	2	9	3
—8	5	—10	2

В программе эту матрицу можно объявить так:

```
DECLARE MATR (3, 4) FIXED (2);
```

При обращении к элементу массива MATR в индексе должно быть указано два числа, разделенных запятой. Первый индекс указывает номер строки (в нашем случае он меняется от 1 до 3), второй — номер столбца (от 1 до 4).

Благодаря атрибуту размерности массива транслятор получает информацию, необходимую для резервирования полей в памяти машины. Общее число элементов массива получается как произведение указанных в атрибуте чисел по всем размерностям. Если полученный результат умножить на число байтов памяти, выделяемое для каждого элемента массива, то можно получить общее количество байтов, выделяемое для хранения массива в памяти.

Заметим, что в основной памяти элементы массива запоминаются поочередно один за другим. В случае двух- и трехмерного массивов они идут один за другим таким образом, что сначала будет изменяться самый правый индекс. В нашем примере с матрицей это будет выглядеть так:

1	3	5	8	7	2	9	3	—8	5	—10	2
(1,1)	(1,2)	(1,3)	(1,4)	(2,1)	(2,2)	(2,3)	(2,4)	(3,1)	(3,2)	(3,3)	(3,4)

Индекс может задаваться целым числом или арифметическим выражением. Для того чтобы по этому индексу можно было найти соответствующий элемент массива, арифметическое выражение должно быть вычислено. Дробная часть результата отбрасывается, и полученное целочисленное значение используется в качестве индекса.

При использовании индексированных переменных программист должен следить, чтобы индексы не выходили за пределы границы измерения, которая указана в операторе DECLARE. Во время выполнения программы не происходит никакой проверки используемых индексов.

4.2. Выражения над массивами

Выражение над массивами представляет собой обобщение понятия скалярного выражения, которое содержит в качестве операнда массив в допустимой комбинации со скалярными.

В выражениях над массивами можно использовать те же операции, что и в скалярных выражениях. Таким образом, над массивами можно выполнять арифметические, логические операции, а также операции сравнения и сцепления.

Выражение над массивами дает в качестве результата массив значений: все операции над массивами выполняются поэлементно. Поэтому все массивы, входящие в одно выражение над массивами, должны иметь одинаковую размерность и одинаковые границы соответствующих измерений.

Если использовать описанный выше массив MATR, то выражение над ним может иметь такой вид:

2*MATR или MATR + 5

В первом выражении над массивом каждый элемент его удваивается, во втором — увеличивается на 5.

Если в указанном массиве все элементы должны быть равны 4, то это может быть выполнено при помощи 12 операторов присваивания:

```
MATR (1,1) = 4;
MATR (1,2) = 4;
. . . . . .
MATR (3,4) = 4;
```

Однако такой способ решения является слишком громоздким. В языке ПЛ/1 понятие оператора присваивания расширяется. Если слева от знака присваивания стоит идентификатор (имя) массива, то речь идет об операторе присваивания массиву. Для решения предыдущего примера достаточно написать следующий оператор:

```
MATR = 4;
```

Формат обобщенного оператора присваивания таков:

$$\text{массив} = \left\{\begin{array}{l}\text{выражение над массивами}\\ \text{скалярное выражение}\end{array}\right\};$$

Заметим еще раз, что операторы присваивания, в которых массив указывается и справа от знака присваивания, могут употребляться только для массивов с одинаковой размерностью и одинаковыми границами измерений.

Если справа от знака присваивания стоит скалярное выражение, то его значение присваивается всем элементам массива.

Примеры:

1) Объявлены массивы:

```
DCL A (2,3) FIXED (2),
    B (2,3) FIXED (2),
    C (2,3) FIXED (2);
```

Пусть массивы A и B содержат следующие значения элементов:

$$A = \begin{vmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{vmatrix}; \qquad B = \begin{vmatrix} 8 & 4 & 2 \\ 3 & 1 & 0 \end{vmatrix}.$$

Тогда после выполнения операторов

```
C = 3*A;
```

массив C будет содержать

$$C = \begin{vmatrix} 3 & 9 & 15 \\ 6 & 12 & 18 \end{vmatrix},$$

```
C = A**2 - B;
```

$$C = \begin{vmatrix} -7 & 5 & 23 \\ 1 & 15 & 36 \end{vmatrix}.$$

2) Имеется массив десятичных чисел с фиксированной точкой из ста элементов, подготовленный на перфокартах. Необходимо найти минимальный элемент массива и вывести его на печать. Программа этой задачи может быть следующей:

```
MINI:    PROC OPTIONS (MAIN);
         DCL A (100) FIXED (4,2),
             MIN FIXED (4,2);
          GET EDIT (A) (F (4,2));
          MIN = A (1); I = 2;
M:     IF A (I) < MIN THEN MIN = A (I);
       I = I + 1;
       IF I< = 100 THEN GOTO M;
       PUT EDIT (MIN) (SKIP, X (30), F (6,2));
END MINI;
```

4.3. Структуры

Существуют такие задачи, в которых желательно логически связать данные различного типа и назвать их одним именем. Для этого в языке ПЛ/1 используются так называемые структуры.

Структура есть иерархически упорядоченная совокупность скалярных переменных, массивов и структур. Элементы структуры могут не принадлежать к одному и тому же типу данных.

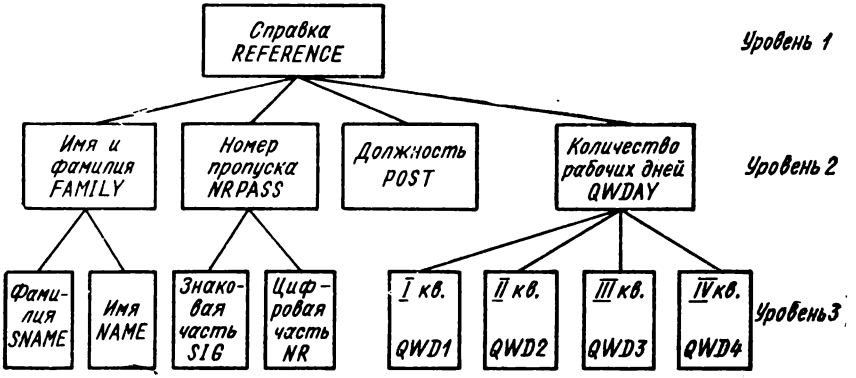
Структуры, в свою очередь, могут содержать в себе другие структуры. Самая внешняя структура называется старшей, а содержащиеся в ней структуры — младшими. Старшая структура должна быть на первом уровне, младшие должны всегда иметь номер уровня, численно больший номера уровня той структуры, в которой они содержатся. Как правило, структуры используются в языке для описания и обработки любых документов, элементы которых связаны общим названием (именем) всего документа и отдельных его граф. Рассмотрим общий принцип построения структуры на следующем примере.

Пример. В организации имеется документ, в котором указываются некоторые сведения о работе сотрудников (количество отработанных дней в квартале). Он имеет следующую форму:

Справка о работе сотрудников

Сотрудник		Табельный номер		Должность	Количество рабочих дней			
Фамилия	Имя	Знаковая часть	Цифровая часть		I кв.	II кв.	III кв.	IV кв.
CHAR(20)	CHAR(10)	CHAR(3)	PICTURE'(7) 9'	CHAR(22)	FIXED(3)	FIXED(3)	FIXED(3)	FIXED(3)

Этот документ можно описать следующей блок-схемой:



Для группы данных дается главное имя REFERENCE. Ему подчиняются другие имена: FAMILY, NRPASS, POST и QWDAY. Этим именам могут подчиняться другие имена: SNAME, NAME, SIG, NR, QWD1, QWD2, QWD3, QWD4. На последнем уровне подчинения находится совокупность элементов данных, каждая из которых является отдельным элементом данных или массивом.

Учитывая заданные характеристики элементов, рассмотренную структуру можно объявить с помощью оператора:

```
DCL  1 REFERENCE,
       2 FAMILY,
           3 SNAME CHAR (20),
           3 NAME CHAR (10),
         2 NRPASS,
           3 SIG CHAR (3),
           3 NR PIC'(7) 9',
         2 POST CHAR (22),
```

```
  2 QWDAY,
  3 QWD1 FIXED (3),
  3 QWD2 FIXED (3),
  3 QWD3 FIXED (3),
  3 QWD4 FIXED (3);
```

Следует отметить, что при записи оператора DECLARE для структуры имя старшей структуры указывается на первом месте, за ним следуют подструктуры и имена элементов структуры с атрибутами.

Оператор DECLARE для структур имеет следующий общий формат:

```
DECLARE 1 идентификатор [атрибут распределения памяти]
          {, положительное число идентификатор
            [атрибут размерности]  [атрибуты данных]} ...;
```

где «идентификатор», следующий за номером уровня 1, является именем старшей структуры;

«Атрибут распределения памяти» — ALIGNED или UNALIGNED определяет способ распределения в памяти элементов структуры. Эти атрибуты могут быть указаны только для имен старших структур. Они позволяют либо ускорить обращение к отдельным элементам структуры, либо сэкономить память машины для их размещения. Способ их использования будет рассмотрен позже; «положительное число» — это номер уровня, который может меняться в пределах от 2 до 255;

«идентификатор» — имя подструктуры или элемента структуры. Если он — имя подструктуры, то за ним следует запятая, а за запятой — следующая подструктура или элемент структуры, который содержится в этой подструктуре. Если он — имя элемента структуры, за ним может следовать атрибут размерности и (или) атрибуты данного. Если после «идентификатора» элемента структуры отсутствуют атрибуты данных, то присваиваются атрибуты по умолчанию, подобно скалярным переменным.

Максимальная глубина иерархического строения структур — 8. Если употребляются элементы структуры с одинаковыми атрибутами данных, то при обьявлении можно не писать эти атрибуты несколько раз, а использовать возможность объединения имен элементов структур с одинаковыми атрибутами в группу, заключенную в скобки.

В предыдущем примере в такую группу можно было объединить

```
QWD1, ..., QWD4.
  2 QWDAY,
    3 (QWD1, QWD2, QWD3, QWD4) FIXED (3);
```

Всем идентификаторам в скобках ставятся в соответствие атрибуты, следующие за списком. Номер уровня, который стоит перед скобками, действует для всех идентификаторов, стоящих в списке.

Объединение в группу возможно не только для элементов структур, но и для подструктур.

Пример.

```
DCL  1 COEF,
       2 POL1,
         3 A0,
         3 A1,
         3 A2,
       2 POL2,
         3 B0,
         3 B1;
```

Можно объявить и так:

```
DCL  1 COEF,
       2 (POL1,
          3 (A0, A1, A2),
            POL2,
          3 (B0, B1));
```

Для наглядности номера уровней в структуре записываются друг под другом. Но можно структуру COEF записать следующим образом:

```
DCL 1 COEF, 2 (POL1, 3 (A0, A1, A2), POL2, 3 (B0, B1));
```

Атрибут размерности может задаваться вместе с атрибутами данных только для младших элементов структуры. Старшая структура и подструктура не могут описываться с атрибутом размерности, следовательно, массивы структур не могут использоваться в программах.

Пример. Объявлена следующая структура:

```
DCL 1 HSTR,
      2 USTR1,
        3 FA (4,3) DEC (3),
        3 FB (4) CHAR (8),
      2 USTR2,
        3 MARKE (4) LABEL,
        3 NR PIC '99V.999';
```

Старшая структура HSTR содержит две подструктуры USTR1 и USTR2. Подструктура USTR1 содержит двухмерный массив FA десятичных чисел с плавающей запятой и одномерный символьный массив FB. Подструктура USTR2 содержит одномерный массив меток MARKE и элемент структуры NR, являющийся цифровой знаковой переменной. Массивы FA, FB и MARKE запоминаются в памяти друг за другом таким же способом, как и в случае массивов, которые не являются элементами структур.

Часто бывает необходимо в программе обратиться к подструктуре или элементу структуры. Это можно достичь, используя имя подструктуры или элемента структуры.

Пример. Идентификатор STR оператором DECLARE объявляется как имя структуры:

```
DCL 1 STR,
      2 U1,
        3 VAR DEC (4),
        3 FD (3,2) FIXED (3),
      2 U2 PIC 'S99.V9';
```

Если в каком-либо операторе программы встречается имя STR, то происходит обращение ко всей структуре. По имени подструктуры U1 обращаются к семи элементам: VAR, FD (1,1), FD (1,2), FD (2,1), FD (2,2), FD (3,1), FD (3,2). По имени элемента структуры FD обращаются к шести элементам: FD (1,1), FD (1,2) ..., FD (3,2).

Если в двух структурах встречаются элементы с одинаковыми именами или скалярная переменная имеет то же самое имя, что и элемент структуры, то необходимо как-то избежать двусмысленности. В языке ПЛ/1 это достигается использованием уточненного имени.

Уточненное имя состоит из последовательности имен, разделенных точками, в которой имена следуют в порядке возрастания номеров уровней.

В уточненном имени необязательно указывать имена всех более высоких уровней, содержащих данное имя, хотя это и не запрещается. Необходимо указать столько имен, чтобы сделать нужное имя уникальным. Уточненное имя может быть индексированным, если этим именем обращаются к элементу массива внутри структуры.

Примеры.

```
1) DCL 1 STR,
         2 D,
           3 E FLOAT (3),
           3 F FIXED,
         2 G,
           3 E (4) CHAR (8),
           3 H FLOAT (7);
```

В объявленной структуре имеются два элемента, обозначенные одинаковыми именами E. При обращении к скалярной переменной уточненное имя будет иметь значение STR.D.E, а при обращении ко всему массиву — STR.G.E. Правда, в этом случае имена элементов структуры будут уникальными и без указания имени старшей структуры, т. е. D.E и G.E. При обращении ко второму элементу массива E уточненное имя будет иметь значение G.E (2).

4.4. Выражения над структурами

Если в программе объявлены структуры, часто бывает необходимо производить операции с элементами структур, подструктурами и целыми структурами.

Под выражением над структурами понимается выражение, которое содержит в качестве хотя бы одного операнда структуру. Этим операндом может быть как имя старшей структуры, так и имя подструктуры. Результатом вычисления выражения над структурами является структура. В качестве операндов в выражении над структурами могут также использоваться константы и скалярные переменные.

В выражениях над структурами могут употребляться все операции, которые разрешаются в скалярных выражениях. Результатом выражения над структурами является структура.

Все операции, выполняемые над структурами так же, как и над массивами, выполняются поэлементно. Это значит, что вычисление выражения производится для каждого отдельного элемента структуры. Следовательно, все структуры, входящие в некоторое выражение над структурами, должны иметь одинаковое строение. Это означает, что:

— структуры должны иметь одно и то же число содержащихся в них скаляров и массивов;

— расположение скаляров и массивов внутри структур должно быть одинаковым;

— одинаково расположенные массивы должны иметь одинаковые размерности и границы.

Однако типы данных в элементах структур могут быть различны, если можно сделать соответствующее преобразование.

Пример. Имеются две структуры:

```
DCL 1 A,
      2 PAZ1,
        3 POD1,
        3 POD2,
        3 POD3,
      2 PAZ2,
        3 POD4,
        3 BETA,
        3 POD5 (2),
    1 B,
      2 PAZ1,
        3 POD1,
        3 ALPHA,
        3 POD2,
      2 PAZ2,
        3 ALPHA,
        3 POD4,
        3 POD5 (2);
```

Выражение

```
A — 2 * B
```

является сокращенной записью следующих выражений (т. е. выполняется следующим образом):

```
A.PAZ1.POD1 — 2 * B.PAZ1.POD1
A.PAZ1.POD2 — 2 * PAZ1.ALPHA
POD3 — 2 * B.PAZ1.POD2
A.PAZ2.POD4 — 2 * PAZ2.ALPHA
BETA — 2 * B.PAZ2.POD4
A.PAZ2.POD5 — 2 * B.PAZ2.POD5
```

Заметим, что последнее выражение является выражением над массивами, т. е. будет состоять из следующих скалярных выражений:

```
A.PAZ2.POD5 (1) — 2 * B.PAZ2.POD5 (1)
A.PAZ2 POD5 (2) — 2 * B.PAZ2.POD5 (2)
```

Общая форма оператора присваивания структуре такова:

$$\text{структура} = \left\{ \begin{array}{l} \text{выражение над структурами} \\ \text{скалярное выражение} \end{array} \right\};$$

В выполнении оператора присваивания структуре есть некоторое сходство с оператором присваивания массиву. Присваивание структуре производится поэлементно, а именно в таком порядке, в каком элементы описываются при объявлении структуры и в каком они располагаются в памяти машины.

Пример. Имеются две структуры

```
DCL 1 S,
      2 B FIXED,
      2 C,
      2 D (3),
    1 P,
      2 L,
      2 Z,
      2 E (3);
```

Оператор присваивания

```
S = P;
```

представляет следующие операторы присваивания для элементов структуры:
B = L; C = Z; D (1) = E (1); D (2) = E (2); D (3) = E (3);

Следует напомнить, что при выполнении оператора B = L; произойдет преобразование значения переменной L (имеющей по умолчанию атрибуты FIXED BIN (15)) в десятичное целое число (согласно объявлению переменной B с атрибутами FIXED DEC (5))

Заметим, что если при вычислении выражений над структурами используется элемент структуры, то значение его может измениться при выполнении оператора присваивания. В дальнейшем в выражениях, написанных после этого оператора присваивания, будет использоваться новое значение элемента.

Пример.

```
DCL 1 STR,
      2 A,
      2 B,
        3 C1,
        3 C2;
```

В программе встречается оператор присваивания

```
STR = STR + STR.A;
```

Он эквивалентен следующим двум операторам:

```
STR.A = STR.A + STR.A;
STR.B = STR.B + STR.A;
```

Последний оператор представляет собой такие операторы:

```
STR.C1 = STR.C1 + STR.A;
STR.C2 = STR.C2 + STR.A;
```

В двух последних операторах используется измененный в первом операторе присваивания элемент структуры STR.A

Во втором варианте оператора присваивания структур

структура = скалярное выражение;

каждому элементу структуры присваивается результат вычисления скалярного выражения.

4.5. Специальные атрибуты для описания данных

Идентификатор, встретившийся в программе, написанной на языке ПЛ/1, может представлять собой одну из переменных, обрабатываемых в этой программе. Он может, например, представлять данное, относящееся к переменным с десятичным основанием системы счисления в форме с фиксированной точкой, или переменную типа строки символов.

Свойства переменной, описанной данным идентификатором, как уже известно, определяются в языке ПЛ/1 с помощью атрибутов. В гл. 2 были рассмотрены атрибуты, определяющие основные характеристики различных переменных. Кроме них могут использоваться специальные атрибуты, которые позволяют повысить эффективность применения данных и сократить текст описываемой программы. Эти атрибуты не относятся к определенному типу данных, а могут применяться к любому рассмотренному нами данному. К этим атрибутам относятся: INITIAL (сокращенно INIT), DEFINED (сокращенно DEF), ALIGNED и UNALIGNED.

Атрибут INITIAL указывает значения констант, которые будут присвоены данным в момент выделения для них области памяти.

Формат атрибута:

INITIAL (элемент [,элемент] ...)

Присвоение начального значения переменной обеспечивает ввод этого значения, но не требует объема памяти для размещения каких-либо команд в объектном модуле. Следовательно, использование этого атрибута позволяет сэкономить некоторый объем основной памяти.

Атрибут INITIAL может быть указан как для элементов данных, так и для массивов. В объявлении структуры атрибут может использоваться только на уровне элементов структуры.

Не разрешается задавать атрибут INITIAL для переменных, объявленных с атрибутами DEFINED, старших структур, параметров процедур, имен входов и имен файлов.

В качестве элементов данного атрибута могут использоваться: арифметические константы, константы типа строки символов или строки битов. Для скалярной переменной может быть указана только одна константа; для массива может быть задано несколько значений констант. При этом значения констант, указанные для массива,

присваиваются последовательно элементам массива с учетом упорядоченности по строкам (последний индекс меняется наиболее быстро).

Если для массива указано большее число констант, чем элементов массива, то лишние константы игнорируются, а если их указано меньше, то оставшимся элементам массива начальные значения не присваиваются.

Пример.

```
DCL A FIXED (3,1) INITIAL (12.1),
    B CHAR (12) INIT ('РЕЗУЛЬТАТ'),
    C (3,2) BIT (4) INIT ('0110'B,'1100'B,'0011'B,'01'B ' 1'B, '0001'B);
```

В данном примере, когда отводится память для переменной B, ей присваивается значение строки символов «результат», дополненное справа пробелами до 12 символов. Дополнение двоичными нулями также производится при выделении памяти и присвоении начальных значений элементам массива C. При этом присвоение этих значений будет производиться в следующем порядке:

```
C (1,1) = '0110'B; C (1,2) = '1100'B; C (2,1) = '0011'B;
C (2,2) = '0100'B; C (3,1) = '1000'B; C (3,2) = '0001'B;
```

Переменным, для которых память выделяется при загрузке программы, т. е. переменным с классом памяти STATIC (см. § 6.6), значение, указанное в атрибуте INIT, присваивается только один раз. Переменным с классом памяти AUTOMATIC, память для которых отводится каждый раз при активизации содержащего их блока, указанные в атрибуте значения присваиваются каждый раз при выделении памяти.

В больших массивах часто приходится встречаться с тем, что нескольким элементам массива должно быть задано одно и то же значение. Для сокращения записи можно использовать коэффициент повторения. Коэффициент повторения — это целое десятичное число, стоящее в круглых скобках перед значением константы.

Пример

```
DCL A (8,7) FIXED (2) INIT ((6)0, (25)12, (5)0);
```

В этом случае первым шести элементам будет присвоено значение нуль, следующим 25 элементам — значение 12, следующим пяти — нуль, а последние 10 элементов массива начального значения не получат

Если элементы массива являются переменными типа строки (символов или битов), то во избежание путаницы коэффициент повторения начальных значений необходимо записывать первым, а коэффициент повторения символов строки — вторым и задавать его даже тогда, когда он равен единице. Если начальному строчному значению предшествует только одно взятое в скобки десятичное число, то оно интерпретируется как коэффициент повторения символов строки.

Пример.

```
DCL CH (5) CHAR (5) INIT ((5)'G'),
    CJ (4) CHAR (8) INIT ((4) (1) ABCDEFGH');
```

В данном случае элементу массива CH (1) присваивается начальное значение 'GGGGG', а остальным элементам начальные значения не присваиваются. С другой стороны, каждому элементу массива CJ присваивается значение в виде строки символов 'ABCDEFGH'.

Атрибут DEFINED. При выполнении рабочей программы отдельным переменным, а также массивам и структурам отводится место в основной памяти машины. В процессе выполнения программы к этим переменным можно обращаться с помощью соответствующих идентификаторов. Однако в языке ПЛ/1 существует возможность двум или нескольким переменным выделять одно и то же место в памяти, что позволяет более эффективно и экономно использовать основную память машины. Доступ к такой области памяти в этом случае возможен как с помощью идентификатора одной, так и остальных переменных. Такая возможность обеспечивается с помощью атрибута DEFINED, общая форма которого имеет вид

DEFINED идентификатор базы

Идентификатор, который имеет атрибут DEFINED, называется определяемым идентификатором. Идентификатор, область памяти которого используется определяемым идентификатором, называется идентификатором базы.

Пример

```
DCL A FLOAT,
    B FIXED DEFINED A;
```

В данном случае идентификатор переменной A является идентификатором базы, а B – определяемым идентификатором. Для переменной B область памяти в которой будет находиться ее значение, не выделяется. Обе переменные на основании данного объявления используют одну и ту же область памяти. Если в операторе программы встретится переменная A это значит, что содержимое области памяти будет рассматриваться как представление десятичного числа с плавающей точкой. Если в операторе дана переменная B, то содержимое этой же области памяти рассматривается как десятичное число с фиксированной точкой.

Существуют два способа использования этого атрибута:
— определение соответствием;
— определение перекрытием.

Определение соответствием реализуется в том случае, когда определяемый идентификатор является массивом с одинаковой размерностью и границами идентификатора базы или имеет тождественную структуру с идентификатором базы. При этом обращение к элементам определяемой переменной интерпретируется как обращение к соответствующему элементу базового идентификатора.

Пример.

```
DCL A (10,10) FIXED (3,2),
    B (10,10) FIXED (3,2) DEF A;
```

Определяемый идентификатор B имеет такую же размерность и границы, что и идентификатор базы A. В программе при обращении к элементу массива B осуществляется обращение к соответствующему элементу массива A. В свою очередь, изменение значения элемента массива A одновременно вызовет изменение значения соответствующего элемента массива B.

Определение перекрытием означает, что определяемая переменная будет занимать часть или всю память, отведенную для базового идентификатора. Длина определяемой переменной не должна быть больше длины идентификатора базы.

Определение перекрытием разрешено для значений переменных, представленных в табл. 4.1.

Таблица 4.1

Определяемая переменная	Базовый идентификатор
Скалярная арифметическая переменная в кодовой форме	Неиндексированная скалярная арифметическая переменная в кодовой форме с тем же основанием, способом представления и разрядностью
Скалярная переменная типа метки	Неиндексированная скалярная переменная типа метки
Скалярная переменная типа указателя	Неиндексированная скалярная переменная типа указателя
Строка символов	Строка символов
Цифровая знаковая переменная	Цифровая знаковая переменная
Массивы с элементами типа цифровой знаковой переменной или строки символов	Массивы с элементами типа цифровой знаковой переменной или строки символов
Структуры с элементами типа строки символов, цифровых знаковых данных или массивов описанного выше типа	Структуры, элементами которых являются строки символов, цифровые знаковые данные и массивы описанного выше типа
Структура	Структура с идентичным строением с элементами типа арифметических данных, данных типа метки, указателя и строки символов

Примеры.

1) Строка символов L длиной 80 байтов используется в программе как единое целое, но иногда необходимо осуществить доступ к каждому отдельному символу этой строки. Это требование может быть удовлетворено с помощью определения наложением, а именно:

```
DCL L CHAR (80),
    L1 (80) CHAR (1) DEF L;
```

Ввиду того, что массив L1 накладывается на переменную L, обращение к любому элементу массива по существу является обращением к соответствующему символу строки L.

Если в программе последовательности символов 1—6 и 24—31 строки должны обрабатываться как отдельные строки, то это можно осуществить

с помощью следующего объявления:

```
DCL L CHAR (80),
    1 ST DEF L,
      2 A1 CHAR (6),
      2 EMPTY CHAR (17),
      2 A2 CHAR (8);
```

В этом случае при объявлении структуры необходимо учесть промежуток (длиной 17 символов) между указанными строками. Это реализуется с помощью переменной EMPTY.

2) В одном из блоков программы последовательно обрабатываются три больших массива данных. Для более экономичного использования основной памяти с целью их размещения целесообразно использовать определение перекрытием. В качестве базовой переменной необходимо выбрать массив, имеющий наибольшие размеры, а остальные определить с помощью атрибута DEFINED. Тогда при обработке каждого массива они будут занимать в памяти одно и то же место. Оператор объявления в этом случае имеет вид

```
DCL A (5, 8, 20),
    B (10, 60) FIXED (5,2) DEF A,
    C (100) CHAR (25) DEF A;
```

Для массива A, элементы которого по умолчанию рассматриваются как десятичные числа с плавающей точкой, будет отведена область памяти $4 \times 800 = 3200$ байтов, для массива B необходимо $3 \times 600 = 1800$ байтов, а для массива C — $25 \times 100 = 2500$ байтов. Отсюда видно, что оба массива (B и C) будут частично перекрывать область памяти, выделенную для массива A.

Атрибуты ALIGNED и UNALIGNED могут быть указаны для элементов данных, массивов и старших структур. Эти атрибуты определяют способ расположения в памяти элементов данных.

Если задается атрибут ALIGNED, транслятор так распределяет память, что элементы массива (структуры) выравниваются на границу памяти соответственно требованию для конкретного типа данных (описание границ выравнивания рассмотрено в § 4.6).

Если атрибут ALIGNED (или UNALIGNED) указан для имени старшей структуры, то действие его относится ко всем элементам структуры, кроме тех, для которых он объявлен явно. Структуры, содержащие элементы типа строки битов, должны иметь атрибут ALIGNED или этот элемент должен быть объявлен явно с атрибутами ALIGNED.

Атрибут UNALIGNED указывает, что каждый размещаемый в памяти элемент массива (структуры) соприкасается с элементом, предшествующим ему. Таким образом, при использовании этого атрибута сокращается объем основной памяти для размещения переменных. Однако при этом увеличивается время обращения к ним.

При определении перекрытием для данных типа строки символов или цифровых знаковых данных атрибут UNALIGNED должен быть объявлен как для определяемого идентификатора, так и для идентификатора базы.

Атрибуты по умолчанию присваиваются на уровне элементов данных. Для данных типа строки символов и цифровых знаковых

данных по умолчанию присваивается атрибут UNALIGNED; для всех остальных типов данных — атрибут ALIGNED. Для структуры по умолчанию определяется атрибут UNALIGNED, а для массива ALIGNED.

В рассматриваемой реализации языка использование этих атрибутов в операторе DECLARE не влияет на расположение данных в основной памяти. Они используются только по умолчанию. Однако использование атрибута ALIGNED в некоторых случаях обязательно, иначе будет выдаваться сообщение об ошибке при трансляции программы.

4.6. Память, требуемая для размещения различных данных

Как уже известно, переменные, используемые в программе, требуют определенного места в памяти для их размещения. Размеры памяти для каждой переменной определяются программистом с помощью атрибутов. Однако при выделении памяти каждая переменная выравнивается на определенную границу (табл. 4.2).

Таблица 4.2

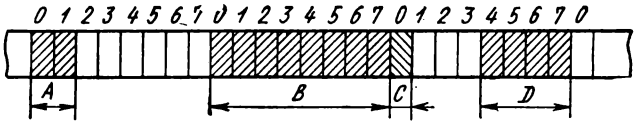
Атрибуты данных	Требуемая память	Граница выравнивания
FIXED BIN	4 байта	Слово
FIXED DEC	1/2 байта на цифру +1/2 байта на знак	Байт
FLOAT BIN	4 или 8 байтов	Слово или двойное слово
FLOAT DEC	4 или 8 байтов	То же
PICTURE	1 байт на один символ шаблона (кроме V, K, I, R)	Байт
CHARACTER	1 байт на символ	»
BIT	1 байт на 8 битов	»
Переменная типа метки	8 байтов	Слово
Переменная типа указателя	4 байта	»

Если бы память для переменных отводилась в соответствии с порядком, в котором они объявлены в программе, то из-за необходимости выравнивания их на определенную границу могли оказаться неиспользованными значительные участки основной памяти.

Пример. Объявлены следующие переменные:

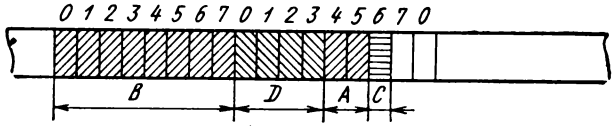
```
DCL A FIXED (3,1),
    B FLOAT (9),
    C CHAR (1),
    D FIXED BIN (10);
```

Учитывая границы выравнивания: для A — байт, для B — двойное слово, для C — байт и для D — слово, распределение памяти для этих переменных, соответствующее порядку объявления, будет иметь вид



Таким образом, объявленные переменные с учетом выравнивания займут объем памяти 24 байта. При этом окажутся неиспользованными участки памяти общим объемом 9 байтов.

Чтобы неиспользованные участки основной памяти между скалярными переменными были минимальными, транслятор собирает вместе все переменные, которые имеют одинаковые границы выравнивания. Размер требуемой памяти при этом значительно уменьшается. Для нашего примера распределение памяти, произведенное транслятором, будет иметь вид



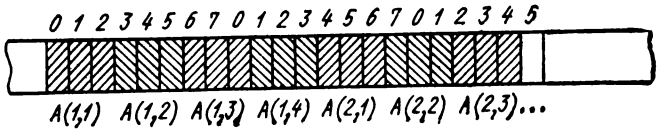
Область памяти, требуемая для размещения массива, равна сумме областей памяти, необходимых для каждого элемента.

Примеры.

1) Объявлен следующий массив:

```
DCL A (7,4) FIXED (5,3);
```

Согласно табл. 4.2 каждому элементу массива, являющемуся десятичным числом с фиксированной точкой, отводится память в соответствии с разрядностью размером 3 байта. Следовательно, для всего массива потребуется $28 \times 3 = 84$ байта. Элементы данных будут располагаться в памяти следующим образом:

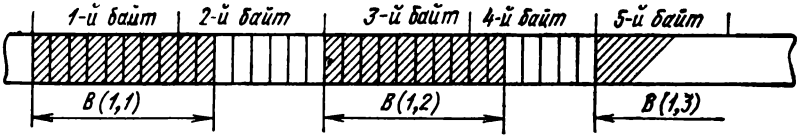


2) Рассмотрим распределение памяти для массива, элементами которого являются строки битов. Если длина строки битов не кратна 8, то появляются неиспользованные биты, так как выравнивание на границу (байт), которое производится для строки битов, будет производиться для каждого элемента массива.

Следовательно, для массива B со строками битов, объявленного в операторе,

```
DCL B (5,5) BIT (10);
```

требуется область памяти, равная 2 × 25 = 50 байтов, а элементы массива расположатся в памяти так:



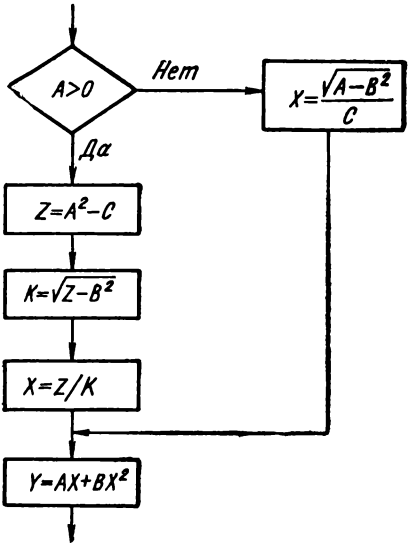
При объявлении структуры с помощью уровней устанавливается последовательность ее элементов. Элементы структуры — в общем случае переменные разного типа и, следовательно, могут иметь различные границы выравнивания. При объявлении имени старшей структуры с атрибутами ALIGNED транслятор будет учитывать требование о выравнивании каждого элемента структуры на соответствующую границу. Порядок элементов при этом не изменяется, иначе объединение элементов в структуру потеряло бы всякий смысл. Естественно, что при таком распределении памяти имеются неиспользованные участки. Чтобы сократить их количество, программист при объявлении структуры должен позаботиться о том, чтобы непосредственно друг за другом не следовали элементы с различными границами выравнивания (если, конечно, допустимо изменение порядка элементов).

Глава 5

ПРОГРАММИРОВАНИЕ РАЗВЕТВЛЯЮЩИХСЯ И ЦИКЛИЧЕСКИХ ВЫЧИСЛИТЕЛЬНЫХ ПРОЦЕССОВ

5.1. Оператор DO первого типа

При программировании разветвляющихся вычислительных процессов с помощью условного оператора перед программистом часто возникает необходимость поместить за ключевыми словами THEN или ELSE несколько операторов. Однако правила написания условного оператора не позволяют этого сделать. Поэтому для реализации таких алгоритмов программист вынужден совместно с условным оператором использовать операторы перехода. Рассмотрим следующий фрагмент такой задачи:



Программа этого алгоритма может быть записана так:

```
      IF A > 0 THEN GOTO M1;
         ELSE X = SQRT (A — B**2)/C;
      GOTO M2;
M1 : Z = A**2 — C;  K = SQRT (Z — B**2);  X = Z/K;
M2 : Y = A*X + B*X**2;
```

Это можно было бы записать короче и наглядней, если бы была возможность записать три оператора в ветви THEN. Такая возможность реализуется в языке ПЛ/1 путем объединения нескольких операторов в один составной оператор с помощью операторов DO и END. Такая совокупность объединенных операторов называется в языке оператором DO первого типа. Общий формат этого оператора имеет вид

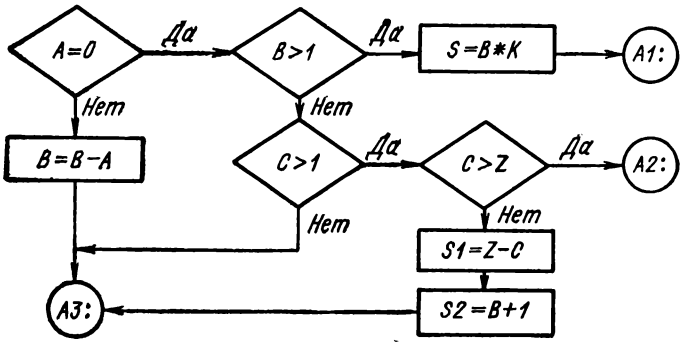
```
[метка:] DO;
                оператор 1;
                оператор 2;
                .............
                оператор n;
         END [метка];
```

Такая группа из *n* операторов представляет собой как бы один составной оператор и может быть использована в любом месте программы, где может быть записан один оператор.

Для нашего примера этот оператор может быть использован так

```
IF A > 0 THEN DO;  Z = A ** 2 — C;
                   K = SQRT (Z — B ** 2);
                   X = Z/K;
                END;
   ELSE X = SQRT (A — B ** 2)/C;
Y = A * X + B * X ** 2;
```

Пример. Необходимо описать следующий алгоритм с помощью условного оператора и DO-группы:



Программа, реализующая данный алгоритм, будет иметь вид

```
IF A = 0 THEN IF B > 1 THEN
   DO; S = B * K; GOTO A1; END;
   ELSE IF C > 1 THEN IF C > Z THEN GOTO A2;
     ELSE DO; S1 = Z - C; S2 = B + 1; GOTO A2; END
        ELSE GOTO A3;
   ELSE DO; B = B - A; GOTO A3; END;
A1: . . . . . .
A2: . . . . . .
A3: . . . . . .
```

5.2. Оператор DO второго типа

Для описания циклических вычислительных процессов в языке ПЛ/1 может использоваться оператор DO второго или третьего типа.

Если последовательность операторов должна выполняться до тех пор, пока выполняется некоторое условие, то должен использоваться оператор DO второго типа. Он имеет такой общий формат:

```
[метка:] DO WHILE (выражение);
           оператор 1;
           оператор 2
           .....
           оператор n;
         END;
```

Этот оператор обеспечивает повторение операторов DO-группы в зависимости от значения выражения, стоящего в скобках. В качестве последнего можно использовать либо выражение типа сравнения, либо скалярное выражение.

При использовании выражения типа сравнения повторное выполнение операторов в группе производится до тех пор, пока значение выражения истинно. Как только значение выражения становится ложным, операторы DO-группы пропускаются и управление передается на оператор, непосредственно следующий за END. Если значение выражения оказывается ложным с самого начала, то операторы в DO-группе совсем не выполняются.

При использовании скалярного выражения выполнение группы операторов (начало цикла) начинается с вычисления значения выражения. Если необходимо, то после вычисления выражение преобразуется в строку битов. Если хотя бы один бит этой строки равен 1, то происходит выполнение операторов DO-группы. После выполнения последнего оператора группы управление передается в начало оператора DO (в начало цикла). Выражение опять вычисляется и проверяется. Если значения всех битов в строке равны 0, то выполнение операторов DO-группы прекращается. В этом случае не происходит повторения выполнения операторов DO-группы,

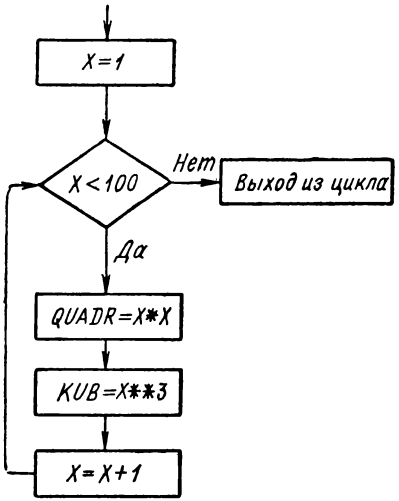
а управление передается оператору, непосредственно следующему за оператором END.

При использовании оператора DO второго типа программисту необходимо позаботиться, чтобы значение хотя бы одного из операндов, входящих в выражение, изменялось операторами DO-группы. Иначе повторение выполнения операторов группы может быть бесконечным.

Примеры.

1) Необходимо вычислить квадраты и кубы положительных чисел из интервала $1 \leqslant x \leqslant 100$.

Схема программы:



Эта часть программы может быть записана следующим образом:

```
X = 1;
DO WHILE (X <= 100);
    QUADR = X * X;
    KUB = X ** 3;
    X = X + 1;
END;
```

2) Одномерный массив D десятичных чисел с фиксированной точкой содержит 78 элементов. Необходимо вычислить сумму элементов, выбирая их через два, начиная со второго. Значения элементов массива вводятся с перфокарт, а результат вычислений выдается на печать.

```
SUMMA: PROCEDURE OPTIONS (MAIN);
       DECLARE D (78) FIXED (4,2);
               SUM FIXED (6,2);
       GET EDIT (D) (F (4,2));
       SUM = 0;  I = 2;
       DO WHILE (I <= 78);
```

```
            SUM = SUM + D (I);
            I = I + 3;
        END;
        PUT EDIT (SUM) (SKIP, X (10), F (8,2));
END SUMMA;
```

В качестве выражения в данном примере использовано выражение типа сравнения (I <= 78). Выполнение цикла будет продолжаться до тех пор, пока значение переменной I, изменяющейся внутри цикла, будет меньше или равно 78.

5.3. Оператор DO третьего типа

Программист может организовать циклический вычислительный процесс более гибко с помощью оператора DO третьего типа. Общий формат этого оператора:

```
[метка:] DO переменная = спецификация [, спецификация]...,
                оператор 1;
                оператор 2;
                .....
                оператор n;
        END;
```

Спецификация имеет следующий вид:

```
выражение 1 [{BY выражение 2 [TO выражение 3]}]
            [{TO выражение 3 [BY выражение 2]}]
             [WHILE (выражение 4)]
```

«Переменная» — это управляющая переменная оператора DO. Она является неиндексированной переменной арифметического типа;

«выражение 1» — начальное значение управляющей переменной;

«выражение 2» — приращение, которое после каждого выполнения DO-группы добавляется к управляющей переменной;

«выражение 3» — конечное значение управляющей переменной;

«выражение 4» указывает дополнительные условия выхода из DO-группы (окончание цикла).

Все выражения в операторе DO должны быть скалярными. Выражения над массивами или структурами не допускаются.

Рассмотрим следующую форму оператора DO:

```
DO переменная = выражение 1 {BY выражение 2 [TO выражение 3]};
                            {TO выражение 3 [BY выражение 2]};
```

Эта форма оператора обеспечивает повторение операторов DO-группы до тех пор, пока управляющая переменная не примет все значения от начального до конечного с заданным приращением.

Вначале вычисляются начальное и конечное значения и приращение управляющей переменной. Затем проверяется, должны ли выполняться операторы DO-группы. Если приращение положительное, происходит проверка, не превышает ли значение управляющей переменной конечного значения. При превышении конечного значения управление будет передаваться оператору, следующему за DO-группой. В противном случае выполняются операторы цикла DO. Затем к управляющей переменной прибавляется приращение и происходит проверка на конечное значение. Если значение управляющей переменной превышает конечное значение, то операторы цикла DO не выполняются. При отрицательном приращении в начале каждого цикла проверяется, является ли значение управляющей переменной меньше конечного значения. Цикл повторяется до тех пор, пока управляющая переменная не достигнет конечного значения (включая и его).

Пример. Вычислить и отпечатать на одной строчке значение функции

$$y = x \operatorname{tg} \frac{x}{2} + 2 \ln \left| \cos \frac{x}{2} \right|$$

для x = 0.5, 0.7, 0.9, 1.1, 1.3, 1.5.

Результаты вычислений описать как десятичные числа с фиксированной точкой и двумя цифрами после запятой

```
PR: PROCEDURE OPTIONS (MAIN);
    DCL X FIXED (2,1),
        Y (6) FIXED (4,2);
    I = 0;
    DO X = 0.5 BY 0.2 TO 1.5;
        I = I + 1;
        Y (I) = X * TAN (X/2) +
            2 * LOG (ABS (COS (X/2)));
    END;
    PUT EDIT (Y) (SKIP, 6 (X (2), F (6,2)));
END PR;
```

В данном примере в качестве значений управляющей переменной X взяты константы как частный случай выражений

При использовании указанной выше формы оператора DO может быть опущена конструкция «TO выражение 3». В этом случае окончание выполнения цикла должно определяться условным оператором внутри DO-группы.

Так, для нашего примера:

```
    DO X = 0.5 BY 0.2;
        IF X > 1.5 THEN GOTO M1;
        I = I + 1;
        Y (I) = X * TAN (X/2) + 2 * LOG (ABS (COS (X/2)));
    END;
M1: PUT EDIT (Y) (SKIP, 6 (X(2), F(6,2)));
```

Оператор DO третьего типа позволяет не указывать приращение в том случае, если оно полагается 1:

DO переменная = выражение 1 TO выражение 3;

Пример. Необходимо написать программу вычисления значений функции

$$y = 8{,}9x + 2{,}3x^2 - 1{,}16x^3$$

для аргумента x, лежащего в пределах

$$1{,}0 \leqslant x \leqslant 5{,}9$$

и меняющегося с шагом 0,1. Результат решения оформить в виде вектора.

Напишем сначала эту программу с помощью условного оператора и оператора перехода:

```
      DECLARE Y (50) FIXED DECIMAL (4,2);
              X FIXED (2,1);
      X = 1; I = 0;
M1:   I = I + 1;
      Y (I) = 8.9 * X + 2.3 * X * * 2 - 1 16 * X * * 3;
      X = X + .1;
      IF I <= 50 THEN GOTO M1;
```

Этот же фрагмент программы проще описать с помощью оператора DO третьего типа:

```
DECLARE Y (50) FIXED DECIMAL (4,2),
        X FIXED (2,1);
X = 1;
DO I = 1 TO 50;
        Y (I) = 8.9 * X + 2.3 * X * * 2 -1.16 * X * * 3;
X = X + .1;
END;
```

Кроме того, имеется возможность выполнить DO-группу с единственным значением управляющей переменной, не задавая ни конечного значения, ни приращения. Такой оператор имеет следующую форму:

```
DO переменная = выражение 1;
    оператор 1;
    оператор 2;
    .....
    оператор n;
END;
```

Рассмотрим теперь полную форму оператора DO третьего типа.

```
DO переменная = выражение 1 BY выражение 2
    TO выражение 3 WHILE (выражение 4);
    оператор 1;
    оператор 2;
    . . . . . .
    оператор n;
END;
```

Этот оператор обеспечивает повторение операторов DO-группы до тех пор, пока значение управляющей переменной не превзошло конечное значение или пока выполняется условие в выражении 4.

Количество повторений цикла DO зависит от начального значения, приращения, конечного значения управляющей переменной и выражения 4.

Пример.

```
DO I = 1 TO N*N + 1 BY N
    WHILE (B > 10.5E - 8);
    A (I) = A (I) + B;    B = B/2;
END;
```

Выход из цикла DO произойдет, когда $I > N*N + 1$ или $B \leqslant 10.5E - 8$.

В операторе DO третьего типа для управляющей переменной может быть указан список спецификаций. В этом случае управляющая переменная последовательно принимает все значения, определенные каждой спецификацией списка.

Пример

```
DO I = 6 TO 9, 12, 17, 20 TO 40 BY 4, 49,
    50 BY 3 WHILE (I < 75), 81;
```

В этом случае группа DO будет выполняться для следующих значений управляющей переменной:

I = 6, 7, 8, 9, 12, 17, 20, 24, 28, 32, 36, 40, 49, 50, 53, 56, 59, 62, 65, 68, 71, 74, 81;

Свойства групп и циклов DO. Группа операторов, начинающихся оператором DO, рассматривается как единое целое и имеет следующие свойства:

— спецификация оператора DO может содержать скалярные выражения;

— выражения для начального значения, приращения и конечного значения управляющей переменной вычисляются при выпол-

нении оператора DO только один раз. Операторы DO-группы не могут изменить эти выражения. Возможно изменение значения управляющей переменной. Если при этом изменении значение управляющей переменной превысит конечное значение, то при положительном приращении произойдет выход из цикла DO;

— возможна передача управления из DO-группы до окончания цикла при помощи оператора перехода GOTO;

— передавать управление оператором DO-группы извне можно только в том случае, если группа начинается оператором DO первого типа;

— если цикл DO, повторение которого регулировалось управляющей переменной, заканчивается естественным путем, то управляющая переменная имеет значение, которое привело к окончанию выполнения цикла.

Пример.

```
DO I = 10 BY 5 TO 60;
.  ........
END;
DO K = 12 BY 0.2 TO 20.25;
........
END;
```

После естественного окончания цикла DO управляющие переменные будут иметь следующие значения:

I = 65, K = 20.45;

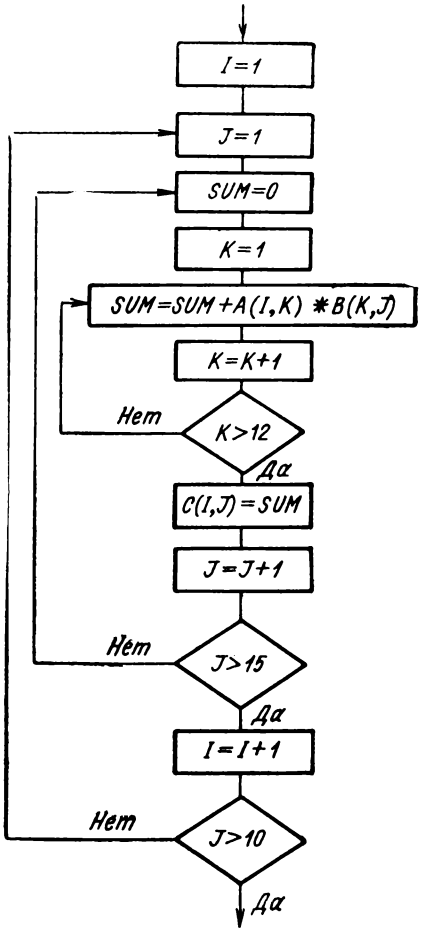
— если выполнение цикла DO прекращается с помощью оператора GOTO или при помощи выражения 4, то управляющая переменная будет иметь значение, которое она получила при соответствующем выполнении цикла.

5.4. Вложенные циклы

На практике часто сталкиваются с задачами, для решения которых необходимо использовать несколько вложенных циклических процессов. Для этого в программе на ПЛ/1 можно группу операторов DO записать внутри другой DO-группы. Максимальный уровень вложенности групп равен 12. Вложенные группы в совокупности могут содержать не более 50 спецификаций.

Рассмотрим конструкцию вложенных циклов на следующем примере.

Матрица C (10,15) является результатом умножения матрицы A (10,12) на матрицу B (12, 15). Схема решения этой задачи имеет вид



Программирование этих трех циклов производится с помощью вложенных операторов DO, причем управляющие переменные используются как индексы массивов (матриц).

Программа данного алгоритма будет иметь вид

```
MATR:   PROCEDURE OPTIONS (MAIN);
        DCL (A (10, 12), B (12, 15) G (10, 15), SUM)
            FIXED (3,1);
        GET EDIT (A  B) (F (5,1));
        DO I = 1 TO 10;
           DO J = 1 TO 15;
             SUM = 0;
             DO K = 1 TO 12;
                 SUM = SUM + A (I,  K)*B (K,  J);
```

```
            END;
            C (I, J) = SUM;
          END;

        END;
        PUT EDIT (C) (SKIP, 15(X (2), F(5,1)));
END MATR;
```

В программе использованы три вложенных цикла. Каждый оператор DO обеспечивает итерационный процесс вычислений при изменении соответствующего индекса массива.

Примеры.

1) Уровень месячной продукции за последние 5 лет накоплен в виде последовательности, составляющей вектор X. Необходимо подсчитать вектор D, который содержит сведения о среднемесячной выработке продукции для каждого месяца года.

```
DECLARE (D (12), X (60)) DECIMAL (3);
DO  K = 1 TO 12;
    Z = 0;
    DO I = K TO 60 BY 12;
       Z = Z + X (I);
    END;
       D (K) = Z/5;
END;
```

2) Решение системы алгебраических уравнений выражается с помощью формулы Гаусса—Зейделя:

$$x_i = (b_i - \sum_{k=i+1}^{n} a_{i,\,k}\, x_k)/a_{i,\,i},$$

где n — порядок системы уравнений; x_i — i-й корень уравнения. При этом значения коэффициентов $a_{i,\,k}$ являются элементами матрицы, а b_i — элементами вектора.

Необходимо составить программу для решения системы из 30 уравнений. Нахождение решения необходимо начинать с последнего корня.

```
PRO1: PROCEDURE OPTIONS (MAIN);
      DECLARE (A (30, 30), B (30), X (30)) FIXED (6,2);
      GET EDIT (A, B) (F (6,2));
      DO  I = 30 TO 1 BY — 1;
          SUM = B (I);
      DO  K = I + 1 TO 30;
          SUM = SUM — A (I, K) * X (K);
      END;
      X (I) = SUM/A (I, I);
      END;
      PUT EDIT (X) (SKIP, X (20), F (8,2));
END PRO1;
```

Глава 6

БЛОКИ

Большинство алгоритмов при описании можно разбить на отдельные смысловые части, которые обеспечивают как наглядность самой программы, так и возможность разделения работы при программировании крупных задач. Чтобы выполнить такое разделение, программист может использовать DO-группу первого типа. Как мы уже видели, операторы, входящие в группу, выполняются в естественном порядке (они как будто не входят в DO-группу), но, с другой стороны, DO-группа представляет собой один оператор (так называемый составной оператор), ограниченный «операторными скобками» DO и END. Целесообразность такого средства языка понятна. Оно облегчает написание отдельных ветвей программы. Кроме того, для выполнения такой задачи программисты имеют в своем распоряжении и другие средства — блоки как расширение DO-групп.

Блок — наиболее крупная часть алгоритма задачи. Каждый блок имеет самостоятельное смысловое значение. Блоки ограничивают последовательность операторов, составляющих некоторую часть программы. Они используются для локализации областей действия имен, распределения памяти для переменных с целью более гибкого ее использования, а также для управления.

В свою очередь, в состав блока входит последовательность других операторов. Она представляет собой последовательность действий, которую необходимо выполнить, чтобы получить результат работы данного блока.

Блоки бывают двух типов: обычные и процедурные.

Обычные блоки (BEGIN-блоки) являются расширением DO-групп в том смысле, что дополнительно к вышеуказанному свойству «операторной скобки», которым обладает DO-группа, с их помощью определяется область действия идентификаторов, явно описанных в нем, от чего, в свою очередь, зависит обычно распределение областей памяти для этих переменных.

Процедурные блоки (блоки PROCEDURE) так же, как и обычные, представляют собой совокупность операторов и ограничивают область действия идентификаторов, объявленных в них. Кроме

того, они позволяют обращаться к ним как к подпрограммам, передавая им для работы некоторые параметры.

Любая программа должна состоять по крайней мере из одного процедурного блока. Приводимые ранее программы представляли собой отдельные процедурные блоки. Каждый обычный блок должен содержаться либо в процедурном, либо в другом обычном блоке. Глубина вложенности любых блоков не должна превышать трех.

6.1. Обычный блок

Обычный блок представляет собой последовательность операторов, которая начинается оператором BEGIN и заканчивается оператором END. Он имеет следующий формат:

```
[метка:] ... BEGIN;
                оператор 1;
                оператор 2;
                . . . . . .
                оператор n;
            END [метка];
```

По аналогии с DO-группой операторы, находящиеся между символами BEGIN и END, объединяются и рассматриваются как один оператор. Как и отдельный оператор, обычный блок может быть помечен. Для этого перед оператором BEGIN записывается одна или несколько меток.

Если в операторе END, принадлежащем блоку, использована метка, то она должна совпадать по написанию с меткой, непосредственно предшествующей оператору BEGIN (по аналогии с DO-группой).

Активизация, т. е. начало работы обычного блока, происходит тогда, когда в процессе последовательного выполнения или с помощью оператора перехода управление достигает начала блока, т. е. оператора BEGIN. В обычный блок можно войти и активизировать его только через оператор BEGIN, т. е. передача управления внутрь неактивного блока запрещена.

Завершение работы обычного блока может осуществляться следующим образом:

— если управление в блоке дошло до принадлежащего ему оператора END. В этом случае выполнение программы продолжается с оператора, непосредственно следующего за END;

— в результате выполнения оператора перехода GOTO, расположенного внутри этого блока. В этом случае выполнение программы продолжается с оператора, метка которого указана в операторе GOTO;

— если внутри блока встречается оператор STOP, с помощью которого завершается выполнение всей программы.

Необходимо отметить, что с помощью оператора перехода могут быть завершены несколько блоков.

Пример.

```
PR2: PROC OPTIONS (MAIN);
     оператор 1;
      оператор 2;
     A: BEGIN;
        оператор 3;
        оператор 4;
        B: BEGIN;
            оператор 5;
            IF L > 0 THEN GOTO M1;
            оператор 6;
     END B;
     оператор 7;
     END A;
     оператор 8;
 M1: оператор 9;
END PR2;
```

В этой программе вначале выполняются два первых оператора. Затем активизируется блок А, и выполняются следующие два оператора. Далее начинается выполнение операторов блока В. При выполнении условного оператора, если переменная L будет действительно больше нуля, произойдет выход из блока В. Управление будет передано оператору с меткой М1, в результате чего произойдет завершение работы как блока В, так и блока А. В противном случае оба блока завершат свою работу естественным путем, т. е. при достижении соответствующих операторов END.

Обычный блок должен содержаться внутри процедуры или другого обычного блока, т. е. может находиться только на втором или третьем уровне.

6.2. Процедурный блок

Часто в алгоритмах встречаются отдельные типовые участки, вычисления по которым необходимо производить во многих местах программы с различными значениями исходных данных. При составлении программы можно описывать эти участки каждый раз, когда они потребуются. Такой подход к составлению программы является принципиально правильным, но он приводит к излишнему расходу времени программирования и памяти машины, так как программа при этом получается неоправданно длинной.

Для сокращения объема программы целесообразно выделить многократно повторяющийся участок вычислений из общей программы, записать его отдельно и обращаться к нему каждый раз, когда появится необходимость его выполнения. Выделенный из общей программы участок в языке ПЛ/1 и получил название процедурного блока.

В программе процедурные блоки (или просто процедуры) могут оформляться в двух формах:

— в виде процедуры-подпрограммы;

— в виде процедуры-функции (или просто функции).

Процедура-подпрограмма представляет собой совокупность операторов, в результате выполнения которых вычисляется значение одной или нескольких переменных.

Процедура-функция также является совокупностью операторов, в результате выполнения которых вычисляется всегда только одно значение переменной.

Процедурный блок состоит из оператора PROCEDURE (сокращенно PROC), «тела» процедуры и оператора END. Главной частью процедуры является «тело» процедуры, которое и представляет собой программу выделенного типового участка алгоритма.

Оператор PROCEDURE определяет начало процедурного блока и его основную точку входа. Он может содержать также и другие сведения, необходимые для работы процедурного блока.

Формально любой процедурный блок имеет следующее строение:

```
имя точки входа: PROCEDURE {[(список формальных парамет-]  }
                           {ров)] [атрибуты функции]      };
                           {OPTIONS (MAIN [, ONSYSLOG])   }
        оператор 1;
        оператор  ;
        . . . . . .
        . . . . . .
        оператор n;
END [имя точки входа];
```

Операторы PROCEDURE и END ограничивают процедурный блок.

«Имя точки входа» является идентификатором и обозначает основную точку входа процедурного блока, начиная с которой этот блок должен выполняться. Имя входа не должно состоять более чем из шести символов.

«Список формальных параметров» представляет собой список переменных (точнее, их имен), над которыми производятся действия в процедурном блоке и которым при каждом обращении к процедуре должны присваиваться конкретные значения. Результаты выполнения процедуры (а их может быть несколько) также являются формальными параметрами. Максимальное число формальных параметров, которые могут быть заданы в одной процедуре (включая все параметры, которые могут быть указаны в дополнительных точках входа), не должно превышать 12.

Фактическое появление идентификатора в списке формальных параметров процедуры является явным его объявлением с атрибутами по умолчанию. Такое объявление может сочетаться с явным объявлением с помощью оператора DECLARE (внутри «тела» процедуры), если атрибуты по умолчанию не устраивают программиста.

В качестве формальных параметров могут быть использованы имена скалярных переменных, массивов, структур, имена входов и файлов, а также имена переменных типа метки или указателя.

«Атрибуты-функции» могут использоваться только при описании процедуры функции. Они определяют атрибуты вычисляемого функцией значения.

Второй вариант указывается только для внешнего процедурного блока программы, т. е. для главной процедуры. В нем могут использоваться два режима:

— MAIN указывает, что процедурный блок является главной процедурой, которая будет вызвана операционной системой;

— ONSYSLOG обеспечивает вывод всех сообщений об ошибках при выполнении программы на пишущую машинку оператора.

При описании некоторых типовых участков программ с помощью процедур бывает целесообразно иметь несколько отдельных входов в различные точки одной и той же процедуры. Эти дополнительные точки входа в процедуру описываются с помощью оператора ENTRY, который имеет вид

имя входа: ENTRY [(список параметров)] [атрибуты функции];

Имена дополнительных точек входа, заданные оператором ENTRY, используются для вызова таким же образом, как и имя основной точки входа. В формате оператора «список параметров» и «атрибуты функции» имеют то же значение, что и в операторе PROCEDURE.

Параметры, указанные в операторе ENTRY, должны быть явно объявлены своим появлением в списке параметров оператора PROCEDURE или (и) в операторе DECLARE внутри «тела» процедуры.

Относительно области действия и распределения памяти для переменных процедурные блоки имеют те же функции, что и обычные блоки. Они отличаются от последних возможностью обмена информацией, основанной на принципе аргументов и параметров, и способом активизации.

В отличие от обычного блока выполнение операторов процедурного блока (его активизация) начинается только при вызове процедуры специальными средствами, т. е. при передаче управления на любой из ее входов. Кроме того, формальным параметрам процедуры во время ее вызова должны быть переданы некоторые конкретные значения. После завершения работы процедурного блока управление и результаты возвращаются в точку вызова.

6.3. Вложение блоков

Любая программа может состоять из нескольких отдельных блоков. Любой блок может содержать один или несколько других блоков, т. е. блоки могут вкладываться один в другой. Максимальная глубина вложенности блоков равна трем, причем на первом уровне вложенности всегда находится внешняя процедура.

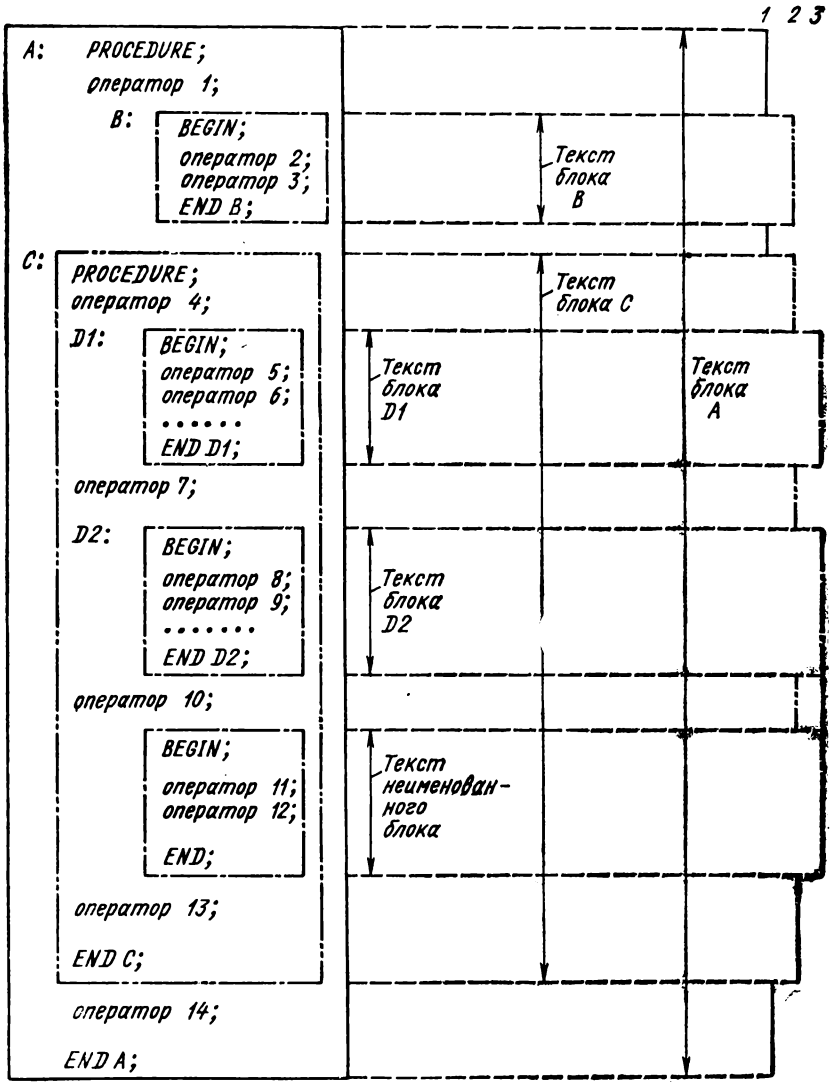


Рис. 6.1. Пример вложения блоков.

На рис. 6.1 приведен пример вложения блоков. Блок A содержит блоки B и C; блок C содержит D1 и D2 и неименованный блок, тексты блоков D1 и D2, а также текст неименованного блока являются связанной частью текста блока C, который так же, как и текст блока B, является связанной частью текста блока A.

Под текстом блока понимается совокупность операторов, входящих в блок, включая операторы BEGIN (PROCEDURE) и END,

а также метки оператора BEGIN и имена всех точек входа в процедуру.

Диаграмма, расположенная справа, показывает глубину вложенности каждого блока (цифры 1, 2, 3).

Блок, содержащийся в другом блоке, называется внутренним. В нашем примере все блоки, кроме А, являются внутренними блоками. Блок, который не содержится в другом блоке, т. е. охватывающий блок, называется внешним. Имена точек входа внешнего процедурного блока (как основной точки входа, так и дополнительных) рассматриваются как внешние имена. Поэтому для них должны быть использованы идентификаторы длиной не более 6 символов.

Программа на языке ПЛ/1 может иметь несколько внешних процедур. Однако транслятором каждая внешняя процедура обрабатывается отдельно. Результатом обработки является объектный модуль соответствующей внешней процедуры. Только программа РЕДАКТОР соединяет объектные модули всех относящихся к программе внешних процедур в единую рабочую программу (рабочую фазу). Если программа состоит только из одной внешней процедуры, то она должна быть записана как главная процедура.

На рис. 6.1 текст, содержащийся в блоках, заключен в рамки. Под «внутренним» к блоку текстом понимают текст, который содержится в нем, но не содержится во внутренних для него блоках. Таким образом, текст, внутренний к блоку, расположен на первом уровне этого блока. Отсюда следует, что метки и имена точек входов блока не относятся к внутреннему тексту этого блока, а являются внутренними к охватывающему блоку.

В нашем примере текст, внутренний по отношению к блоку А:

```
PROCEDURE;
оператор 1;
B:
C:
оператор 14;
END A;
```

Текст, внутренний к блоку B:

```
BEGIN;
оператор 2;
оператор 3;
END B;
```

Текст, внутренний к блоку C:

```
PROCEDURE;
оператор 4;
D1:
оператор 7;
D2:
оператор 10;
оператор 13;
END C;
```

6.4. Объявление имен

При написании программы в ней используются идентификаторы, которые определяют имена объектов, обрабатываемых в этой программе. Под различными объектами понимаются скалярные переменные, массивы, структуры, файлы и др.

Однако в различных частях программы одно и то же имя может определять различные объекты. Поэтому каждый объект, обрабатываемый в программе, должен быть объявлен с помощью соответствующих атрибутов. Объявление устанавливает тип объекта, определяет его имя и область действия этого объекта.

Область действия имени, т. е. та область, где идентификатор объявлен как имя некоторого объекта и может быть там использован, — это блок, в котором содержится объявление этого имени, или внешняя процедура, которая содержит этот блок.

В языке различают три типа объявлений: явное, контекстуальное и неявное.

Явное объявление имени производится:

— оператором DECLARE;

— появлением в качестве метки оператора;

— появлением в качестве имени точки входа перед оператором PROCEDURE или перед оператором ENTRY;

— появлением в списке параметров оператора PROCEDURE.

Оператор DECLARE явно объявляет идентификаторы как внутренние имена блока, для которого оператор DECLARE является внутренним. Он может появиться в любом месте внутреннего текста блока, но всегда воспринимается транслятором так, как будто он записан в начале блока.

Метка оператора — это идентификатор, который отделен от оператора двоеточием, явно объявляется как метка оператора (константа типа метки). Она является внутренней по отношению к блоку, для которого этот оператор также является внутренним. Константа типа метки имеет атрибут LABEL.

Имя точки входа — это идентификатор перед оператором PROCEDURE или ENTRY, отделенный от оператора двоеточием. Он считается явно объявленным как имя точки входа процедурного блока, во внутреннем тексте которого содержится соответствующий оператор. Идентификатор может быть явно объявлен как имя точки входа при помощи оператора DECLARE с атрибутом ENTRY.

Если процедурный блок является внутренним блоком, то имена точек входа являются внутренними по отношению к охватывающему блоку. В случае внешнего процедурного блока имя точки входа рассматривается транслятором как внутреннее к этой внешней процедуре, хотя и объявленное с атрибутом EXTERNAL.

Параметр. Идентификаторы в списке параметров оператора PROCEDURE явно объявляются как внутренние имена процедурного блока, который начинается этим оператором. Атрибуты присваиваются по умолчанию в соответствии с начальной буквой иден-

тификатора (начинающиеся с букв от I до N имеют атрибуты FIXED BINARY (15), а остальные — атрибуты FLOAT DECIMAL (6)) или задаются в операторе DECLARE.

Явное объявление позволяет использовать идентификатор в некотором блоке. Под областью действия объявления понимают блок или блоки, в которых идентификаторы могут быть использованы как имена определенных объектов (например, имена переменных). Областью действия явно объявленного имени является блок, по отношению к которому это объявление является внутренним, кроме всех блоков, находящихся в данном, где этот идентификатор снова переобъявлен явно. Рассмотрим следующую программу:

```
PRO: PROCEDURE;
     DECLARE (I 1, I 2) FIXED BINARY;
     DECLARE K FIXED BINARY;
     . . . . . .
     I1 = 0; I2 = 0;
     . . . . . .
     GOTO E1;
     . . . . . .
  L: GET EDIT (K) (F (5));
     . . . . . .
     CALL P1 (I1);
  E1: I1 = I1 + 1;
     . . . . . .
     B: BEGIN;
     DECLARE K CHARACTER (10);
     . . . . . .
     CALL P1 (I2);
     E2: I2 = I2 + 1;
     . . . . . .
     GOTO L;
     . . . . . .
     END B;
     . . . . . .
     GOTO L;
     . . . . . .
     P1: PROCEDURE (I);
     DECLARE P PICTURE '(5)9';
     . . . . . .
     P = I;
     END P1;
END PRO;
```

Имена переменных и их атрибуты приведены в таблице на стр. 92.

В программе приведено явное объявление имен. Для большей наглядности используемый в блоке B идентификатор K, который в блоке PRO является именем другого объекта, на диаграмме обозначается $\hat{K}$.

Контекстуальное объявление. Синтаксис языка ПЛ/1 допускает опознавание идентификаторов, появляющихся в определенном контексте, без их явного объявления. Контекстуально, т. е. в зависимости от текста, можно объявить только имена точек входов.

В примере используются следующие имена (идентификаторы):

Идентификатор	Использование идентификатора	Атрибуты	Область действия
PRO	Имя точки входа	ENTRY EXTERNAL	PRO
I1	Имя переменной	FIXED BINARY (15)	PRO
I2	» »	FIXED BINARY (15)	PRO
K	» »	FIXED BINARY (15)	PRO, кроме B
L	Константа типа метки	LABEL	PRO
E1	» » »	LABEL	PRO
B	» » »	LABEL	PRO
K	Имя переменной	CHARACTER (10)	B
E2	Константа типа метки	LABEL	B
P1	Имя точки входа	ENTRY	PRO
I	Параметр	FIXED BINARY (15)	P1
P	Имя переменной	PICTURE '(5) 9'	P1

Если идентификатор появился в таком месте текста, в котором может находиться только имя точки входа, то он в соответствии с этим контекстом объявляется как имя точки входа. В этом случае идентификатор не нуждается в явном объявлении в качестве имени точки входа.

Контекстуальное объявление идентификаторов распознается в следующих случаях:

— идентификатор следует сразу же за ключевым словом CALL. Это означает, что идентификатор является именем точки входа в процедуру-подпрограмму;

— идентификатор, за которым следует список аргументов в круглых скобках, появился в качестве операнда в выражении. Это означает, что идентификатор является именем точки входа в процедуру-функцию.

Контекстуальное объявление предполагает по умолчанию атрибут области действия EXTERNAL.

Областью действия контекстуально объявленного имени является вся внешняя процедура, которая содержит этот текст, исключая все содержащиеся в этой процедуре блоки, в которых этот идентификатор объявлен явно. Контекстуальное объявление идентификатора равносильно явному объявлению этого имени во внешней процедуре.

Пример. Рассмотрим две внешние процедуры PRO и PE.

```
PRO: PROCEDURE;
     DECLARE (I1, I2, K) FIXED BINARY;
     . . . . . .
     I1 = 0;   I2 = 0;
     . . . . . .
   L: GET EDIT (K) (F (5));
  E1: I1 = I1 + 1;
```

```
        B:  BEGIN;
        DECLARE K CHARACTER (10);
        . . . . . .
        CALL PE (I1);
        E2: I2 = I2 + 1;
        . . . . . .
        GOTO L;
        . . . . . .
        END B;
        . . . . . .
        GOTO L;
        . . . . . .
    END PRO;

        PE: PROCEDURE (I);
        DECLARE P PICTURE  '(5)9';
        . . . . . .
        P = I;
        . . . . . .
        END PE;
```

В первой процедуре PRO идентификатор PE появляется сразу за ключевым словом CALL в блоке B. Несмотря на то, что это внутренний блок, идентификатор будет контекстуально объявлен как имя точки входа внешней процедуры с атрибутом EXTERNAL

Неявное объявление. Идентификатор, который появился в операторах программы без явного или контекстуального объявления, считается объявленным неявно. Следует отметить, что неявное объявление может быть использовано только для скалярных арифметических переменных.

При неявном объявлении идентификатора атрибуты присваиваются ему по умолчанию в зависимости от первой буквы.

Областью действия неявно объявленного идентификатора является вся внешняя процедура, за исключением тех содержащихся в ней блоков, где этот идентификатор явно переобъявлен.

Пример.

```
B1:   PROCEDURE (Z1, Z2);
      TEMP1 = ABS (Z1 ** 2 + Z2 ** 2);
      B2: BEGIN;
      TEMP2 = 1/(TEMP1 + Z2) ** 2;
      IF TEMP2 > TEMP1 THEN RETURN (TEMP2);
      END B2;
END B1;
```

В этом примере неявно объявлены переменные TEMP1 и TEMP2 с атрибутами по умолчанию FLOAT DECIMAL (6). Несмотря на то, что переменная TEMP2 встречается во внутреннем блоке B2, областью ее действия является вся внешняя процедура B1.

6.5. Область действия имен

С помощью объявления идентификатора объекту (например, переменной) присваивается имя. Если в программе объявляется некоторый идентификатор, то это значит, что существует вполне определенная область программы, где это объявление имеет силу.

Эта область называется областью действия объявления или областью действия имени, введенного этим объявлением.

Блок, в котором производится объявление, а также способ, которым идентификатор объявляется, определяют область действия объявления и, следовательно, объявленной переменной. Если в каком-то месте программы необходимо сослаться на некоторую переменную, то это место должно находиться в области действия объявления имени этой переменной.

При объявлении идентификаторов в программе могут использоваться атрибуты области действия INTERNAL и EXTERNAL, с помощью которых программист определяет область действия каждого объявленного имени.

Атрибут INTERNAL указывает, что объявленное имя является внутренним. Область действия такого имени определяется как блок B, для которого это объявление является внутренним, за исключением тех блоков, которые содержатся в блоке B и в которых данное имя переобъявлено как внутреннее имя для этих блоков.

Таким образом, под различными объявлениями одного и того же идентификатора с атрибутом INTERNAL подразумеваются различные объекты (имена) с различными непересекающимися областями действия, т. е. для одинаковых имен в памяти машины отводятся разные поля.

Атрибут EXTERNAL (сокращенно EXT) указывает, что объявленное имя является внешним.

Считается, что все внешние объявления одного и того же идентификатора в программе определяют только одно имя. Это означает, что для одинаковых имен с атрибутом EXTERNAL выделяется одно поле в памяти машины. Областью действия этого имени будет объединение областей действия всех объявлений этого идентификатора. Необходимо отметить, что во всех внешних объявлениях одного и того же идентификатора атрибуты должны быть одинаковыми, так как все эти объявления определяют только одно имя.

Атрибут EXTERNAL может использоваться для установления связи между различными внешними процедурами или получения прерывных областей действия для имени внутри внешней процедуры.

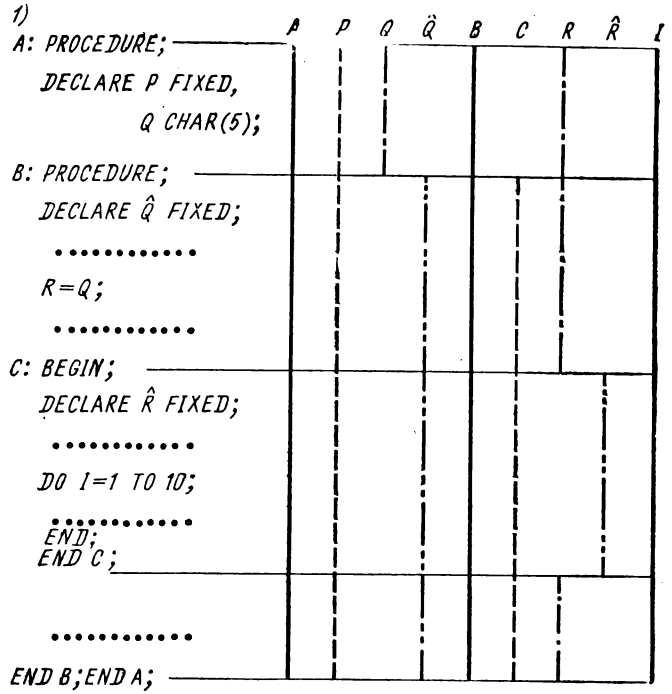
Атрибуты области действия по умолчанию присваиваются следующим образом:

EXTERNAL — для имени точки входа внешней процедуры, для имени файла;

INTERNAL — для имени точки входа внутренней процедуры, для имен переменных, для меток.

Примеры.

1)



Этот пример иллюстрирует область действия объявлений. Для большей наглядности используемый в процедуре B идентификатор Q, который в процедуре A является именем другого объекта, в программе обозначен как $\hat{Q}$. С этой же целью идентификатор R в блоке C обозначен как $\hat{R}$.

В процедурном блоке A явно объявлены:

— переменная A как имя входа в процедуру с атрибутами по умолчанию ENTRY EXTERNAL;

— переменные P и Q с помощью оператора DCL по умолчанию с атрибутами области действия INTERNAL;

— переменная B, как имя входа в процедуру с атрибутами по умолчанию ENTRY INTERNAL.

В процедурном блоке B явно объявлены:

— переменная Q с помощью оператора DCL с атрибутом области действия по умолчанию INTERNAL;

— переменная C как метка обычного блока B с атрибутом по умолчанию INTERNAL.

В обычном блоке C явно объявлена переменная R с помощью оператора DCL и атрибутом по умолчанию INTERNAL.

Неявно объявлены переменные R и I с атрибутом области действия INTERNAL. При этом область действия переменной R — весь процедурный блок A, исключая обычный блок C; переменной I — весь процедурный блок A.

2)

```
P1: PROCEDURE;
    DECLARE GK FLOAT EXTERNAL,
            FK FIXED;
    . . . . . .
```

Таблица 6.1

Идентификатор	Способ объявления	Использование идентификатора	Атрибуты	Атрибут области действия	Область действия
P1	Явный	Имя точки входа	ENTRY	EXTERNAL	P1
GK	Явный	Имя переменной	FLOAT DECIMAL (6)	EXTERNAL	P1, кроме P2 (без B3), B3, KP
FK	Явный	То же	FIXED DECIMAL (5)	INTERNAL	P1, кроме B3
P2	Явный	Имя точки входа	ENTRY	INTERNAL	P1
GK	Явный	Имя переменной	FLOAT DECIMAL (6)	INTERNAL	P2, кроме B3
CALL	Контекстуальный	Имя точки входа внешней процедуры	ENTRY	EXTERNAL (по умолчанию)	P1, кроме B3 EPROC, CALL
B3	Явный	Константа типа метки	LABEL	INTERNAL	P2
CALL	Явный	Имя переменной	BIT (3)	INTERNAL	B3
EPROC	Контекстуальный	Имя точки входа внешней процедуры	ENTRY	EXTERNAL (по умолчанию)	P1, EPROC
FK	Явный	Имя переменной	FIXED DECIMAL (7)	INTERNAL	B3

EPROC	Явный	Имя точки входа внешней процедуры	ENTRY	EXTERNAL (по умолчанию)	EPROC
$\hat{GK}$	Явный	Имя переменной	FLOAT DECIMAL (6)	INTERNAL	EPROC, кроме KP
CALL	Контекстуальный	Имя точки входа внешней процедуры	ENTRY	EXTERNAL (по умолчанию)	EPROC
KP	Явный	Имя точки входа внутренней процедуры	ENTRY	INTERNAL (по умолчанию)	KP
CALL	Явный	Имя точки входа внешней процедуры	ENTRY	EXTERNAL (по умолчанию)	CALL
$\tilde{GK}$	Неявный	Имя переменной	FLOAT DECIMAL (6)	INTERNAL	CALL

```
P2: PROCEDURE;
    DECLARE ^GK;
    . . . . . .
    CALL CALL;
    . . . . . .
    B3: BEGIN;
DECLARE GK EXTERNAL,
        CALL BIT (3);
      CALL EPROC;
        DECLARE ^FK FIXED (7);
        . . . . . .
    END B3;
    . . . . . .
END P2;
. . . . . .
END P1;
EPROC: PROCEDURE;
    DECLARE ^^GK FIXED;
    . . . . . .
    CALL CALL;
    . . . . . .
    CALL KP;
    . . . . . .
    KP: PROCEDURE;
    DECLARE GK EXTERNAL;
    . . . . . .
    END KP;
    . . . . . .
END EPROC;
CALL: PROCEDURE;
    . . . . . .
    GK = ~GK/17;
END CALL;
```

В данном примере две переменные имеют имя FK. В обычном блоке B3 имя FK означает переменную с атрибутами FIXED (7), а в процедурном блоке P1 — переменную с атрибутами FIXED DECIMAL (5).

Идентификатор CALL для внешних процедур CALL, EPROC и P1 объявляется как внешнее имя точки входа, кроме B3, где CALL означает переменную типа строка битов длиной 3 бита.

Идентификатор GK используется для обозначения четырех различных переменных одного и того же вида. Четыре области действия имени GK;

GK — внешняя процедура P1 за исключением внутренней процедуры P2 (кроме блока B3), блок B3 и внутренняя процедура KP;

$\widehat{GK}$ — внутренняя процедура P2 за исключением блока B3;

$\widehat{\widehat{GK}}$ — внешняя процедура EPROC за исключением внутренней процедуры KP;

$\widetilde{GK}$ — внешняя процедура CALL.

Назначение всех идентификаторов, способ объявления и их атрибуты показаны в табл. 6.1.

6.6. Способы распределения памяти

В языке ПЛ/1 включены средства, предоставляющие программисту практически любую степень контроля над распределением памяти для данных. С другой стороны, программист может целиком пренебречь вопросами размещения, если экономия памяти в его программе не имеет большого значения, однако и тогда будет автоматически получаться некоторое достаточно эффективное распределение памяти.

Как известно, на переменную можно ссылаться в программе только тогда, когда ее имя объявлено и область действия распространяется на место, где производится эта ссылка. Но чтобы с переменной в программе можно было работать и чтобы ей можно было присвоить значение, этого еще недостаточно.

Необходимо переменной выделить (распределить) некоторую область в памяти машины. Подготовку области памяти можно производить различными способами.

В одном случае в начале программы для каждой встречающейся в программе переменной выделяется постоянная область в памяти. В этой области хранятся значения переменной, которые она принимает в процессе выполнения программы. Такой способ дает возможность сохранить значения переменных на весь период решения задачи. Однако при большом общем количестве переменных, обрабатываемых в программе, и эпизодическом использовании отдельных небольших групп переменных в операторах программы такой способ может быть крайне неэкономичен с точки зрения эффективного использования основной памяти машины.

Другой способ заключается в том, чтобы выделение области памяти для переменной сделать зависимым от активизации блока, т. е. от начала его выполнения. В этом случае область памяти для переменных выделяется только на период выполнения данного блока. По окончании его работы место в памяти освобождается для размещения значений других переменных программы. Такой способ позволяет экономить объем основной памяти для размещения переменных, но не позволяет использовать значения переменных при выходе из блока.

В языке ПЛ/1 имеются средства, позволяющие программисту определить способ распределения памяти для некоторой переменной. Этими средствами являются присвоение переменной определенного класса памяти с помощью соответствующих атрибутов.

В данном подмножестве языка имеются три класса памяти:

— статический, атрибут STATIC;

— динамический, атрибут AUTOMATIC;

— базированный, атрибут BASED (переменная типа указателя).

При статическом классе памяти место для переменных распределяется перед выполнением программы в так называемой области статической памяти. Переменные, объявленные с атрибутом

STATIC, остаются размещенными в основной памяти на все время выполнения программы.

Для статической переменной могут быть использованы атрибуты области действия как INTERNAL, так и EXTERNAL. Кроме того, необходимо отметить, что переменные, объявленные с атрибутом EXTERNAL, по умолчанию имеют класс памяти STATIC.

Для динамического класса памяти осуществляется блочное управление памятью. Для каждого блока существует динамическая память, которая содержит область для всех описанных в блоке переменных. При активизации блока определяется необходимая для использования область памяти. При завершении блока она снова освобождается. Это освобождение является причиной того, что значения переменных, объявленных только в этом блоке, теряются. Переменная с атрибутом AUTOMATIC может иметь атрибут области действия только INTERNAL.

Подготовку памяти для базированной переменной осуществляет программист. Для такой переменной в программе должно быть указано, какую область памяти использовать. Подробное знакомство с базированными переменными и их объявлением будет сделано позднее.

Для всех трех классов памяти существует правило, что на переменную можно ссылаться в программе, если известно ее имя и для нее определена область памяти.

Пример. Имеется следующая программа:

```
PRO: PROCEDURE OPTIONS (MAIN);
     DECLARE Z STATIC EXTERNAL;
     Z = 0;
     . . . . . .
A:   оператор;
     CALL UP;
     PUT EDIT (Z) (SKIP, F (5));
     GOTO A;
     . . . . . .
END PRO;
UP:  PROCEDURE;
     DECLARE Z STATIC EXTERNAL;
     . . . . . .
     Z = Z + 1;
END UP;
```

Прежде чем начнется выполнение программы, состоящей из двух внешних процедур, произойдет выделение памяти для статических переменных. В данном случае переменной Z отводится память до того, как главная процедура PRO будет активизирована операционной системой. Когда процедура PRO начнет выполняться, переменной Z будет присвоено значение 1. Так как Z имеет в обеих процедурах атрибут EXTERNAL, она будет представлять собой одну и ту же переменную. Каждый раз при завершении процедуры UP значение Z будет увеличиваться на 1, а при возвращении управления в главную процедуру — печататься с помощью оператора PUT. Изменение значения Z в процедуре UP происходит независимо от места вызова. Например, UP может вызываться из других внешних процедур программы. Но эти внешние процедуры, если они не содержат объявления идентификатора Z с атрибутами EXTERNAL, сами не имеют доступа к значению Z.

Глава 7

ОПИСАНИЕ И ИСПОЛЬЗОВАНИЕ ПРОЦЕДУР

7.1. Процедуры-подпрограммы

Как было указано выше (§ 6.2), для сокращения объема программы целесообразно выделять многократно повторяющиеся участки стандартных вычислений и описывать их с помощью процедурных блоков. Процедура-подпрограмма представляет собой процедурный блок, в результате выполнения которого вычисляется и возвращается в вызываемую программу значение одного или нескольких переменных.

Общий формат процедуры-подпрограммы имеет вид

имя входа: PROCEDURE [(список формальных параметров)];

оператор 1;
оператор 2;
. . . . .
оператор *n*;

END [имя входа];

Активизация (т. е. начало выполнения) процедуры-подпрограммы осуществляется путем ее вызова из любой точки программы с помощью специального оператора CALL, который имеет следующий формат:

CALL имя входа [(список аргументов)];

В момент вызова процедуры управление передается на точку входа, имя которой указано в операторе CALL, и осуществляется связь между аргументами и формальными параметрами этой процедуры. Эта связь заключается в том, что процедуре передаются имена аргументов (вернее, адреса областей памяти размещения этих аргументов). Поэтому после осуществления такой связи ссылка на параметры внутри вызванной процедуры фактически означает обращение к соответствующему аргументу. И так как аргумент и параметр занимают одну и ту же область памяти, то всякое изменение значения формального параметра вызывает изменение значения соответствующего аргумента в вызывающем блоке.

Однако в некоторых случаях аргумент может и не иметь имени. Это бывает, например, когда он является константой, выражением или вызовом функции со списком аргументов. Поэтому такие аргументы не могут быть прямо переданы в вызываемую процедуру. Транслятор должен предусмотреть область памяти для таких аргументов и назначить для них некоторое внутреннее имя. Эти внутренние имена называются фиктивными аргументами. Формальные параметры процедуры связываются в этом случае с фиктивными аргументами.

Передача аргументов происходит таким образом, что список аргументов и список формальных параметров обрабатываются попарно слева направо. Следовательно, оба списка должны содержать одинаковое число элементов. Первый аргумент соответствует первому формальному параметру, второй — второму и т. д. Следует еще раз напомнить, что можно использовать не более 12 формальных параметров (учитывая дополнительные точки входа в процедуру). Так как при передаче аргументов никакого преобразования не происходит, то атрибуты их должны соответственно совпадать.

Необходимо заметить, что при создании фиктивного аргумента для константы его атрибуты определяются формой записи константы.

Например, если FUNK является процедурой-подпрограммой, в которую необходимо передать аргумент в виде двоичного числа с фиксированной точкой, то оператор

```
CALL FUNK (2);
```

вызовет ошибку, так как фиктивный аргумент, созданный для константы 2, имел бы атрибуты десятичного числа с фиксированной точкой. Таким образом, атрибуты аргумента и формального параметра будут неодинаковы. Этого можно избежать, если записать константу в виде двоичного числа (10B) или передать имя переменной:

```
I = 2; CALL FUNK (I);
```

В заключение следует заметить, что формальные параметры не могут быть объявлены ни с атрибутами области действия, ни с атрибутами класса памяти. Аргументы не могут принадлежать к базированному классу памяти.

Завершение выполняемой процедуры-подпрограммы может осуществляться тремя способами:

— с помощью оператора RETURN, который содержится во внутреннем тексте («теле») процедуры. В результате выполнения этого оператора управление будет передано в вызывающую программу на оператор, непосредственно следующий за оператором CALL, с помощью которого была вызвана данная процедура-подпрограмма.

— при достижении конца «тела» выполняемой процедуры-подпрограммы, т. е. оператора

END [имя входа].

Как и в первом случае, управление передается в вызывающую процедуру оператору, непосредственно следующему за оператором CALL;

— с помощью оператора перехода GOTO. В этом случае выполнение вызывающей программы будет продолжаться с оператора, метка которого указана в операторе GOTO.

Примеры.

1) Рассмотрим две внешние процедуры PR и P, первая из которых является вызывающим процедурным блоком для второй:

```
PR: PROC OPTIONS (MAIN);
    DCL (A, B) FIXED (5, 4),
        C FIXED (7,4);
    . . . . . .
L:  GET EDIT (A, B) (F (5, 4));
    . . . . . .
    CALL P (A, B, C, L);
    PUT EDIT (C) (SKIP, F (9,4));
    . . . . . .
    GOTO L;
    . . . . . .
END PR;
P:  PROC (E1, E2, K, M);
    DCL (E1, E2) FIXED (5,4),
        K FIXED (7,4)
        M LABEL;
    IF E1 < 0 | E2 < 0 THEN GOTO M;
    IF  E1 + E2 = 0 THEN
        DO; K = 0; RETURN; END;
    K = E1**2 + E2**2;
END P;
```

В процедуре PR вводятся значения двух переменных A и B (с помощью оператора GET). Затем с помощью оператора CALL P (A, B, C, L) вызывается внешняя процедура-подпрограмма P. При этом осуществляется связь между аргументами (A, B, C, L) и параметрами (E1 E2, K, M). Атрибуты формальных параметров объявляются с помощью оператора DECLARE, так как атрибуты по умолчанию нас в данном случае не устраивают. В результате выполнения процедуры-подпрограммы вычисляется значение G (формальный параметр K), управление передается в основную процедуру на оператор вывода PUT, который и выдает полученную величину на печать в виде десятичного числа с фиксированной точкой

В данном примере процедура-подпрограмма завершается всеми тремя возможными способами:

— при отрицательном значении хотя бы одного из двух первых аргументов (если условие A < 0| B < 0 истинно) управление возвращается в вызывающую процедуру на метку L, т. е. процедура завершается с помощью оператора GOTO;

— если сумма аргументов равна нулю, то управление передается в вызывающую процедуру с помощью оператора RETURN;

— если не соблюдается второе условие то завершение процедуры происходит естественным путем, т. е. достижением оператора END P.

2) Непосредственная обработка комплексных чисел в данном подмножестве языка ПЛ/1 невозможна. Опишем в виде процедуры с несколькими входами программу, которая выполняет сложение, вычитание и умножение комплексных величин. Каждый вход имеет три параметра — массивы из двух элементов (действительная часть мнимого числа и коэффициент при мнимой части): для первого и второго комплексного числа A и B соответственно и для результата C.

```
ADD: PROCEDURE (A, B, C);
     DECLARE A (2), B (2), C (2);
     K = 1; GOTO M1;
SUB: ENTRY (A, B, C);
     K = -1;
M1:  DO I = 1 TO 2;
        C (I) = A (I) + B (I) * K;
     END;
     RETURN;
MUL: ENTRY (A, B, C);
     C (1) = A (1) * B (1) - A (2) * B (2);
     C (2) = A (1) * B (2) + A (2) * B (1);
END ADD;
```

В данной процедуре формальные параметры A, B и C (результат выполнения процедуры) объявлены явно, как векторы с атрибутами по умолчанию DECIMAL FLOAT (6) Кроме формальных параметров, в данной процедуре использованы переменные K, I и M1, которые нужны только при выполнении ее «тела». Такие переменные называются локальными. Они локализуются в процедуре с помощью объявлений. Так переменные K и I объявлены в процедуре неявно с атрибутами по умолчанию FIXED BINARY (15), а переменная M1 — явно с атрибутом LABEL.

При вызове этой процедуры по имени основной точки входа CALL ADD (E, F, P) выполняется операция сложения двух комплексных переменных, а при обращении оператором CALL MUL (E, F, P) будет выполнена операция умножения. Значения локальным переменным присваиваются соответствующими операторами присваивания, которые должны быть включены в процедуру программистом.

Кроме того, в процедуре могут использоваться так называемые глобальные переменные, которым не надо присваивать значения в момент ее вызова. Такие переменные не входят в список формальных параметров. Они должны быть объявлены как в самой процедуре, так и во внешних по отношению к ней блоках с атрибутом класса памяти STATIC (или с атрибутом области действия EXTERNAL, который по умолчанию подразумевает атрибут STATIC). Использование глобальной переменной Z рассмотрено в последнем примере § 6.6.

7.2. Процедуры-функции

В том случае, когда в результате выполнения процедуры новое значение присваивается только одной переменной, ее можно оформить упрощенно с помощью так называемой процедуры-функции (или сокращенно функции). Отличие в описании процедуры-функции от процедуры-подпрограммы заключается в том, что един-

ственный результат ее выполнения не является формальным параметром и, следовательно, не включается в список. Кроме того, обращение к функции осуществляется с помощью идентификатора (имени) функции, который может использоваться в качестве операнда при вычислении выражений.

Описание процедуры-функции имеет следующий вид:

```
имя входа: PROCEDURE [(список формальных параметров)] [атрибуты функции];
        оператор 1;
        оператор 2;
        . . . . .
        оператор n;
        RETURN (выражение);
END [имя входа];
```

«Выражение», указанное в операторе RETURN, представляет собой результат выполнения процедуры-функции.Оно вычисляется, и его значение преобразуется к виду, определяемому «атрибутами функции», указанными в операторе PROCEDURE. Если «атрибуты функции» не указаны, то значению этого выражения присваиваются транслятором атрибуты по умолчанию в соответствии с первым символом имени входа данной процедуры. Это значение может быть арифметическим, строковым, цифровым знаковым данным или указателем.

При использовании процедуры-функции ее имя входа должно быть объявлено явно в вызывающей процедуре с помощью оператора DECLARE следующим образом:

$$\text{DECLARE имя входа } \left\{\begin{array}{l}\text{ENTRY}\\ [\text{ENTRY}]\ \text{RETURNS (атрибуты функции)}\\ [\text{атрибут области действия}]\end{array}\right\};$$

Первый вариант с ENTRY используется в том случае, когда значение выражения возвращается в вызывающую процедуру с атрибутами по умолчанию. Во втором варианте атрибут RETURNS подразумевает атрибут ENTRY, поэтому последний можно опустить.

В качестве атрибута области действия по умолчанию принимается EXTERNAL. Поэтому при объявлении внутренних к вызывающей программе процедуре-функций в операторе DECLARE должен быть указан атрибут INTERNAL.

При использовании процедур-функций необходимо иметь в виду, что во время трансляции основной программы транслятор не имеет возможности проверить функцию с тем, чтобы определить атрибуты возвращаемого ею значения. При отсутствии атрибута RETURNS в вызывающей программе транслятор может только предполагать, что атрибуты возвращаемого значения будут соответствовать атрибутам, которые объявляются по умолчанию по первой букве имени функции. Это справедливо и для случая, когда функция содержится внутри вызывающей программы. Если значение, воз-

вращаемое функцией, имеет атрибуты, отличные от тех, которые предполагает получить вызывающая программа, преобразование их не производится. Поэтому для функции, которая возвращает некоторое значение с атрибутами, не соответствующими атрибутам по умолчанию для имени функции, всегда должен быть указан атрибут RETURNS.

Активизация функции осуществляется путем ее вызова из любой точки программы с помощью идентификатора (имени) функции, за которым может следовать список аргументов. Форма обращения к функции имеет вид

идентификатор [(список аргументов)].

Особенность вызова функции заключается в том, что он осуществляется при вычислении выражения, операндом которого может быть этот идентификатор. Например, при вычислении выражения, используемого в операторе присваивания,

A = C * D + FUNK (M, N);

идентификатор FUNK (M, N) является обращением к функции с именем FUNK. Заключенный в круглые скобки список, следующий за именем функции, представляет собой список аргументов, которые передаются функции FUNK.

Завершение выполнения функции может осуществлять следующим образом:

— с помощью оператора RETURN. Его применение в функции несколько отличается от применения в процедуре-подпрограмме. Будучи выполнен в «теле» функции, он возвращает в точку вызова не только управление, но и вычисленное значение результата выполненной функции. Выражение обязательно должно присутствовать в этом операторе. Причем оно может быть только скалярным выражение ı;

— с помощью оператора GOTO. В этом случае вычисление выражения, из которого произошел вызов функции, не будет закончено, а управление будет передано оператору, метка которого указана в операторе GOTO.

Примеры.

1) Рассмотрим две внешние процедуры, описанные в предыдущем параграфе. Учитывая, что во внешнем процедурном блоке *P* вычисляется значение только одной переменной K, опишем его в форме процедуры-функции:

```
PR:  PROC OPTIONS (MAIN);
     DCL (A, B) FIXED (5,4)),
         P RETURNS (FIXED (7,4));
L:   GET EDIT (A, B) (F (5,4));
     . . . . . .
     PUT EDIT (P (A, B, L)) (SKIP, F (9,4)),
     GOTO L;
     . . .  . . . . . .
END PR;
P:   PROC (E1, E2, M) FIXED (7,4);
```

```
      DCL (E1, E2) FIXED (5,4),
          M LABEL;
      IF E1 < 0 | E2 < 0 THEN GOTO M;
   IF E1 + E2 = 0 THEN RETURN (0);
   RETURN (E1 ** 2 + E2 ** 2);
   END P;
```

Вызов функции P (A, B, L) обеспечивает вычисление выражения, а при ненормальном окончании (переменная A или B имеют отрицательное значение) происходит переход к оператору с меткой L.

Нормальное окончание вызванной функции происходит с помощью оператора RETURN, при этом возвращается в точку вызова либо нуль, либо значение выражения (A **2 + B ** 2). Атрибуты возвращаемого значения FIXED DECIMAL (7, 4) указаны в операторе PROC.

2) В некоторой программе читаются перфокарты с цифровыми данными. Необходимо описать в виде функции часть программы, в которой должна быть проверена правильность набивки. Проверяется каждая колонка перфокарты. Если в ней набит не числовой символ (что определяется его внутренним представлением), то его значение всегда будет «меньше» символа нуль. Тогда в основную программу сообщается номер колонки, в которой сделана ошибка, и дается возможность выдать сообщение об ошибке.

Программа будет иметь вид

```
FR:   PROCEDURE (X, I) BIT (1);
      DECLARE X (80) CHAR (1);
      DO I=1 TO 80;
      IF X (I) < '0' THEN RETURN ('1'B);
      END;
      RETURN ('0'B);
END FR;
```

В функции формальный параметр, являющийся управляющей переменной цикла, определяет номер колонки, в которой сделана ошибка.

В основной программе должно быть проанализировано возвращаемое значение (битовая строка единичной длины), и если оно равно 1, то имеет место неправильная пробивка и значение I-й колонки перфокарты не цифра.

7.3. Встроенные функции языка ПЛ/1

Необходимость проведения определенных типов вычислений, таких, как извлечение квадратного корня, вычисление значений логарифмов, определение абсолютных значений и др., возникает настолько часто, что для программистов были разработаны специальные средства реализации этих функций в программах. Они включают не только часто употребляемые арифметические функции, но и функции для обработки строк, массивов и другие необходимые и полезные функции, связанные со специальными средствами, предусмотренными в языке. Любую из этих функций программист может включать из операционной системы. Такое включение производится автоматически, а эти функции называются встроенными и являются неотъемлемой частью языка ПЛ/1. Это означает, что имена входов встроенных функций транслятору известны и их не нужно объявлять явно. Все встроенные функции вызываются, как процедуры-функции.

В отличие от функций, составленных программистом, которые всегда возвращают скалярное значение, имеется много встроенных функций, которые могут возвращать массив значений. Эта возможность удобна при использовании выражений над массивами. Хотя идентификаторы встроенных функций и являются ключевыми словами, они не резервируются и могут быть использованы программистом в качестве имен данных. Поэтому в блоке, где программистом объявлено ключевое слово в качестве идентификатора, встроенная функция с таким же именем не может быть вызвана. Однако имя, объявленное в качестве идентификатора переменной, можно переобъявить в любом блоке с помощью атрибута BUILTIN как имя для встроенной функции. Для встроенных функций без параметров (например, для DATE, TIME и NULL) объявление их с помощью атрибута BUILTIN обязательно.

Пример.

```
P:   PROC OPTIONS (MAIN);
     DCL SIN FIXED;
     .....
     SIN = A + B/C;
     .. ....
     PS: BEGIN;
         DCL SIN BUILTIN;
         .....
         SL=SIN (D);
         . . . . . .
     END PS;
     . . . . . .
END P;
```

В данном примере в процедуре P объявлен идентификатор SIN как десятичная переменная с фиксированной точкой. Область ее действия — вся внешняя процедура за исключением блока PS, где идентификатор SIN объявлен явно с помощью атрибута BUILTIN как встроенная функция вычисления синуса.

В соответствии с областями применения различают пять различных групп встроенных функций:

— математические;
— вспомогательные арифметические;
— для обработки строк;
— для обработки массивов;
— для специальных задач.

Математическими функциями являются:

— тригонометрические функции SIN, SIND, COS, COSD, TAN, TAND, ATAN, ATAND;
— гиперболические функции SINH, COSH, TANH, ATANH;
— функции извлечения квадратного корня, показательной функции и логарифмов SQRT, EXP, LOG, LOG10, LOG2;
— функции ошибок ERF и ERFC.

Аргументами этих функций являются данные арифметического типа с плавающей точкой. Если они представлены в другой форме, то будут (если это возможно) автоматически преобразованы к та-

кому представлению до того, как будет вызвана встроенная функция. Аргументы могут быть скалярным арифметическим выражением или именем массива.

Все математические встроенные функции возвращают значения, представленные в форме с плавающей точкой и с основанием и разрядностью, совпадающей с основанием и разрядностью аргументов.

Если аргумент — массив, то значение, возвращаемое встроенной функцией, есть массив, размер и границы которого соответствуют аргументу. При этом встроенная функция применяется к каждому элементу массива.

Рассмотрим некоторые особенности использования этих встроенных функций.

При использовании тригонометрических функций SIN (X), COS (X), TAN (X) значения аргументов задаются в радианах, а для SIND (X), COSD (X), TAND (X) — в градусах.

Функция ATAN вычисляет арктангенс заданного значения и возвращает результат в радианах. Общая форма обращения к функции

ATAN (X [, Y]).

Если указан только аргумент X, то возвращаемое значение арктангенса приводится к главному, т. е.

$$-\frac{\pi}{2} < \text{ATAN (X)} < \frac{\pi}{2}.$$

Если указаны аргументы X и Y, то возвращается значение арктангенса выражения X/Y. При этом оно определяется по следующим правилам:

— для $Y > 0$ и любого X значение равно ATAN (X/Y);

— если $X > 0$ и $Y = 0$, то значение равно $\frac{\pi}{2}$;

— если $X \geqslant 0$ и $Y < 0$, то значение равно π + ATAN (X/Y);

— если $X < 0$ и $Y = 0$, то значение равно $-\frac{\pi}{2}$;

— если $X < 0$ и $Y < 0$, то значение равно $-\pi$ + ATAN (X/Y);

— если $X = 0$ и $Y = 0$, то происходит прерывание программы.

Функция ATAND вычисляет арктангенс заданного значения и возвращает результат в градусах. Общая форма обращения к функции

ATAND (X [, Y]).

Если указан только аргумент X, то возвращаемое значение арктангенса приводится к главному значению, т. е.

$$-90° < \text{ATAND (X)} < 90°.$$

Если указаны аргументы X и Y, то возвращается значение арктангенса, полученное по формуле

ATAND (X/Y) = (180/π) * ATAN (X/Y).

Функции LOG (X), LOG10 (X) и LOG2 (X) вычисляют соответственно натуральный, десятичный и двоичный логарифмы аргумента.

Задаваемое значение аргумента должно быть больше нуля, иначе произойдет прерывание программы.

Функция EXP (X) вычисляет значение степени основания натурального логарифма, т. е. e^x.

При обращении к *функции SQRT (X)* значение аргумента X не должно быть меньше нуля, иначе произойдет прерывание программы.

Функция ERF(X) вычисляет значения интеграла вероятности

$$\mathrm{ERF(X)} = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2}\, dt,$$

а функция ERFC (X) = 1 — ERF (X).

Пример. Запишем с использованием встроенных функций вычисление следующих функций:

а) $y = \frac{1}{2} \ln \frac{1 + \sin x}{1 - \sin x}$

```
Y = 0.5 * LOG ((1Г0 + SIN (X))/(1E0 — SIN(X)));
```

б) $y = \lg \operatorname{tg} \sqrt{x^2 + z^2}$

```
Y = LOG10 (TAN (SQRT (X**2 + Z ** 2)));
```

в) $v = \frac{1}{\cos x} + \lg\left(\operatorname{tg} \frac{x}{a}\right)$

```
V = 1E0 / COS (X) + LOG10 (TAN (X/A));
```

Вспомогательные арифметические функции могут использоваться для выполнения определенных арифметических операций. К этим функциям относятся:

ABS (X) — вычисление абсолютного значения;

SIGN (X) — выделение знака;

MAX (X_1, X_2, ..., X_n) — выбор максимального значения из списка аргументов;

MIN (X_1, X_2, ..., X_n) — выбор минимального значения из списка аргументов;

FLOOR (X) — нахождение ближайшего целого, не превышающего значения аргумента;

CEIL (X) — нахождение ближайшего целого, большего или равного аргументу;

TRUNC (X) — нахождение ближайшего целого (равного FLOOR (X), если $X \geqslant 0$, и CEIL (X), если $X < 0$);

FIXED (X [, *p*[, *q*]]) — преобразование в форму с фиксированной точкой;

FLOAT (X [, *p*]) — преобразование в форму с плавающей точкой;

DECIMAL (X [, *p* [, *q*]]) — преобразование в десятичную систему счисления;

BINARY (X, [, *p* [, *q*]]) — преобразование в двоичную систему счисления;

ROUND (X, *n*) — округление числа;

PRECISION (X, [, *p* [, *q*]]) — преобразование аргумента к заданной разрядности;

MOD (X, Y) — получение положительного остатка при целочисленном делении;

ADD (X, Y, *p* [, *q*]) — вычисление суммы двух аргументов с заданной разрядностью;

MULTIPLY (X, Y, *p* [, *q*]) — вычисление произведения двух аргументов с заданной разрядностью;

DIVIDE (X, Y, *p* [, *q*]) — вычисление частного от деления двух аргументов с заданной разрядностью.

Аргументы этих функций должны быть заданы в кодированной арифметической форме. В противном случае они преобразуются в эту форму перед вызовом функции. Аргументы, которые не могут быть преобразованы в эту форму, запрещаются. Если не указано иное, аргументом этих функций может быть скалярное выражение или массив. В последнем случае значение, возвращаемое функцией, есть массив, имеющий те же изменения и границы.

Каждая функция возвращает значение в кодированной арифметической форме. Если не указывается ни основание, ни форма представления, ни разрядность, то эти данные берутся из аргумента. Рассмотрим более подробно выполнение данных функций.

Функция MAX (X_1, X_2, ..., X_n) возвращает значение наибольшего аргумента. Основание, тип и разрядность определяются в соответствии с правилами, используемыми для вычисления арифметических выражений. Но если все аргументы числа с фиксированной точкой с разрядностью (p_1, q_1), ..., (p_n, q_n), то значение функции будет с разрядностью (MIN (m, MAX, ($p_1 - q_1$, ..., $p_n - q_n$) + + MAX (q_1, ..., q_n)), MAX (q_1, ..., q_n)), где m — максимальное количество цифр (15 десятичных или 31 двоичных).

Функция MIN (X_1, X_2, .., X_n) возвращает значение наименьшего аргумента. Характеристики результата функции определяются по тем же правилам, что и для MAX.

Функция FIXED (*X* [, *p* [, *q*]]) возвращает значение аргумента X, преобразованное в форму с фиксированной точкой с разрядностью (p, q). Основание возвращаемого значения такое же, как у аргумента X. Аргументы p и q должны быть целыми десятичными константами (q может иметь знак). Если p и q опущены, то используется разрядность по умолчанию.

Встроенная функция дает программисту возможность самому указать разрядность результата преобразования в число с фиксированной точкой.

Функция FLOAT (*X* [, *p*]) возвращает значение аргумента X, преобразованное в форму с плавающей точкой с разрядностью (p) с таким же значением основания, как у аргумента X. Если аргумент p не задан, то разрядность используется по умолчанию.

Функция DECIMAL (X[, *p* [, *q*]]) возвращает значение аргумента X, преобразованное в десятичную систему счисления с разрядностью (*p*, *q*). Если аргумент X типа FLOAT, то значение *q* можно не задавать. Если *p* и *q* не заданы, то используется разрядность по умолчанию.

Функция BINARY (X [, *p* [, *q*]]) возвращает значение аргумента X, преобразованное в двоичную систему счисления с разрядностью (*p*, *q*). Если аргумент X типа FLOAT, то значение *q* можно не задавать. Если *p* и *q* не заданы, то используется разрядность по умолчанию.

Функция ROUND (X, n) возвращает значение аргумента X с округлением. В качестве аргумента может быть использовано скалярное выражение или имя массива. Параметр *n* является десятичной целой константой (значение *n* должно быть меньше количества цифр справа от десятичной (или двоичной) точки аргумента).

Если аргумент представляется с фиксированной точкой, то функция возвращает значение аргумента, округленное с *n*-го разряда справа от десятичной (или двоичной) точки.

Если аргумент представляется с плавающей точкой, то параметр *n* игнорируется. При выполнении функции крайний правый бит во внутреннем представлении аргумента устанавливается в 1, если он равен 0, а если он равен 1, то значение аргумента не изменяется.

Заметим, что округление отрицательного аргумента приводит к округлению абсолютного значения этого аргумента.

Пример. Объявлены следующие переменные:

```
DCL  A FIXED (7,5),
     (B, C) FIXED (6,4);
A = 45.44876;
B = ROUND (A,3);
C = ROUND (A,4);
```

При выполнении функции ROUND переменные B и C получат значения B = 45.450; C = 45.4488;

Функция PRECISION (X, p [, *q*]) возвращает значение аргумента X, преобразованное к заданной разрядности (*p*, *q*). Если аргумент X имеет представление с плавающей точкой, то значение *q* можно не задавать.

Пример.

```
A = 36.284;
```

В результате выполнения функции:

```
PRECISION (A,7,4)
```

возвращается значение, равное 036 2840

Функция MOD (X, Y) возвращает в качестве значения функции положительный остаток после деления X на Y. Основание и способ представления возвращаемого значения функции такие же, как у

аргументов. Разрядность значения функции при способе представления с плавающей точкой

$$p = \text{MAX}\ (p_1,\ p_2),$$

а при способе представления с фиксированной точкой

$$p = \text{MIN}\ (m,\ p_2 - q_2 + \text{MAX}\ (q_1,\ q_2)),$$
$$q = \text{MAX}\ (q_1,\ q_2),$$

где m — максимальное количество цифр (15 десятичных или 31 двоичных).

Пример.

```
A = MOD(17 5);
```

Переменная A получает значение, равное 2, т. е. положительный остаток при делении числа 17 на 5.

В операторах присваивания использованы следующие функции:

а) A = FLOOR (3.92), тогда A = 3;
   B = FLOOR (—6.71), тогда B = 7;

б) A = CELL (3.92), тогда A = 4;
   B = CELL (—6.71), тогда B = 6;

в) A = TRUNC (3.92), тогда A = 3,
   B = TRUNC (—6.71), тогда B = —6.

Функция ADD (X, Y, p [, q]) вычисляет сумму аргументов X и Y и позволяет программисту управлять разрядностью результата операции сложения.

Аргументы p и q должны быть десятичными целыми константами, которые определяют разрядность результата. Если результат определяется в форме с плавающей точкой, то аргумент q не задается.

При использовании функций DIVIDE и MULTIPLY значения аргументов и результаты вычислений подчиняются тем же правилам, что и при использовании функции ADD.

Функция для обработки строк позволяет программисту более гибко и эффективно описывать алгоритмы обработки символьной информации, а также логические алгоритмы.

В качестве аргументов этих функций могут быть использованы скалярные выражения и массивы. В последнем случае в качестве значения функции возвращается массив с размерностью и границами аргумента, т. е. функция выполняется для каждого элемента массива. К данным функциям относятся: BIT, BOOL, CHAR, HIGH, INDEX, LOW, REPEAT, SUBSTR и UNSPEC.

Две последние функции (SUBSTR и UNSPEC) могут также использоваться в качестве псевдопеременных. Псевдопеременными называются такие встроенные функции, которые могут использоваться как переменные для присваивания значения их аргументам. Псевдопеременные могут использоваться в левой части оператора присваивания, а также в списке данных оператора ввода GET.

Функция BIT, общий формат которой BIT (X [,*l*]), осуществляет преобразования аргумента X в строку битов длиной *l*. Аргументом функции может быть скалярное выражение, строка символов или битов, а также массив. Длина *l* — десятичная целая константа. Если длина *l* не указана, то результат определяется атрибутами аргумента.

Пример. Если X = 5, а *l* = 6, то функция BIT (X, *l*) возвратит значение строки битов '101000'B, т. е. десятичное число 5 преобразуется в двоичное 101B, затем в строку битов '101'B и справа добавляются двоичные нули до длины *l*. Если *l* = 2, то возвращаемое значение равно '10'B.

Функция BOOL дает возможность осуществить любую логическую операцию над строками битов. Общий формат функции

BOOL (X, Y, *op*).

Аргументы X и Y представляют собой битовые строки (при выполнении функции более короткая строка расширяется до длинной строки добавлением справа нулей). Если аргументы — не строки битов, то перед выполнением функции они преобразуются к ним. В качестве аргументов могут также использоваться массивы. Результат выполнения функции является строкой битов с длиной, равной длине большего аргумента.

Определение логической операции, производимой над каждой парой битов обоих аргументов, происходит в зависимости от третьего аргумента *op*. Этот аргумент может быть битовой, знаковой строкой или арифметическим выражением и преобразуется в битовую строку длиной 4 бита, которые указывают, какие результаты должны быть приняты для четырех возможных комбинаций битов первых двух аргументов.

Эти комбинации определяются следующим образом:

	Для бита X	Для бита Y
1-й бит *op*	0	0
2-й бит *op*	0	1
3-й бит *op*	1	0
4-й бит *op*	1	1

Таким образом, первый бит аргумента *op* указывает, какой результат будет при нулевых значениях битов первого и второго аргументов, второй бит аргумента *op* указывает, какой результат будет при нулевом значении бита аргумента X и единичном значении бита аргумента Y и т. д.

Например, для битовых строк A и B операция логического умножения может быть представлена в форме

C = BOOL (A, B,'0001'B);

а для логической операции тождества — в форме

C = BOOL (A, B,'1001'B);

Пример. При следующем объявлении:

```
DECLARE (A, B, B1, B2) BIT (5);
B = '10110'B;  A = '01010'B;
B1 = BOOL  (A,B,'1010'B);
B2 = BOOL (A,B,'0110'B);
```

переменные B1 и B2 будут иметь значения

B1 = '01001'B; B2 = '11100'B;

Функция CHAR осуществляет преобразование в строку символов длиной *l*. Формат функции

CHAR (X [,*l*]).

Аргументом функции может быть скалярное выражение, результат которого — строка битов, строка символов, цифровое знаковое данное или массив. Аргумент *l* — целая десятичная константа. Если он опущен, длина определяется значением аргумента X.

Пример. Если A = '1011'B, а *l* = 5, то возвращаемое значение функции CHAR (A, *l*) равно '1011 —'.

Функция HIGH возвращает в качестве значения функции строку знаков длиной *l* байтов. Формат функции

HIGH (*l*),

где *l* — десятичная константа без знака.

Возвращаемая строка знаков состоит из последовательности знаков, которые представлены шестнадцатиричным FF в каждом байте.

Функция INDEX указывает позицию, с которой строка Y содержится в строке X впервые. Функция имеет формат

INDEX (X, Y).

Аргументы X и Y могут быть строками битов или символов. В качестве аргументов могут использоваться также выражения (со строками битов или символов, с цифровыми знаковыми данными и кодированными арифметическими данными) или массивы. Перед вызовом функции они преобразуются в строки битов или символов. Если оба аргумента — массивы, то они должны иметь одинаковые границы. В качестве значения функции возвращается целое двоичное число, означающее позицию, с которой строка X содержит первый раз строку Y. Если строка X не содержит, то возвращается нуль.

Пример

INDEX ('FABRUARY', 'R') — возвращается значение 4;
INDEX ('10100111011'B, '11'B) — возвращается значение 6;
INDEX ('PL/I', 'I') — так как символа 'I' в строке нет, возвращается значение 0.

Функция LOW возвращает в качестве значения строку знаков длиной *l* байтов. Формат функции

LOW (*l*),

где *l* — десятичная константа без знака.

Каждый байт возвращаемого значения содержит шестнадцатиричное 00.

Функция REPEAT возвращает в качестве значения строку, которая получается m-кратным сцеплением строки аргумента X. Формат функции

REPEAT (X, m),

где X — битовая, символьная или цифровая строка, которая перед сцеплением преобразуется в битовую или символьную строку; m — десятичная константа без знака.

Результат функции — строка, содержащая аргумент X $m + 1$ раз. Например, результатом REPEAT ('TAM', 3) является строка 'TAMTAMTAMTAM'.

Функция SUBSTR выделяет из указанной строки, начиная с некоторой позиции, подстроку заданной длины. Формат функции

SUBSTR (X, p, l),

где X может быть строкой битов, символов, арифметическим двоичным данным или цифровым знаковым данным. Если аргумент является двоичным числом или цифровым знаковым данным, то перед выполнением этой функции аргумент преобразуется в строку битов или символов. Аргумент может быть массивом. В этом случае выделяется подстрока из каждого элемента массива; p указывает, с какой позиции исходной строки должна выделяться подстрока. Этот аргумент может быть также скалярным выражением, если его можно преобразовать в целое число. Если аргумент X является массивом, то p может быть также массивом с размерностью и границами X;

l указывает длину подстроки и выражается целой положительной десятичной константой.

Аргументы p и l должны выбираться так, чтобы подстрока содержалась внутри исходной строки.

Примеры

1)
```
DECLARE X CHARACTER (11), Y CHARACTER (6);
   X = ' PROGRESSION';
   Y = SUBSTR (X, 4,4);
   SUBSTR (X, 9,3) ='';
```

В результате выполнения операторов присваивания Y получит значение 'GRES__', а X — значение PROGRESS ___. В последнем случае функция SUBSTR использовалась как псевдопеременная.

2) Во фрагменте программы

```
DECLARE A BIT (7);
   . . . . . .
   . . . . . .
IF SUBSTR (A, 4, 1) = '1' THEN GOTO M1;
   . . . . .
M1: . . . . . .
```

функция SUBSTR используется для организации разветвления вычислительного процесса.

3. Каждый символ некоторой символьной строки длиной 8 байтов необходимо разместить в одномерном массиве. Это реализуется с помощью функции SUBSTR следующим образом:

```
DECLARE Z1 CHARACTER (8),
        Z2 (8) CHARACTER (1);
DO I = 1 TO 8;
        Z2 (I) = SUBSTR (Z1, I, 1);
END;
```

Функция UNSPEC возвращает в качестве своего значения строку битов, которая является внутренним представлением аргумента X. Форма функции

UNSPEC (X),

где X может быть арифметическим данным, строкой символов, цифровым знаковым данным, значением указателя или массивом. Аргумент не может быть строкой битов.

Длина строки битов соответствует числу битов во внутреннем представлении X и определяется атрибутами аргумента следующим образом:

— если X имеет атрибуты FIXED BINARY, то длина строки битов равна 32;

— если X имеет атрибуты FIXED DEC (p, q), то длина строки определяется по формуле 8 * FLOOR ((p + 2)/2);

— если X имеет атрибуты FLOAT BIN (p), то длина строки равна 32, когда $p \leqslant 21$, или 64, когда $p > 21$;

— если X имеет атрибуты FLOAT DEC (p), то длина строки равна 32, когда $p \leqslant 6$, или 64, когда $p > 6$;

— если X — строка символов, то ее длина не может превышать 8 символов, т. е. возвращаемая строка битов не превышает 64 битов;

— если X — указатель, то длина строки равна 32. Однако внутреннее представление указателя определяется крайними правыми 24 битами.

Пример. При объявленных атрибутах

```
DECLARE A FIXED (3,1), B BIT (16),
        Z CHARACTER (2), F FIXED (5);
A = 46.5;
B = UNSPEC (A);
Z = UNSPEC (A);
F = UNSPEC (A);
```

получаем следующее значение переменных:

B = '0100 0110 0101 1100'B — внутреннее представление значения A
    4   6   5   +

в двоичной форме.

Z = '01' — возвращаемое функцией внутреннее представление преобразуется в строку символов и присваивается переменной Z. Ввиду того, что

переменная объявлена с атрибутом разрядности (2), берутся два первых символа, а остальные отсекаются.

F = 18012 — сначала внутреннее представление A преобразуется в двоичное число с фиксированной точкой, затем — в десятичное число с фиксированной точкой (в нашем случае оно равно 18012)

Встроенные функции для обработки массивов позволяют программисту реализовать некоторые логические и арифметические операции над элементами массива. Все эти функции требуют, чтобы в качестве аргументов были использованы массивы. В результате выполнения функции возвращается скалярное значение. К этим функциям относятся ALL, ANY, PROD и SUM.

Функция ALL (X) осуществляет логическое умножение всех элементов массива. Каждый элемент массива X перед выполнением функции преобразуется в строку битов и выравнивается до наибольшего элемента присоединением справа нулей.

Значение, возвращаемое в результате выполнения функции, есть строка битов, длина которой равна длине наибольшего элемента массива X. Каждый *i*-й бит возвращаемой строки равен 1, если *i*-е биты всех элементов X равны 1, в противном случае *i*-й бит результата равен 0.

Пример. Имеется массив B и переменная C, объявленные следующим образом:

```
DECLARE B (4) BIT (5), C BIT (5);
```

Элементы массива B имеют значения

```
B (1) = '01101'B;  B (2) = '11001'B;  B (3) = '01001'B;  B (4) =
                                                    = '01000'B;
```

При выполнении оператора присваивания

```
C = ALL (B);
```

переменная C получит значение '01000'B, т. е. реализуется операция логического умножения.

Функция ANY (X) осуществляет логическое сложение всех элементов массива.

Результат такой же, как и для функции ALL, за исключением того, что *i*-й бит возвращаемой строки равен 1, если имеется *i*-й бит, равный 1, хотя бы для одного элемента аргумента X; в противном случае *i*-й бит результата равен 0.

Для предыдущего примера

```
C = ANY (B);
```

переменная C получит значение '11101'B, т. е. реализуется операция логического сложения.

Функция PROD (X) возвращает скалярное значение, равное произведению всех элементов массива X. Элементы массива X либо должны иметь тип FLOAT, либо перед выполнением функции они преобразуются к такому типу.

Результат всегда представляется в форме с плавающей точкой. Основание и разрядность возвращаемого значения такие же, как у элементов аргумента.

Функция SUM (X) возвращает скалярное значение, равное сумме всех элементов массива X.

Встроенные функции для специальных задач. Функции, рассматриваемые в этом подразделе, можно в зависимости от их назначения разбить на следующие три группы:

— DATE и TIME, которые позволяют получить во время выполнения программы текущие дату и время. Первая дает возможность реализовать печать даты обрабатываемого документа, а вторая — определить время выполнения любого оператора или группы операторов программы, что позволяет при отладке анализировать полученные временные характеристики и написать программу наилучшим образом;

— ADDR и NULL, которые используются для присвоения значения переменной типа указателя, обеспечивающего выделение области памяти для базированных переменных (о них будет подробно сказано в гл. 9);

— STRING, которая используется при обработке структур.

Функция DATE в качестве значения возвращает строку символов длиной 6 байтов, которая содержит текущую дату. При этом строка имеет следующую форму:

'jjmmdd',

где *jj* — год; *mm* — месяц; *dd* — день.

Как видно из формата, значение выдаваемой строки не соответствует общепринятому написанию даты. Для печати в этом случае можно использовать следующий оператор вывода PUT:

```
PUT EDIT (SUBSTR (DATE,5,2)||'.'|| SUBSTR (DATE,3,2)
    ||'.'||SUBSTR (DATE, 1, 2)) (SKIP, A);
```

Этот оператор обеспечивает вывод текущей даты в форме *dd.mm.jj*.

Функция TIME в качестве значения возвращает строку символов длиной 9 байтов, которая содержит текущее время. Строка имеет следующую форму:

'hhmmsttt',

где *hh* — часы; *mm* — минуты; *ss* — секунды; *ttt* — миллисекунды.

Данную функцию можно использовать для распечатки текущего момента времени и по разности двух моментов получить время выполнения одного или нескольких операторов.

Наиболее целесообразно использовать данную функцию при отладке отдельных участков программы, описание которых можно осуществить несколькими способами.

Пример

```
DECLARE (Z, Z1) CHARACTER (9);
Z = TIME;
DO I = 1 TO 100;
  PUT EDIT (SQRT (I)) (SKIP, X (20), F (9, 6));
END;
Z1=TIME;
PUT EDIT (Z, Z1) (SKIP, X (10), 2A);
```

В данном фрагменте программы вычисляется и печатается время, которое необходимо для выполнения DO-группы, включая вызов функции TIME и присваивание возвращаемого значения.

Функция ADDR (X) возвращает значение, которое является адресом переменной, заданной в качестве аргумента.

Аргумент X может быть скалярной переменной, индексированной переменной, элементом структуры, массивом или структурой.

Пример

```
DECLARE B FIXED (5,2),
        A1 POINTER;
```

Если переменная A1 (указатель) должна получить значение, которое является адресом переменной B, то необходимо использовать встроенную функцию ADDR следующим образом:

```
A1 = ADDR (B);
```

Функция NULL используется для инициализации указателя при использовании базированной переменной.

Ввиду того, что указатель (POINTER) является переменной управления программой, мы не имеем права присваивать ему арифметическое значение нуля. Поэтому присвоение значения функции NULL, которое не является адресом какой-либо переменной в памяти машины, как бы «обнуляет» указатель.

Функция STRING (X) в качестве значения возвращает символьную строку, которая получается сцеплением всех элементов аргумента X.

Аргумент X должен быть структурой с атрибутом UNALIGNED, которая содержит только строки символов и (или) цифровые знаковые данные.

Пример.

```
DCL 1 ST UNALIGNED,
        2 A CHAR (12),
        2 B CHAR (8),
        2 C CHAR (4),
      D CHAR (24);
A = 'С — ПРАЗДНИКОМ'; B = ' — ПЕРВОГО';
C = ' — МАЯ';
  D = STRING (ST);
```

В результате выполнения функции STRING (ST) будет выдана и присвоена переменной D строка символов

'С — ПРАЗДНИКОМ — ПЕРВОГО — МАЯ'.

Глава 8

СОЗДАНИЕ И ОБРАБОТКА НАБОРОВ ДАННЫХ

8.1. Наборы данных и их идентификация

Всякая программа, написанная на языке ПЛ/1, предназначена для обработки некоторого объема информации. Если в ней выполняются инженерные расчеты, то этой информацией могут быть массивы чисел. В экономических расчетах это, как правило, — содержание определенных документов, включающих как числовые, так и символьные данные.

Любая информация, представляющая собой организованную совокупность логически связанных между собой данных, может быть названа набором данных. В современных цифровых ЭВМ для хранения этой информации используются различные внешние носители, такие, как перфокарты, магнитные ленты, магнитные диски и др.

Несмотря на большое разнообразие носители информации имеют много общих характеристик. Поэтому любой внешний носитель информации называют одним общим термином — том. В качестве тома может использоваться, например, бобина магнитной ленты или пакет магнитных дисков. При использовании в качестве внешнего носителя магнитной ленты или магнитных дисков, которые допускают перезапись информации (т. е. уничтожение старой или запись новой информации), идентификация наборов данных приобретает большое значение.

Обычно на одном томе может храниться несколько наборов данных. Для выделения из множества наборов данных определенного набора и получения информации о возможности использования определенного участка внешней памяти для записи нового набора в операционной системе имеются определенные средства сохранения записанных наборов данных.

Для идентификации наборов данных, расположенных на магнитной ленте или магнитных дисках, используются так называемые метки. В метке для каждого набора данных содержатся сведения о его имени, местоположении, организации, а также другая управляющая информация. При обращении к конкретному набору данных операционная система с помощью специальной подпрограммы

производит проверку информации, расположенной в метке, с тем, чтобы определить, нужный ли набор данных используется. Если используется не тот набор данных, который нам нужен, система после выдачи сообщения прекращает дальнейшую его обработку.

8.2. Объявление файлов

Каждый конкретный набор данных существует независимо от определенной программы. Однако он всегда согласуется с особенностями операционной системы. Это является большим преимуществом, так как конкретные наборы данных могут создаваться и обрабатываться операционной системой в различных программах. На практике это означает, например, что набор данных, созданный в программе, которая написана на языке ПЛ/1, может быть обработан в программе, написанной на языке ФОРТРАН.

Для того чтобы обратиться к конкретному набору данных из программы, написанной на языке ПЛ/1, необходимо в этой программе описать так называемый логический информационный файл. Этот логический файл представляет собой символическое обозначение (имя) реально существующего набора данных, записи которого обрабатываются в данной программе. Причем все данные для операционной системы задаются при описании, как последовательность свойств и признаков в форме атрибутов и режимов данного логического файла. Окончательная связь файла, описанного в программе, с набором данных, расположенным на конкретном внешнем устройстве, устанавливается с помощью операторов программы УПРАВЛЕНИЯ ЗАДАНИЯМИ.

Логический файл в программе на языке ПЛ/1 объявляется с помощью оператора:

DECLARE идентификатор FILE атрибуты;

Идентификатор представляет имя файла, которое является внешним именем, поэтому его максимальная длина не должна превышать шести символов.

Атрибуты, которые следуют в произвольном порядке за атрибутом FILE, дополняют описание файла определенными свойствами, а также признаками, которые связаны с особенностями машины.

Среди атрибутов файла различают альтернативные и аддитивные. При объявлении файла программист обязательно должен использовать по одному атрибуту из каждой группы альтернативных атрибутов. Аддитивные атрибуты присоединяются по мере надобности.

К альтернативным атрибутам файла относятся:

— атрибуты типа передачи данных;
— атрибуты функции файла;
— атрибуты доступа;
— атрибуты буферизации.

Атрибуты типа передачи данных. Тип передачи данных задается с помощью атрибутов RECORD или STREAM.

Как известно, всякая передача данных с внешнего устройства в основную память и наоборот происходит блоками. После передачи данных записи находятся в области ввода — вывода (буфере). Далее возможны две принципиально разные формы обработки записей, находящихся в буфере. При этом порядок обработки как при вводе, так и при выводе аналогичен.

Первой формой обработки является передача, ориентированная на записи, которая описывается атрибутом RECORD. В этом случае записи из буфера переносятся без всякого изменения в память, отведенную для переменной, которая была указана в операторе ввода, или обрабатываются непосредственно в буфере.

Второй формой обработки является передача, ориентированная на поток, которая вводится атрибутом STREAM. Если во время передачи, ориентированной на записи, записи всегда рассматриваются как единое целое, то во время обработки, ориентированной на поток, записи, находящиеся в буфере, рассматриваются как непрерывный поток символов. Символы подвергаются обработке в той последовательности, в какой они находятся в буфере. Чаще всего во время этой передачи происходит преобразование данных.

Среди этих атрибутов STREAM является атрибутом по умолчанию.

Атрибуты функции файла. Функция файла задается с помощью атрибутов INPUT, OUTPUT и UPDATE.

В объявлении файла, как правило, должно быть задано точное указание относительно функции файла. Для этого используется один из перечисленных выше атрибутов. Однако для файлов с последовательной организацией и атрибутами RECORD, UNBUFFERED функции INPUT или OUTPUT можно указывать при их открытии.

INPUT (ВВОД) означает, что данные могут передаваться только с внешнего носителя в основную память. OUTPUT (ВЫВОД) означает, что данные могут передаваться только из основной памяти на внешний носитель. Попытка записать данные в файл с атрибутом INPUT ведет к ошибке (прерыванию программы). Точно также невозможно прочитать данные из файла, описанного с атрибутом OUTPUT.

Однако на практике часто бывает необходимо читать записи из набора данных в основную память, изменять их там и записывать их обратно. Кроме того, может возникнуть потребность добавить к уже существующему набору данных отдельные записи. Эта функция файла, при которой записи приводятся в новое состояние, задается с помощью атрибута функции файла UPDATE. Можно считать, что при использовании этого атрибута файл получает свойства файлов INPUT и OUTPUT.

Атрибуты метода доступа определяют порядок доступа к записям набора данных.

При использовании файлов, описанных с атрибутом способа обработки STREAM, т. е. в случае передачи данных, ориентированной на поток, к записям набора данных возможен только последовательный доступ. Только в случае файлов с атрибутом обработки RECORD (передача данных, ориентированная на запись) можно задавать любой метод доступа. Последовательный метод доступа к записям набора данных указывается с помощью атрибута SEQUENTIAL, прямой метод доступа — с помощью атрибута DIRECT. Атрибутом по умолчанию является SEQUENTIAL.

Атрибуты буферизации. К этим атрибутам относятся BUFFERED и UNBUFFERED.

Буфер — это промежуточная область в основной памяти, которую может создавать операционная система и которая используется в каждой передаче данных. При вводе информации с внешнего носителя записи набора данных прежде всего попадают в буфер (если он выделен системой), откуда они передаются в область памяти переменных или обрабатываются внутри самого буфера. При выделении буферов система работает особенно эффективно, так как одновременно с операциями ввода — вывода происходит процесс обработки записей.

Для файлов с атрибутом BUFFERED в основной памяти отводится по умолчанию одна область в качестве буфера. Размер (длина) буфера равен длине одного блока. Когда внутри блока содержится большое число записей, буфер может иметь значительный размер. Программист всегда должен иметь это в виду.

В случае блокированных записей нужно стремиться установить оптимальное соотношение между рациональным использованием памяти внешнего устройства и увеличением потребности в основной памяти для буфера.

Атрибуты буферизации могут быть указаны только для файлов RECORD SEQUENTIAL. Для файлов с атрибутом STREAM использование атрибутов буферизации запрещено. Запрещение обусловливается тем, что эти файлы всегда обрабатываются только с буферами и нет необходимости указывать при объявлении файла атрибут буферизации.

Атрибут UNBUFFERED может быть указан только для файлов на МЛ или МД. Причем на МЛ он указывается только в том случае, когда задан режим NOLABEL.

Атрибут BUFFERED указывает, что записи, передаваемые из набора данных в область памяти переменной, указанной в программе, должны проходить через буфер.

Атрибут UNBUFFERED означает, что записи при передаче не должны проходить через буфер, т. е. они передаются непосредственно в область основной памяти, отведенную некоторой переменной, или из области памяти переменной в набор данных на внешний носитель. В наборе данных, связанном с файлом UNBUFFERED, записи должны быть неблокированными.

Если для файла, в котором использование атрибутов буферизации возможно, они не заданы, то по умолчанию подразумевается атрибут BUFFERED.

Аддитивные атрибуты позволяют описать специальные свойства и особенности файлов. Они используются только в том случае, когда необходимо реализовать какие-то дополнительные возможности при обработке записей наборов данных.

Атрибут PRINT указывает, что описываемый файл предназначается для выдачи данных на печать. Этот атрибут может применяться только для вывода потоком. Он предполагает атрибуты STREAM и OUTPUT, т. е. их можно опустить, если указан атрибут PRINT.

Атрибут KEYED указывается для тех файлов, которые связаны с наборами данных, расположенных на внешних устройствах прямого доступа (МД), и записи которых распознаются с помощью ключей. Следует отметить, что атрибут KEYED предполагается при указании альтернативного атрибута DIRECT, т. е. если указан DIRECT, то атрибут KEYED можно не указывать.

Атрибут BACKWARDS можно использовать для описания файлов, связанных с наборами данных, расположенных на МЛ и имеющих атрибут типа передачи RECORD и атрибуты SEQUENTIAL и INPUT. Он используется в том случае, если необходимо указать операционной системе, что записи этого набора данных будут обрабатываться в обратной последовательности, т. е. сначала будет читаться последняя запись, затем предпоследняя и так далее до первой записи набора. Использование этого атрибута предусматривает обязательное включение режима LEAVE в список режимов атрибута ENVIRONMENT.

Атрибут ENVIRONMENT (сокращенно ENV) позволяет программисту задавать информацию об организации и характеристиках набора данных, связанного с объявленным в программе файлом. Вся информация указывается с помощью режимов этого атрибута. Формат атрибута

ENVIRONMENT (список режимов)

Режимы в списке могут указываться в любой последовательности и должны отделяться друг от друга хотя бы одним пробелом. Если предыдущий режим заканчивается закрывающей скобкой, пробел может отсутствовать. В целях большей наглядности программы и единообразия записи рекомендуется придерживаться некоторой определенной последовательности.

Формат записи. Записи набора данных, обрабатываемые в программе, могут иметь следующие форматы:

— фиксированной длины (формат F);
— переменной длины (формат V);
— неопределенной длины (формат U).

Записи могут быть блокированы. При этом длина записи (или блока) устанавливается в байтах. Размер блока должен указывать-

ся в каждом из приведенных режимов. Если размер записи не установлен, то предполагается, что они не блокированы, т. е. каждый блок содержит одну запись.

Записи фиксированной длины имеют следующий формат:

F (длина блока [, длина записи])

Данный формат записей может применяться для всех видов файлов. Отношение длины блока к длине записи всегда должно быть целым числом, которое представляет собой коэффициент блокирования. Блокирование записей при выводе и деблокирование при вводе производится автоматически и не требует никаких специальных действий от программиста.

Блокированные записи фиксированной длины можно использовать только для файлов с атрибутом RECORD (как на магнитной ленте, так и на магнитных дисках). Причем это должен быть файл с последовательной или индексно-последовательной организацией.

Записи переменной длины имеют формат

V (максимальная длина блока)

При использовании этого режима блок содержит столько записей, сколько позволяет максимальная длина блока. Если записи набора данных имеют этот формат, то в каждом блоке содержится информация о длине блока (четырехбайтовое поле) и о длине каждой записи блока (также четырехбайтовое поле). Несмотря на то что длина подсчитывается и вносится в блок системой автоматически при создании набора данных, программист при указании формата записи должен учитывать и байты для этой информации.

Записи неопределенной длины имеют формат

U (максимальная длина блока)

При использовании записей этого формата каждый блок содержит только одну запись. Вся ответственность за обработку записей данного формата ложится на программиста. Если в запись включается программистом информация о ее длине, то он должен сам позаботиться о выделении и обработке этой информации.

Режим MEDIUM используется для связи файла с конкретным устройством, на котором располагается набор данных. Он имеет следующий формат:

MEDIUM (имя логического устройства, шифр устройства)

Имя логического устройства — это имя, связанное с файлом. Оно указывается в форме SYSxxx, где xxx может принимать одно из следующих значений:

IPT — системное устройство ввода (обычно устройство ввода с перфокарт);

LST — системное устройство вывода (обычно устройство вывода на печать);

PCH — системное устройство вывода (обычно устройство вывода на перфокарты);

000 — 221 — логическое устройство программиста.

Связь логических устройств с самими устройствами ввода — вывода устанавливается либо во время первоначальной загрузки системы, либо с помощью оператора ASSGN программы УПРАВЛЕНИЯ ЗАДАНИЯМИ.

Шифр устройства — это четырехзначное число, которое задает номер устройства ввода — вывода. Они задаются во время генерации операционной системы. Возможные шифры таких устройств приведены в табл. 8.1.

Таблица 8.1

Тип устройства	Шифры устройства
Устройство ввода с перфокарт	6012, 6013, 6014, 6016
Устройство вывода на перфокарты	7010, 7012, 7013, 7014
Устройство печати	7030, 7031, 7032, 7033, 7034, 7035, 7038
Накопитель на магнитных лентах	5010, 5012, 5014, 5015, 5016, 5017, 5019, 5021
Накопитель на магнитных дисках	5052, 5054, 5056, 5058

При объявлении файла режим MEDIUM должен быть указан обязательно, имя логического устройства должно быть присвоено конкретному устройству ввода — вывода или автоматически, или с помощью оператора ASSGN. Если в режиме MEDIUM были допущены ошибки, то во время трансляции будет выдано соответствующее сообщение. Если же ошибка произошла при назначении устройств ввода — вывода, то во время открытия соответствующего файла выполнение задания будет прекращено.

Режим BUFFERS (n) указывает количество промежуточных буферов (областей ввода или вывода). Использование буферов позволяет совместить передачу данных с обработкой их в процессоре, чем сокращается общее время выполнения программы. Число буферов устанавливается параметром *n* и может быть равным 1 или 2. Если режим не указан, то по умолчанию будет выделяться один буфер.

Данный режим может указываться как для файлов с атрибутом STREAM, записи которых всегда обрабатываются с буферами, так и для файлов с атрибутами RECORD и BUFFERED.

Режимы способов организации набора данных. Способ организации набора данных определяет, как данные располагаются на внешнем носителе и передаются с него в основную память (или из нее). Записи могут храниться и выбираться из набора данных или последовательно (в соответствии с их физическим размещением на носителе), или прямым методом с помощью ключей. Эти способы хранения и извлечения данных определяют в языке ПЛ/1 три способа организации наборов данных:

— последовательный (CONSECUTIVE);
— прямой (REGIONAL);
— индексно-последовательный (INDEXED).

Если в атрибуте ENVIRONMENT не указан ни один из этих режимов, то набор данных по умолчанию рассматривается как последовательно организованный, т. е. подразумевается режим CONSECUTIVE.

В наборе данных CONSECUTIVE записи организованы только на основе их последовательного физического расположения (так, как они организованы, например, на магнитной ленте). Для определения их положения в наборе данных ключи не используются. Записи выбираются (читаются) только последовательно друг за другом. Файл, связанный с последовательным набором данных, может иметь атрибуты SEQUENTIAL и STREAM. При использовании этого набора записи могут быть в любом из трех форматов F, V или U. Однако форматы V и U можно использовать только для ввода — вывода записей файла с атрибутом RECORD. При этом нужно учесть, что ленточные наборы данных с такими форматами не могут читаться в обратном направлении (т. е. не могут иметь атрибут BACKWARDS).

При региональной организации набора данных (режим REGIONAL) программист может управлять физическим расположением записей в наборе. Каждая запись включает дополнительный идентификатор (признак) в виде ключа, с помощью которого она может быть помещена в соответствующее место набора данных. При помощи этого же ключа любая запись может быть прочитана из набора данных. Для набора данных REGIONAL можно использовать только устройства прямого доступа (накопители на магнитных дисках). Набор данных должен содержать неблокированные записи формата F.

Существует две разновидности наборов данных региональной организации:

— REGIONAL (1), где каждая запись размещается в отдельной области, которые нумеруются по порядку от начала набора данных, начиная с нуля. Номер области и является ключом, с помощью которого может быть помещена (или найдена) любая запись;

— REGIONAL(3), где областью является дорожка магнитного диска, на которой располагается несколько записей. Области нумеруются системой по порядку от начала набора данных, начиная с нуля. Для идентификации записи на дорожке используется некоторый логический ключ. Номер соответствующей дорожки и логический ключ записи образуют так называемый внесенный ключ, с помощью которого запись может быть найдена в наборе данных.

При объявлении любого файла, связанного с набором данных региональной организации, должен быть указан атрибут KEYED. Кроме того, для набора данных с организацией REGIONAL (3) должен быть указан режим KEYLENGTH, который задает длину внесенного ключа.

При индексно-последовательной организации набора данных (режим INDEXED) размещение каждой записи также определяется ключом, который представляет собой строку символов. Записи в наборе упорядочиваются в логически возрастающей последовательности своих ключей согласно последовательности символов ЕС ЭВМ (код ДКОИ).

При создании набора данных операционная система формирует индексы (в виде записей особой структуры), которые содержат информацию о размещении записей на дорожках и цилиндрах магнитного диска, выделенных для набора данных. Эти индексы позволяют операционной системе повысить эффективность поиска записей в наборе данных.

Для набора данных INDEXED можно использовать только устройства памяти прямого доступа. Записи в нем должны иметь формат F и могут блокироваться. Набор данных с индексно-последовательной организацией может создаваться только последовательно. После того как набор данных создан, с ним можно связать файл, который имеет атрибуты INPUT или UPDATE и SEQUENTIAL или DIRECT. Если файл имеет атрибут DIRECT, то записи могут быть извлечены, добавлены или заменены в произвольном порядке.

Режимы CTLASA и CTLYES позволяют в случае вывода записей на печать или на перфоратор управлять переходом к новой строке или странице для устройства печати или выбором приемного кармана для устройства перфорации. Эти режимы могут использоваться только для файлов, имеющих атрибут RECORD. Управляющие символы для каждого из этих режимов приведены в приложении 2.

Режимы для наборов данных на магнитной ленте. *Режим LEAVE* позволяет отменить операцию перемотки магнитной ленты к началу набора данных, которая обычно происходит при открытии или закрытии файла, связанного с таким набором. Этот режим следует устанавливать в том случае, когда сразу же после создания набора данных можно его обрабатывать в обратной последовательности (используя атрибут BACKWARDS).

Режим NOLABEL позволяет программисту создавать или обрабатывать так называемый рабочий набор данных на МЛ. Этот режим указывает, что для набора данных не нужна обработка меток, т. е. не нужно записывать метки в набор данных при открытии файла с атрибутом OUTPUT или проверять метки при открытии файла с атрибутом INPUT.

Режим NOTAPEMK используется для автоматической записи маркера ленты при открытии рабочих файлов с атрибутом OUTPUT, т. е. он может устанавливаться только совместно с режимом NOLABEL. Его нельзя использовать для файлов, которые обрабатываются без буферов.

Режим VERIFY проверки правильности внесения записей в набор данных на магнитном диске позволяет осуществить проверку путем чтения и сравнения каждой записи, помещаемой в набор данных, т. е. после выполнения каждой операции вывода.

Режимы для наборов данных с прямым методом доступа. *Режим KEYLENGTH* (*n*) позволяет программисту указать длину ключа записи в байтах для наборов данных с организацией REGIONAL (3) и INDEXED.

Для файлов REGIONAL (3) ключ задается в форме строки символов длиной n ($9 \leqslant n \leqslant 255$). При этом восемь цифровых символов определяют относительный номер дорожки и хотя бы один символ должен идентифицировать положение записи на дорожке.

Для файлов INDEXED ключ, задаваемый также в виде строки символов длиной n ($1 \leqslant n \leqslant 255$), определяет конкретную запись в наборе данных.

Режим EXTENTNUMBER (*n*) указывает количество участков на магнитном диске, которые используются для наборов данных с организацией REGIONAL и INDEXED.

Для набора данных с региональной организацией (атрибуты REGIONAL (1) или REGIONAL (3)) размещение на нескольких отдельных участках может быть вызвано отсутствием незанятой непрерывной области на пакете дисков. Тогда приходится размещать записи набора данных на свободных дорожках разных цилиндров. Последовательность отдельных дорожек, используемых для размещения набора данных, называется участком. Если в этом случае набор данных расположится менее чем на четырех участках, указание данного режима не обязательно (хотя каждый из этих участков должен быть описан с помощью отдельного оператора EXTENT программы УПРАВЛЕНИЯ ЗАДАНИЯМИ). Количество таких участков может лежать в пределах $1 \leqslant n \leqslant 255$.

Для индексно-последовательного набора данных этот режим указывается обязательно. Значение n должно в этом случае включать все участки основной области данных, участок индекса цилиндров и главного индекса (которые должны быть смежными) и все участки независимой области переполнения. Индексы цилиндров и главный индекс рассматриваются как один участок несмотря на то, что каждый из них требует отдельного оператора EXTENT.

Минимальное количество участков, которое может быть указано для набора данных INDEXED, равно двум: один участок для основной области данных и один — для индекса цилиндров (т. е. $2 \leqslant n \leqslant 255$).

Специальные режимы для индексно-последовательных файлов. *Режим OFLTRACKS* (*n*) используется программистом, чтобы указать операционной системе количество дорожек на каждом цилиндре, которые должны быть выделены для области переполнения. Причем число n может изменяться в интервале $0 \leqslant n \leqslant 8$.

Режим KEYLOG (*n*) определяет положение поля ключа внутри каждой записи (при использовании блокированных записей). Значение n указывает позицию (в байтах) внутри записи, с которой начинается ключ записи. Длина ключа устанавливается в режиме KEYLENGTH. При отсутствии данного режима по умолчанию принимается режим KEYLOG (1).

Режим INDEXMULTIPLE указывается в том случае, когда программист хочет потребовать у операционной системы создания главного индекса. Применение этого режима позволяет сократить время обработки очень больших индексно-последовательных файлов. Его целесообразно указывать, когда область памяти для индекса цилиндров занимает более трех дорожек.

Режим INDEXAREA (*n*) указывает системе, что индекс цилиндров объявляемого файла должен полностью или частично считываться в основную память. В этом случае при чтении или обновлении набора данных поиск в индексе цилиндров будет выполняться в основной памяти, что позволяет значительно сократить время обработки записей набора данных. Этот режим может указываться только для файлов, имеющих атрибуты DIRECT и INPUT (или UPDATE). Параметр *n* указывает объем выделяемой области основной памяти ($0 < n \leqslant 32$ К).

Режим ADDBUFF (*n*) используется программистом, чтобы указать системе, что добавление записей при обновлении набора данных должно происходить в основной памяти, а не на внешнем устройстве. В этом случае несколько блоков набора данных считываются в память, происходит необходимое добавление записей и обновленный блок записывается в набор данных. Этот способ обновления позволяет уменьшить затраты времени на обработку, так как значительно уменьшается количество операций ввода — вывода. Данный режим может использоваться для файлов с атрибутами DIRECT UPDATE.

Параметр *n* указывает длину области основной памяти в байтах. Его величина должна удовлетворять следующему условию:

$$m\ (40 + \text{ДБ} + \text{ДК}) + 24 \leqslant n \leqslant 32\text{К},$$

где ДБ — длина блока; ДК — длина ключа; *m* — максимальное количество блоков, которые должны быть одновременно прочитаны в основную память (как правило, *m* равно количеству блоков, которые помещаются на одной дорожке основной области данных).

В приложении 3 даны обобщенные таблицы, в которых указывается, какие атрибуты и режимы должны обязательно использоваться при объявлении различных файлов и какие принимаются по умолчанию.

Примеры.

1) В программе обрабатываются 80-байтовые записи набора данных, которые размещаются на перфокартах и вводятся потоком с системного устройства. Имя файла, связанного с этим набором данных, CARD. Объявление файла:

```
DECLARE CARD FILE INPUT STREAM SEQUENTIAL ENVIRONMENT
        (F (80) MEDIUM (SYSIPT, 6012) CONSECUTIVE);
```

Используя принцип умолчания и возможные сокращения ключевых слов, этот файл можно объявить так:

```
DCL CARD FILE INPUT ENV (F (80) MEDIUM (SYSIPT, 6012));
```

2) Необходимо вывести в набор данных на магнитную ленту 100-байтовые записи, которые блокируются по 20 записей. Имя логического устройства программиста—SYS011. Передача данных осуществляется с двумя буферами. Имя файла—TAPE.

```
DECLARE TAPE FILE RECORD SEQUENTIAL
             OUTPUT BUFFERED ENVIRONMENT
             (F (2000, 100) MEDIUM (SYS011, 5012)
             BUFFERS (2) CONSECUTIVE);
```

Учитывая возможные сокращения ключевых слов и принцип умолчания, можно объявить файл так:

```
DCL TAPE FILE RECORD OUTPUT
         ENV (F (2000, 100) MEDIUM (SYS011, 5012)
         BUFFERS (2));
```

3) Имеется индексно-последовательный набор данных, в котором находятся записи длиной по 200 байтов. Длина ключа равна 12 символам. Для обработки записей на каждом цилиндре отведено по три дорожки для области переполнения. Набор данных должен обрабатываться прямым методом доступа (обработка заключается в корректировке записей и добавлении новых). С целью более эффективной обработки записей целесообразно, чтобы индекс цилиндров записывался в основную память. Имя файла, связанного с этим набором данных, — SKLAD, а имя логического устройства — SYS012.

Объявление файла в программе

```
DCL SKLAD FILE RECORD UPDATE DIRECT
          KEYED ENV (F (200) MEDIUM(SYS012, 5056)
          INDEXED KEYLENGTH (12)
EXTENTNUMBER (2)
    OFLTRACKS (3) INDEXAREA (3));
```

8.3. Открытие и закрытие файлов

Прежде чем записи набора данных будут обработаны в некоторой программе, операционная система должна выполнить определенные действия по подготовке процесса ввода — вывода. К этим действиям относится, например, поиск и обработка меток набора данных. Процесс такой подготовки называется открытием файла. При открытии файла устанавливается связь между набором данных (расположенным на внешнем носителе) и логическим файлом (объявленным в программе.)

Для выполнения действий по открытию файла используется оператор OPEN:

OPEN FILE (имя файла) [INPUT или OUTPUT] [PAGESIZE (выражение)]
[, FILE (имя файла) [INPUT или OUTPUT] [PAGESIZE (выражение)]]...;

С помощью одного оператора OPEN можно открыть несколько файлов, но не более шести. Необходимо отметить, что открытие файла выполняется специальным модулем, который загружается из библиотеки во время выполнения оператора OPEN. Поэтому для уменьшения времени выполнения этих действий целесообразно

в операторе указать имена всех файлов, обрабатываемых в данной программе.

Все файлы с атрибутом RECORD должны быть открыты явно, т. е. с помощью оператора OPEN. Для файлов с атрибутом STREAM этот оператор может быть опущен, тогда файл открывается автоматически (неявно) при выполнении первого оператора GET или PUT для этого файла. Неявное открытие равносильно тому, как если бы перед первым оператором GET или PUT для данного файла выполнился оператор OPEN.

Открытие уже открытого файла не вызывает никаких действий. Новый режим PAGESIZE в этом случае игнорируется. Для файла, который должен быть открыт явно, оператор OPEN должен выполняться до того, как для этого файла будет выполнен любой из операторов ввода — вывода

Ранее было указано, что в операторе DECLARE для файла должен быть указан один из атрибутов функции файла (INPUT, OUTPUT или UPDATE). Там же было отмечено, что для некоторых файлов функцию INPUT или OUTPUT можно устанавливать при открытии файла. Этими файлами являются файлы с атрибутами RECORD, SEQUENTIAL и UNBUFFERED. При этом один и тот же атрибут не должен указываться одновременно и в операторе DECLARE, и в операторе OPEN. Кроме того, не должно быть несоответствия между атрибутами, указанными при объявлении файла, и атрибутами, присоединенными к объявленым атрибутам в результате явного открытия файла. Например, будет противоречие, если в операторе DECLARE указан атрибут BACKWARDS, а в операторе OPEN — атрибут OUTPUT. Так как атрибуты BACKWARDS и OUTPUT несовместимы, то при трансляции программы будет выдано соответствующее сообщение об ошибке.

Режим PAGESIZE разрешается использовать только для файлов с атрибутами STREAM и PRINT. Выражение, стоящее в этом режиме, вычисляется, и результат преобразуется в двоичное число стандартной разрядности, которое обозначает количество печатаемых строк на странице. Это позволяет независимо от стандартного числа строк (которое, как правило, равно 64) указать желаемое количество строк на страницу печати. Заметим, что максимальное количество строк не должно превышать 255.

После окончания обработки записей файла его необходимо закрыть с помощью оператора CLOSE. После закрытия файла связь между ним и набором данных разрывается. Кроме того, при выполнении этого оператора отменяется атрибут функции файла (INPUT или OUTPUT), если он был установлен в операторе OPEN. Если необходимо, атрибут функции файла может быть снова присоединен при выполнении следующего оператора OPEN для данного файла.

Формат оператора CLOSE

CLOSE FILE (имя файла) [, FILE (имя файла)] ...;

Повторное закрытие файла или закрытие неоткрытого файла не вызывает никаких действий. Если оператор CLOSE для некоторого файла отсутствует, то закрытие файла происходит при завершении программы, в которой он был открыт.

Примеры.

1)
```
DCL SPRAVKA FILE RECORD INPUT ENV (F (140)
      MEDIUM (SYS009, 5056));
DCL DOCL FILE PRINT ENV (F (120) MEDIUM (SYSLST,
7030));
.....
OPEN FILE (SPRAVKA), FILE (DOCL) PAGESIZE (40);
....
CLOSE FILE (SPRAVKA);
CLOSE FILE (DOCL);
```

В данном примере файлы SPRAVKA и DOCL открываются одним оператором OPEN, а закрываются каждый отдельно. Для файла DOCL с помощью режима PAGESIZE указывается, что на одну страницу будет выводиться 40 строк.

2)
```
DCL BANK FILE RECORD UNBUFFERED
      ENV (F (100) MEDIUM (SYS019,5012) NOLABEL);
.....
OPEN FILE (BANK) OUTPUT;
.....
/* СОЗДАНИЕ НАБОРА ДАННЫХ */
CLOSE FILE (BANK);
OPEN FILE (BANK) INPUT;
/* ЧТЕНИЕ ЗАПИСЕЙ НАБОРА ДАННЫХ */
CLOSE FILE (BANK);
```

Этот пример демонстрирует открытие и закрытие файла на магнитной ленте, для которого атрибуты INPUT и OUTPUT указываются в операторе OPEN. Такая возможность позволяет без переобъявления файла в операторе DECLARE создавать и читать записи файла в одной и той же программе.

8.4. Ввод и вывод данных, ориентированный на поток

Как указывалось в § 8.2, в языке ПЛ/1 имеются две возможности передачи данных: передача, ориентированная на поток, и передача, ориентированная на записи.

При передаче данных, ориентированной на поток, набор данных рассматривается как непрерывная строка (поток) символов. Этот поток символов с помощью оператора GET вводится в основную память (буфер ввода) отдельными записями, размер которых определяется при объявлении файла в атрибуте ENVIRONMENT.

В момент передачи в буфере ввода производится первое преобразование значений вводимых переменных из символьного формата во внутреннее представление. Это преобразование осуществляется в соответствии с форматами данных, указанных в операторе ввода. При передаче переменных из буфера ввода в область памяти может

производиться повторное преобразование в соответствии с атрибутами этих переменных, объявленных в операторе DECLARE.

При выводе данных с помощью оператора PUT процесс предварительного преобразования их значений из внутреннего представления в символьную форму происходит в буфере вывода на основании форматов, указанных программистом.

При передаче данных, ориентированной на поток, наборы данных должны иметь последовательную организацию и содержать неблокированные записи фиксированной длины. Кроме того, следует напомнить, что файлы с атрибутом STREAM всегда обрабатываются с буфером, причем для них не указывается атрибут BUFFERED. При такой передаче данных автоматически создаются так называемые скрытые буферы, через которые и происходит обмен данных между внешним устройством и областью основной памяти, связанной с каждой переменной, используемой в программе. Скрытые буферы — это одна или две области в основной памяти (в соответствии с параметром режима BUFFERS) вполне определенной длины, которые указываются с помощью режима формата записи в атрибуте ENVIRONMENT. В случае применения двух буферов они располагаются непосредственно друг за другом. В дисковой операционной системе имеются два способа потоко-ориентированной передачи:

— передача, управляемая списком;

— передача, управляемая редактированием.

Для каждого из этих способов передачи в операторах ввода и вывода используется следующая информация:

— имя файла, связанное с конкретным набором данных, из которого данные должны быть считаны или в который они должны быть записаны;

— список данных, которые передаются во время ввода и вывода;

— список форматов, в котором указывается формат каждого передаваемого данного.

В некоторых случаях не вся указанная информация используется в операторах ввода — вывода.

Оператор ввода GET имеет следующий формат:

GET [{FILE (имя файла) / STRING (переменная типа строки символов)}] спецификация данных;

Оператор вывода PUT имеет формат:

— для файлов с атрибутами STREAM OUTPUT

PUT [{FILE (имя файла) / STRING (переменная типа строки символов)}] спецификация данных;

— для файлов печати (атрибут PRINT)

PUT [FILE (имя файла)] [{PAGE [LINE (выражение)] / SKIP [(выражение)] / LINE (выражение)}]

[спецификация данных];

Когда указан один из управляющих режимов, «спецификацию данных» для файлов печати можно не задавать. Спецификация данных в зависимости от способа передачи данных может иметь одну из следующих форм:

— при передаче данных, управляемой списком,

LIST (список данных)

— при передаче данных, управляемой редактированием,

EDIT (список данных) (список форматов) [(список данных) (список форматов)]...

Режим FILE устанавливает имя файла, к которому происходит обращение в операторах ввода — вывода. Если этот режим не указан, то будет происходить обращение к стандартному файлу, который открывается всегда неявно.

Для стандартного вводного файла по умолчанию принимаются следующие атрибуты и режимы: STREAM INPUT ENVIRONMENT (F (80) MEDIUM (SYSIPT, 6012) CONSECUTIVE BUFFERS (1)). Для стандартного выводного файла приняты: STREAM OUTPUT ENVIRONMENT (F(120) MEDIUM (SYSLST,7030) CONSECUTIVE BUFFERS (1)).

Режим STRING позволяет использовать операторы ввода — вывода для внутренней передачи данных. Подробно этот режим будет рассмотрен в § 8.8.

Режим PAGE указывает, что вывод на печать должен осуществляться с новой страницы.

Режим LINE указывает, что печать должна начинаться со строки, указанной выражением в круглых скобках. Предварительно выражение вычисляется и преобразуется в целое число. Если режимы PAGE и LINE используются в одном операторе, то режим PAGE выполняется первым. Например, оператор

PUT FILE (REF) PAGE LINE (34) LIST (P, Q, R);

вызывает печать значений переменных P, Q и R на новой странице, начиная со строки 34.

Режим SKIP вызывает переход к новой строке. Выражение в этом режиме указывает количество строк, на которое нужно продвинуться вперед. Предварительно выражение вычисляется и преобразуется в целое число. Однако это число может иметь значение только 0, 1, 2 и 3. Если выражение в этом режиме опущено, то принимается значение SKIP (1). При указании SKIP (0) вывод на печать будет призводиться на только что напечатанной строке.

Спецификация данных в операторах GET и PUT задает элементы данных, которые должны быть переданы этими операторами. Список данных состоит из отдельных элементов, разделенных между собой запятыми.

При вводе элементом списка может быть:
— скалярная переменная;
— переменная типа массива;
— переменная типа структуры;
— псевдопеременная;
— спецификация повторения.

При выводе элементом списка кроме этого может быть и скалярное выражение. Скалярная переменная может быть простой переменной, индексированной переменной или уточненным именем.

Переменная типа массива или структуры представляет в списке данных столько элементов, сколько их объявлено для массива или структуры в операторе DECLARE. В качестве псевдопеременной используется SUBSTR, которая представляет значение вводимой или выводимой подстроки.

Формат спецификации повторения в качестве элемента списка данных имеет вид

(элемент [, элемент]... DO переменная-спецификация
[, спецификация] ...)

В свою очередь, спецификация описывается следующим образом:

выражение 1 $\begin{bmatrix} \text{BY выражение 2 [TO выражение 3]} \\ \text{TO выражение 3 [BY выражение 2]} \end{bmatrix}$ [WHILE (выражение 4)]

Элемент в спецификации повторения может быть таким же, как элемент списка данных.

Все выражения, входящие в спецификацию, являются скалярными. «Выражение 1» задает начальное значение управляющей переменной. «Выражение 2» — это приращение, которое добавляется к значению управляющей переменной после каждого повторения элементов спецификации. «Выражение 3» представляет конечное значение управляющей переменной. «Выражение 4» задает дополнительное условие для управления количеством повторений.

Спецификация повторения заключается в круглые скобки и представляет собой один элемент списка данных. Спецификации повторения могут вкладываться друг в друга, т. е. каждый элемент в списке может быть спецификацией повторения. В этом случае элемент списка заключается в скобки и имеет вид

((элемент DO переменная = спецификация)
DO переменная = спецификация)

Пример. Матрица A объявлена следующим образом:

```
DCL A (5,5) FIXED;
```

Необходимо ввести списком значения второго и третьего столбца матрицы. В этом случае оператор ввода будет иметь формат

```
GET LIST (((A (I, J) DO I = 1 TO 5) DO J = 2 TO 3));
```

При выполнении оператора ввод элементов массива A осуществляется в следующем порядке: A (2,1), A (2,2), A (2,3), A (2,4), A (2,5), A (3,1) A (3,2), A (3,3), A (3,4), A (3,5).

Этот оператор эквивалентен двумя вложенным DO-группам:

```
DO J = 2 TO 3;
   DO I=1 TO 5;
   GET LIST (A (I,J));
   END;
END;
```

Передача элементов списка данных осуществляется в следующем порядке: A (2,1), A (2,2), A (2,3), A (2,4), A (2,5), A (3,1).. A (3,5)

Если элемент списка данных является массивом, то элементы массива передаются в порядке возрастания правого индекса.

Пример

```
DCL B (2,5) FLOAT;
```

Оператор вывода имеет вид

```
PUT EDIT (B) (список форматов);
```

В данном случае будет следующий порядок вывода элементов массива: B (1,1), B (1,2), B (1,3), B (1,4), B (1,5), B (2,1), B (2,2), B (2, 3), B (2,4), B(2 ,5);

Если элемент списка данных — структура, то ее элементы вводятся в порядке, указанном в объявлении структуры.

Пример.

```
DCL 1 ST,
      2 A CHAR (15),
      2 B (5) FIXED;
```

Оператор ввода

```
GET LIST (ST);
```

Ввод будет осуществлен в следующем порядке: ST.A, ST.B (1) ST.B (2), ST.B (3), ST.B (4), ST.B (5).

Если внутри списка данных, который используется в операторе ввода, управляемом списком или редактированием, некоторая переменная получает значение, то ее значение может использоваться при вводе последующих элементов списка.

Пример.

```
GET LIST (N, (X (I) DO I=1 TO N), J, SUBSTR (NAME, J,4));
```

При выполнении данного оператора значения переменных будут вводиться в следующем порядке:

— получает значение переменная N;

— вводятся элементы массива, как указано в спецификации повторения X(1), X(2), X(3), ..., X(N). Для определения количества вводимых значений элементов массива используется введенное значение переменной N;

— получает значение переменная J;

— строковой переменной NAME присваивается значение подстроки длиной в 4 символа, начиная с позиции, указанной введенным значением переменной J.

8.5. Передача данных, управляемая списком

Передача, управляемая списком, позволяет программисту задавать переменные, которым присваиваются значения, или значения которых передаются на внешний носитель, не указывая формата данных.

Спецификация данных для передачи, управляемой списком, имеет следующий формат:

LIST (список данных)

При вводе или выводе данные в потоке имеют одну из форм:

а) [+ | —] десятичная арифметическая константа;

б) символьная константа;

в) битовая константа.

При вводе каждый элемент данных в потоке должен быть отделен от другого одним или несколькими пробелами или запятой, слева и справа от которой может быть любое количество пробелов. В потоке может содержаться пустое поле, которое обозначается с помощью одной или двух запятых, разделенных любым количеством пробелов. Пустое поле указывает, что значение соответствующего элемента в списке данных остается прежним, т. е. ему нового значения не присваивается.

Передача списка констант при вводе прекращается после окончания списка вводимых данных или файла. В первом случае начало ввода для следующего оператора GET определяется символом, стоящим за первым пробелом, или запятой, указанной за последним переданным элементом данного.

Если данное представляет собой символьную константу, окружающие ее кавычки при вводе удаляются и заключенные в них символы интерпретируются как символьная строка и присваиваются соответствующей переменной, указанной в списке данных. Если данное является битовой константой, то удаляются кавычки и символ B, а остальные символы представляются как битовая строка.

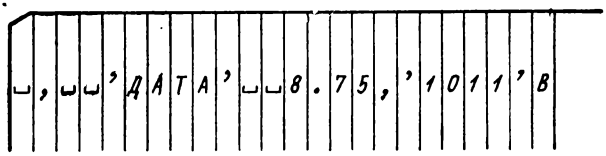
Если данное в потоке — десятичная арифметическая константа, то сначала она преобразуется во внутреннюю форму представления с основанием, способом представления и разрядностью, которые определяются способом записи этой константы. Вторичное преобразование значения данного происходит при передаче его в область памяти переменной в соответствии с атрибутами в операторе DECLARE.

Пример

Объявлены переменные

```
DCL A FIXED,
    B CHAR (6),
    C FLOAT
    D BIT (8);
```

На перфокарте набита следующая последовательность символов:



Какие значения будут присвоены переменным при выполнении оператора ввода

GET LIST (A, B, C, D);

В буфере ввода переменным будут присвоены следующие значения: B = 'ДАТА'; C = 8.75; D = '1011'B. Переменной A значение не присваивается.

При передаче значений в область памяти переменных они будут преобразованы в соответствии с объявлением в операторе DECLARE. Таким образом им будет присвоена

B = 'ДАТА␣ ␣'; C = 8.75000E + 00; D = '10110000'B.

При выводе данные, которые нужно передать, определяются списком данных. Представление данного на внешнем носителе зависит от его значения и аргументов. В выходном потоке элементы данных отделяются друг от друга пробелами. Для файлов с атрибутом PRINT элементы данных размещаются на листе печати, начиная с определенных позиций листа, а именно с позиций 1, 25, 49, 73, 97 и 121. Длина поля (w), которое элемент данных занимает в выходном потоке, зависит от разрядности и значения соответствующего элемента списка данных. Элементы арифметических данных записываются на внешний носитель в форме десятичной константы, которая может иметь знак.

Десятичное данное с фиксированной точкой преобразуется к разрядности (p, q). Если q удовлетворяет условию $0 \leqslant q \leqslant p$, то $w = p + 3$ и $d = q$ (где d — количество дробных разрядов). Если $q > p$, то $w = p + 3 + n$ (где n — количество цифр, которое необходимо для представления q).

Представление на внешнем носителе, которое будет иметь при выводе значение, равное нулю, определяется следующими правилами:

— если $q = 0$, то выводится нуль, которому предшествует $p + 2$ пробела и $w = p + 3$

$$\underbrace{\text{␣␣ ... ␣}}_{p+2}0;$$

— если $0 < q \leqslant p$, т. е. число имеет дробную часть, то в выводной поток помещаются $p + 2 - q$ пробелов, точка и q нулей, $w = p + 3$

$$\underbrace{\text{␣␣...␣}}_{p+2-q}.\underbrace{00\ldots0}_{q}$$

Подавление нулей выполняется влево от поля, и, если значение данного меньше нуля, знак минус непосредственно предшествует первой значащей цифре или нулю слева от десятичной точки, если значение дробное.

Десятичное данное с плавающей точкой преобразуется в соответствии с правилами для элемента формата E (l, d, s), который будет рассмотрен в § 8.9. Для такого преобразования $w = p + 6$, $d = p - 1$, $s = p$. Значение, равное нулю, представляется на внешнем носителе в следующей форме:

$$\underbrace{\_0.00...0}_{p-1}\_\mathrm{E} \pm 00$$

Двоичное данное с фиксированной точкой перед выводом преобразуется в десятичную форму представления с фиксированной точкой.

Двоичное данное с плавающей точкой перед выводом преобразуется в десятичную форму представления с плавающей точкой.

Цифровые знаковые данные преобразуются к арифметическому типу с десятичным основанием. Формат данного на внешнем носителе и длина поля определяются спецификацией шаблона.

Данное типа строки битов перед выводом преобразуется в символьную форму: бит 0 в символ 0, бит 1 в символ 1. Полученная строка символов заключается в кавычки и за ней помещается буква B.

Символьное данное выводится на внешний носитель в том виде, как оно представлено в основной памяти. Длина поля, занимаемого данным в потоке, равна длине строки.

8.6. Передача данных, управляемая редактированием

Передача данных, управляемая редактированием, позволяет программисту указать их формат на внешнем носителе. Спецификация данных в этом случае имеет следующий общий формат:

EDIT (список данных) (список форматов)
[(список данных) (список форматов)]...

При вводе, управляемом редактированием (так же, как и при выводе), для каждого списка данных должен быть указан список форматов, который задает, каким образом должны интерпретироваться символы, содержащиеся во входном потоке. Каждый список форматов имеет следующее общее строение:

$$\left(\left\{\begin{matrix}\text{элемент}\\ n\ \text{элемент}\\ n\ (\text{список форматов})\end{matrix}\right\}\left[\begin{matrix},\text{элемент}\\ ,n\ \text{элемент}\\ ,n\ (\text{список форматов})\end{matrix}\right]\ \ldots\right)$$

Из этого представления следует, что список форматов может быть составлен из нескольких разделенных запятыми элементов списка

форматов, что в одном списке форматов может содержаться другой список форматов и что с помощью коэффициента повторения возможно многократное использование одного элемента списка форматов или одного списка форматов.

Следует отметить, что при записи коэффициента повторения не должно быть пробелов между ним и элементом формата. Максимальный уровень вложенности списков формата равен пяти. При этом список, который содержит все другие списки элементов формата, имеет уровень вложенности, равный единице.

В операторе ввода GET могут использоваться:

— элемент формата управления;

— элементы формата данных.

Элемент формата управления X позволяет при вводе пропустить один или несколько символов вводимого потока, т. е. с помощью элемента формата управления вида X (*l*) можно пропустить *l* символов ($1 \leqslant l \leqslant 255$).

Элемент формата данных F, общая форма которого

F (*l* [, *d* [, *p*]]),

означает, что *l* символов из потока должны интерпретироваться как десятичное число с фиксированной точкой ($1 \leqslant l \leqslant 32$). Этот элемент формата может быть использован для ввода данных, описанных с атрибутом PICTURE и представляющих собой десятичные данные с фиксированной точкой.

Данные, которые читаются с элементом формата F, должны иметь следующее внешнее представление:

[␣...][+ или -][цифра...][•][цифра...][␣...]

l символов

Если данные, прочитанные в соответствии с элементом формата F, имеют внешнее представление, отличное от указанного, то возникает прерывание программы (исключительная ситуация CONVERSION). Если последовательность символов содержит только пробелы, то соответствующему элементу списка данных присваивается нуль.

В знаковой последовательности могут содержаться знак числа и десятичная точка. Если последовательность знаков не содержит десятичную точку, то с помощью параметра *d* указывается, что число обрабатывается так, как если бы десятичная точка стояла перед последними *d* цифрами. Если же десятичная точка встречается в последовательности знаков, ее положение отменяет спецификацию *d*. Масштабный множитель *p* изменяет значение, которое получается из строки знаков при умножении на 10^p. Из трех параметров *l*, *d*, *p* только масштабный множитель может быть задан со знаком.

Примеры
1)

Внешнее представление строки символов	Формат	Значение вводимого данного
␣165 или 165␣	F (4,1)	16,5
—17␣␣	F (5,4)	—0.0017
251473	F (6,4)	25.1473
	F (6,8)	0.00251473
431921	F (6, 0, 2)	43192100
	F (6, 0, — 4)	43.1921
	F (6, 3, — 2)	4.31921

2) На перфокарте набита последовательность символов:

— ␣ 210 — 630020 + 51

С помощью следующих операторов ввода переменным будут присвоены значения:

а) GET EDIT (I, J, K) (F(5,2), F (7), F (3,2));
I = 2.1; J = — 630020; K = 0,51;

б) GET EDIT ((M (I) DO I = 1 TO 6)) (F (3,2), F (2,0), F (3,1), F (4,3), F (2,1), F (1,4));
M (1) = 0.02; M (2) = 10; M (3) = — 6.3; M (4) = 0.02; M (5) = = 0.5; M (6) = 0.0001;

в) GET EDIT (I, J, K, L, M) (F(4,0, — 7), F(1), F (5,2), F(2,1), F(3, 0, 4));
I = 0.0000021; J = 0; K = — 63; L = 2 ; M = 510000;

3) На перфокарте набита следующая последовательность символов:

7.102— —12.360 — 379.9

С помощью операторов ввода GET переменным будут присвоены следующие значения:

а) GET EDIT ((X (I) DO I = 1 TO 6))
(F (3), F (4,2,5), F (4,3), F (2,5), F (3,1), F (3,0, —3));
X (1) = 7.1; X (2) = 2000; X (3) = 12.3; X (4) = .0006; X (5) = = — 3.7; X (6) = 0.0099;

б) GET EDIT (A, B, C, D, E, F)
(F (2.0), F (3,2), F (6), F (2,2, — 3), F (4,3), F (2));
A = 7; B = 1 .02; C = 12.3; D = 0.0006; E = — 0.379; F = 0.9;

в) GET EDIT (((Z (I, J) DO I = 1 TO 3) DO J = 1,2))
(F (5,2), F (1), F (3,3), F (4,2), F (2, 0, 2), F (4, 0));
Z (1,1) = 7.102; Z (2,1) = 0; Z (3,1) = 0.012;
Z (1,2) = 0.36; Z (2,2) = — 300; Z (3,2) = 79.9;

4) Некоторые измерения, производимые в определенные сутки через каждые два часа, заносятся в таблицу с указанием даты и шестизначного номера измерения.

Дата	Номер измерения	Время измерения и его значение			
		0 ч	2 ч	4 ч	6 ч
29.10.73	837592	125.32	132.78	135.52	128.15
29.10.73	837594	126.18	138.12	130.30	125.00

Данные будут набиты на перфокарту в следующем порядке:

Колонки карты	Содержимое колонок
1—6	Дата
7—10	Пробелы
11—16	Номер измерения
17—20	Пробелы
21—25	Измерение 1
26—80	Измерения 2—12

Для первого дня последовательность символов на карте такова:

281073 ␣ ␣ ␣ ␣ 837592 ␣ ␣ ␣ ␣ 12532132781355212815

Пусть в программе обрабатываются данные: дата — DAY, номер измерения — NUMBER, значение измерения — SENSE. Их объявление будет иметь вид

```
DCL DAY PIC'(6)9',
    NUMBER PIC'(6)9',
    SENSE (12) FIXED (5,2);
```

В этом случае оператор ввода запишется так:

```
GET EDIT (DAY, NUMBER, SENSE)
  (F (6), X (4), F (6), X (4), 12F (5,2));
```

Можно в одном списке форматов объединить два элемента формата, т. е. записать

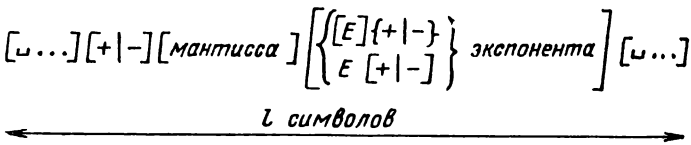
```
GET EDIT (DAY, NUMBER, SENSE)
  (2 (F (6), X (4)), 12F (5,2));
```

Элемент формата данных E, общая форма которого в списке форматов оператора ввода имеет вид

E (*l*, *d*),

означает, что группа из *l* символов (байтов) интерпретируется как десятичное число с плавающей точкой.

Данные, которые читаются с элементом формата E, должны иметь следующую внешнюю форму:



Если во внешнем представлении отсутствуют буква E и экспонента, то подразумевается, что экспонента равна нулю.

Если вводимые данные имеют представление, отличное от указанного, то происходит прерывание программы (исключительная си-

туация CONVERSION). Однако если вводимое значение меньше или равно 2^{-260} ($\sim 5{,}4 \cdot 10^{-78}$) или больше 2^{252} ($\sim 7{,}2 \cdot 10^{75}$) или вся последовательность символов состоит из пробелов, то прерывания программы не происходит. Во всех этих случаях значение данного принимается равным нулю.

Параметром d указывается предполагаемое положение десятичной точки перед последними d цифрами мантиссы. Этот параметр должен быть указан в любом случае, причем должно соблюдаться условие $d \leqslant l$ и $d \leqslant 16$. Однако если в мантиссе указывается десятичная точка, то информация, содержащаяся в параметре d, игнорируется.

Примеры.

1)

Внешнее представление строки символов	Формат	Значение вводимого данного
— 12.3E+01 —	E (10, 2)	123*
— —+33+02 — —	E (10, 1)	330
	E (10, 3)	3.3
1348—3 —	E (7, 0)	1.348
13E14	E (5,1)	$1.3.10^{14}$

* Параметр d игнорируется, так как в мантиссе содержится десятичная точка.

2) Некоторая перфокарта содержит следующую последовательность символов:

7.46E — 01 — 8.41 — 2.274E + 02

С помощью оператора ввода

```
GET EDIT (A, B, C, D) (E (8, 0), E (7,2), F (3,3)  E (5  0));
```

переменные A, B, C, D получат значения

A = 0 746; B = — 0.0841; G = 0.27; D = 400.

Элемент формата данных A, который в списке форматов оператора ввода имеет форму A (l), означает, что последовательность из l символов (байтов) должна рассматриваться как строка символов. При использовании этого элемента формата преобразования данных не производится. Количество символов (параметр l), предназначенных для передачи, должно указываться обязательно. Оно может не совпадать с длиной переменной, объявленной в операторе DECLARE. Если l меньше длины объявленного данного, то они дополняются справа пробелами, а если больше, последние «лишние» символы отсекаются без каких-либо сообщений.

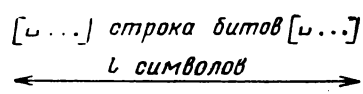
Пример.

Рассмотрим приведенную выше задачу относительно обработки измерений. Если предположить, что внутри программы переменные DAY и NUMBER не участвуют в вычислениях, то можно рассматривать эти данные, как строки символов, и записать:

```
DECLARE DAY CHARACTER (6),
        NUMBER CHARACTER (6),
        SENSE (12) FIXED (5,2);
GET EDIT (DAY, NUMBER, SENSE) (2 (A(6),
   X (4)), 12F(5,2));
```

Элемент формата данных B, который в списке форматов оператора ввода имеет форму B (*l*), означает, что последовательность из *l* символов (байтов) должна рассматриваться как строка битов ($1 \leqslant l \leqslant 64$). Значение *l* должно указываться всегда.

Данные, которые читаются с элементом формата B, должны иметь следующую внешнюю форму:



причем «строка битов» может состоять только из символов 0 и 1. Если последовательность символов содержит символы, отличные от 1 и 0, происходит прерывание программы (исключительная ситуация CONVERSION).

Пример.

1) Задана последовательность символов

— — — 10100 —

Элемент формата B (9). При чтении значение должно быть присвоено переменной, являющейся строкой битов и описанной следующим образом:

```
DECLARE MIT BIT (12);
        GET EDIT (MIT) (B(9));
```

Результат ввода

```
MIT = '101000000000'B;
```

2) На перфокарте пробита последовательность символов:

626—**XYZ**11012.

Некоторая структура ST объявлена оператором

```
DCL1 ST,
     2 (A FIXED(4,1), B CHAR (4),
       C BIT (2), D FIXED(3));
```

При использовании оператора ввода

```
GET EDIT (ST) (F (4,2), X (2), A (3), X (2), B (3), X (1), F (1));
```

элементы структуры получат следующие значения:

A = 6.2; B = ' XYZ—'; C = '11'B; D = 2.

Следует заметить, что при вводе элемента структуры A согласно формату F (4,2) в скрытый буфер ввёлось значение 6.26, но оно было в дальнейшем усечено в соответствии с атрибутом FIXED (4 1). То же произошло с элементом структуры C.

Значение элемента B (введенное по формату A (3) как строка символов 'XYZ'), наоборот, было расширено с помощью пробела до объявленной величины (атрибут переменной CHAR (4))

В списке данных оператора PUT могут находиться все те элементы списка данных, которые могут встретиться в операторе GET. Кроме того, в качестве элемента списка данных допустимо скалярное выражение и, следовательно, константа.

Правила построения списка форматов для оператора PUT не отличаются от таких же правил для оператора GET. Существует также разделение элементов списка форматов на элементы формата данных и управляющие элементы формата. Почти не имея внешних отличий, элементы формата данных, которые могут встретиться в списке форматов оператора PUT, все же отличаются по своему действию от элементов списка форматов оператора GET.

Так, например, при использовании управляющего элемента формата X (l) в выводной поток вставляется l пробелов ($1 \leqslant l \leqslant 256$).

Введен элемент формата положения столбцов COLUMN, который имеет общую форму

COLUMN (n),

где $1 \leqslant n \leqslant 256$.

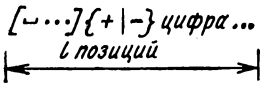
При использовании элемента формата COLUMN заполнение буфера вывода продолжается с позиции, определенной параметром n. До появления в текущей строке требуемой позиции, определенной с помощью параметра n, в выводной поток вставляются пробелы. Если n-я позиция в буфере вывода уже заполнена, процесс заполнения буфера завершается, содержимое буфера выводится и заполнение буфера начинается снова с позиции n. В этом случае элемент формата COLUMN определяет переход к новой строке. Элемент формата COLUMN обрабатывается только в том случае, если после него будет использоваться элемент формата данного.

Элемент формата данных F. При использовании элемента формата данных F, общая форма которого F (l [, d [, p]]), внутренние арифметические данные предварительно обрабатываются как десятичные числа с фиксированной точкой; причем в выводной поток, если необходимо, вставляется знак минус и десятичная точка. Параметр p указывает, что полученное после преобразования десятичное число с фиксированной точкой перед выводом умножается на 10^p. Значения l, d и p должны быть целыми десятичными константами, причем параметр p может иметь знак. Действия параметров l и d рассмотрим для следующих случаев.

Случай 1.

Элемент формата используется в виде F (l), причем количество знаковых позиций l достаточно велико, чтобы поместить все целые значащие цифры выводимого числа и знак минус. Тогда имеющиеся

в нашем распоряжении l знаковых позиций используются таким образом:



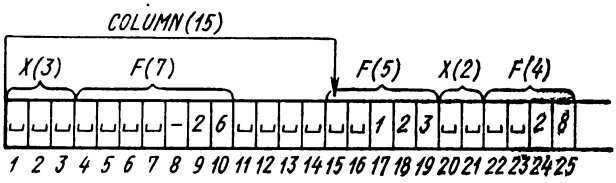
Пример.

A = — 25.734; B = + 123; C = 28.375;

Оператором вывода

```
PUT EDIT (A, B, C) (SKIP, X (3), F (7),
    COLUMN (15), F (5) X (2), F (4));
```

информация будет выводиться в следующем виде:



Переменная A выведена со значением — 26, так как при выводе происходит округление первой цифры, стоящей перед десятичной точкой.

Использование формата COLUMN (15) указывает, что переменная B должна выводиться с 15-й позиции. Значения всех переменных выводятся как целые десятичные числа, причем знак плюс заменяется пробелом.

Случай 2.

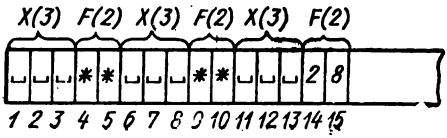
Используется элемент формата F (l) или F (l, d), причем количество знаковых позиций l недостаточно для размещения всех целых значащих цифр выводимого числа или знака минус. В этом случае возникает прерывание программы (исключительная ситуация SIZE). При выключенном состоянии этой ситуации на печать выводится l символов *.

Пример.

Если значения переменных A, B, C такие же, как и в предыдущем примере, то результат оператора

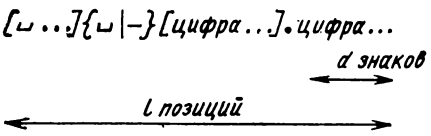
```
PUT EDIT (A, B, C) (SKIP, 3 (X (3), F (2)));
```

будет иметь вид



Случай 3.

Используется элемент формата F (l, d), причем количество позиций после десятичной точки, заданной параметром d, меньше или равно количеству позиций в дробной части числа, заданного в операторе DECLARE (т. е. $q \geqslant d$). Тогда имеющиеся в нашем распоряжении l знаковых позиций используются следующим образом:



Таким образом, если элемент формата задан так, что в выводном потоке недостаточно позиций для всех цифр дробной части данного, то после десятичной точки выводится только d цифр, причем последняя из этих цифр округляется.

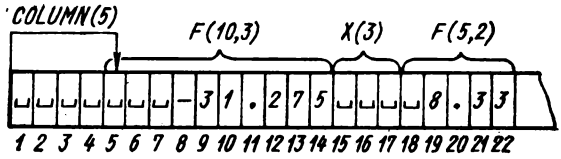
Пример. Переменные A и B объявлены оператором

```
DECLARE A FIXED (7,3),
        B FIXED (5,4);
```

Значения переменных: A = — 31.275; B = 8.3275;
Результат действия оператора

```
PUT EDIT (A, B) (SKIP, COLUMN (5), F (10, 3), X (3), F (5,2));
```

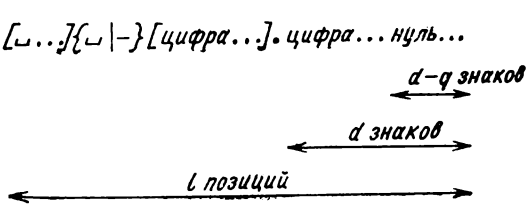
имеет вид



Заметим, что при заполнении позиций дробной части числа B произошло округление и на печать выдано число 8.33.

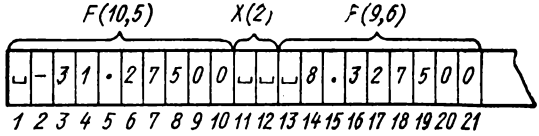
Случай 4.

Применяется элемент формата F (l, d), причем количество позиций после десятичной точки, заданной параметром d, больше количества позиций в дробной части числа, заданного в операторе DECLARE (т. е. $q < d$). Тогда имеющиеся в нашем распоряжении позиции используются так:



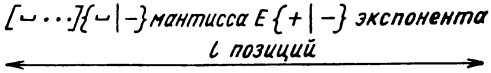
Пример. Значение переменных A и B такое же, как и в предыдущем примере.

```
PUT EDIT (A, B) (SKIP, F (10,5), X (2), F (9,6));
```



Элемент формата данных E. При использовании элемента формата данных E, имеющего общую форму E (*l*, *d* [,*s*]), внутренние арифметические данные обрабатываются как десятичные числа с плавающей точкой. Если выводимое значение отрицательно, в выводимый поток во время подготовки вставляется знак минус. В общем случае мантисса содержит десятичную точку. В последовательность знаков помещается буква E, которая в представлении числа является признаком числа с плавающей точкой и разграничителем мантиссы и экспоненты. Кроме того, вставляется знак экспоненты.

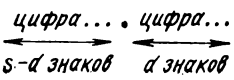
Внешняя форма числа с плавающей точкой имеет вид



Значение «экспоненты» всегда есть двузначное целое число. Установление ее значения происходит автоматически и управляется каждый раз указанной в формате мантиссой (*d*). Мантисса может быть представлена в четырех формах. Форма зависит от параметров *d* и *s*, так же как и от той связи, в которой они находятся.

Форма 1.

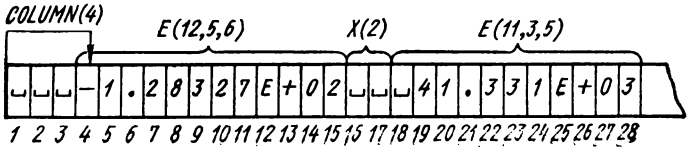
В элементе формата заданы *d* и *s*, причем $s > d$. Тогда мантисса имеет форму



Пример.

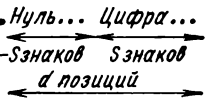
```
A = — 128,327; B = + 41331 58;
PUT EDIT (A B) (COLUMN (4), E (12,5, 6), X (2), E (11,3, 5));
```

Результат выполнения оператора вывода:



Форма 2.

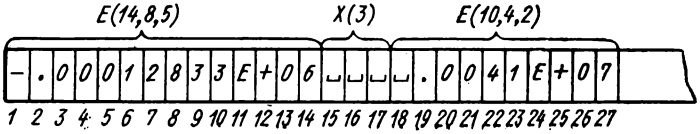
В элементе формата заданы d и s, причем $s < d$. Тогда мантисса имеет форму



Пример Те же значения данных A и B.

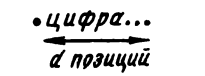
```
PUT EDIT (A, B) (E (14, 8, 5), X (3), E (10, 4,2));
```

Результат выполнения оператора:



Форма 3.

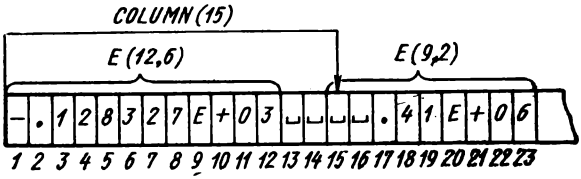
В элементе формата наряду с параметром l указан только параметр d. Тогда мантисса имеет следующую форму:



Пример.

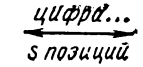
```
PUT EDIT (A, B) (E (12,6), COLUMN (15), E (9,2));
```

Результат выполнения оператора:



Форма 4.

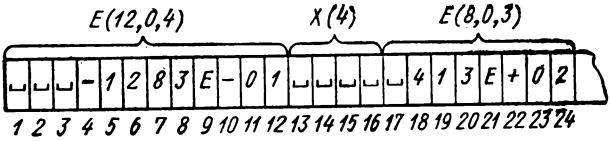
В элементе формата параметр d равен нулю. Это единственный случай, когда внутри мантиссы не появляется десятичная точка. Тогда мантисса имеет форму



Пример.

```
PUT EDIT (A, B) (E (12, 0, 4), X (4), E (8, 0, 3));
```

Результат выполнения оператора:



В заключение следует отметить, что при выводе числа с плавающей точкой должны быть соблюдены условия:

$$\text{если } d \geqslant s, \text{ то } l \geqslant 6 + d;$$
$$\text{если } d < s, \text{ то } l \geqslant 6 + s;$$
$$\text{если } d = 0, \text{ то } l \geqslant 5 + s.$$

Пять или шесть дополнительных позиций необходимы для вывода следующих знаков:

— знака мантиссы;
— десятичной точки;
— символа E, обозначающего начало экспоненты;
— знака экспоненты;
— двух десятичных цифр экспоненты.

Если указанные условия не выполняются, то число полностью не будет выводиться, «лишние» разряды будут сокращены, а последняя выводимая цифра мантиссы округляется.

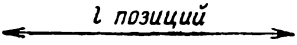
Элемент формата данных A. С помощью элемента формата A, общая форма которого A [(*l*)], описывается внешнее представление строки символов. Программисту предоставлена возможность указать количество символов, которое строка занимает в выводимой последовательности.

Если не указывать *l*, т. е. задавать только A, то:

— когда соответствующий элемент списка данных — переменная типа строки символов, длина строки при выводе такая же, какой она описана в операторе DECLARE;

— когда соответствующий элемент списка данных есть константа типа строки символов, для вывода используется длина константы.

При использовании элемента формата A (*l*) строка символов вносится в поле из *l* позиций и заполняется пробелами справа, если *l* больше количества записываемых символов:



Если *l* меньше количества записываемых символов, тогда строка усекается справа.

Пример. На первой строке нового листа должна быть напечатана следующая информация:

Позиция строки	Значение
1—9	Текущая дата
112—115	Лист
117—119	Номер листа

На третьей строке, начиная с позиции 42, должен быть дан заголовок листа:

ИНВЕНТАРНАЯ ВЕДОМОСТЬ

Пусть текущая дата представляет собой переменную типа строки символов с идентификатором DAT, а номер листа записывается с помощью переменной N

Вариант 1.

```
DECLARE DAT CHAR (9),
        N FIXED (3);
N = 0;
.....
M1: N = N + 1;
PUT PAGE EDIT (DAT, 'ЛИСТ␣,' N) (A, COLUMN (112), A, F (3));
PUT EDIT ('ИНВЕНТАРНАЯ ВЕДОМОСТЬ') (SKIP (3), COLUMN (42), A);
```

Так как нужно, чтобы между словом ЛИСТ и номером листа, который может быть трехзначным, был пробел, в строку букв ЛИСТ в рассматриваемом решении вставлен пробел и в списке данных указана константа типа строки знаков 'ЛИСТ ␣'.

Вариант 2.

```
.....
PUT EDIT (DAT, 'ЛИСТ', N)
     (PAGE, A, COLUMN (112), A (5), F (3))
     ('ИНВЕНТАРНАЯ ␣ ВЕДОМОСТЬ')
     (LINE (3), COLUMN (42), A);
```

Во втором варианте мы воспользовались тем, что в указанном элементе формата A (5) в пяти байтах располагается слева строка символов ЛИСТ.

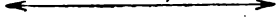
Вариант 3

```
.....
PUT EDIT (DAT, 'ЛИСТ', N)
     (PAGE, A, COLUMN (112), A, F (4)) и т. д.
```

В данном варианте мы воспользовались тем, что в элементе формата F (4) текущий трехзначный номер листа записывается справа.

Элемент формата данных B. С помощью элемента формата данных B, имеющего общую форму B (*l*), описывается внешнее представление строки битов. Форма выводимой строки битов имеет вид

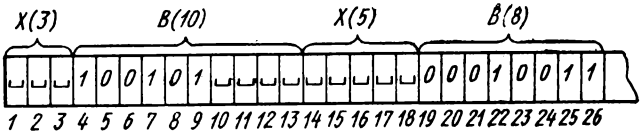
Строка битов [␣...]



Пример

```
DECLARE A BIT (8)
        B FIXED BINARY (5);
    A = '100101' B; B = 19;
PUT EDIT (A, B) (SKIP X (3), B (10), X (5), B (8));
```

Результат действия оператора вывода:



8.7. Оператор FORMAT

Если в процедурах встречаются операторы GET и PUT с одинаковыми или эквивалентными списками форматов, программист может использовать оператор FORMAT. Применение оператора FORMAT дает возможность сократить текст программы, а кроме того, с помощью него достигается бóльшая наглядность программы.

Оператор FORMAT имеет вид

метка: [метка]... FORMAT (список форматов);

Оператор FORMAT и операторы GET или PUT, которые используют этот оператор, должны содержаться внутри одного и того же блока. Для оператора FORMAT максимальный уровень вложенности списков элементов формата равен двум.

Метка, стоящая перед оператором, служит для ссылки к заранее подготовленным элементам формата, указанным в «списке форматов» оператора FORMAT. В этом случае в списке форматов операторов GET или PUT используется следующий элемент формата:

R (метка).

Буква R означает, что с помощью этого элемента формата происходит обращение к списку форматов, связанному с указанной в скобках меткой.

В этой связи необходимо подчеркнуть, что список форматов, заданный в оператор FORMAT, не может содержать элемент формата R.

Примеры.

1) Предположим, что в программе используются следующие операторы ввода — вывода:

```
.....
GET EDIT (A1, B, Y) (X (5), E (8,3), X (37), A (20), X (5), F (5));
.....
GET EDIT (A1, K, P, L) (X (5), E (8,3), X (37), A (20), X (5), F (5), X (3),
                        A (8));
```

```
PUT EDIT (A1, A2, Z, P, N) (SKIP, E (10, 3), X (5), E (8,3), X (37), A (20),
                            X (5), F (5), X (5), F (5));
```

Обращает внимание то, что в операторах ввода и вывода имеются одинаковые элементы списка форматов. Для сокращения записи текста программы выделим одинаковые элементы формата и запишем их в оператор FORMAT, который обозначим меткой M1. Тогда этот фрагмент программы будет иметь вид

```
M1: FORMAT (X (5), E (8, 3), X (37), A (20), X (5), F (5));
    ....
    GET EDIT (A1, B, Y) (R (M1));
    ....
    GET EDIT (A1, K, P, L) (R (M1), X (3), A (8));
    ....
PUT EDIT (A1, A2, Z, P, N) (SKIP, E (10, 3), R (M1), X (5) F (5));
```

2) В программе имеются следующие операторы ввода — вывода.

```
GET EDIT (A, B, C, D) (F (8,4), X (3), F (5,2), A (10), X (5), E (10,3));
.. .....
GET EDIT (M  N) (F (8,4), X (3), F (5,2));
. . . ...
PUT EDIT (M, N, C) (SKIP, F (8,4), X (5), F (5,2), A (10));
```

Используя оператора FORMAT, получаем программу вида:

```
M3: FORMAT (F (8,4), X (3), F (5,2), A (10), X (5), E (10,3));
    ..........
    GET EDIT (A  B  C, D) (R (M3));
    ..........
    GET EDIT (M, N) (R (M3));
    ..........
    PUT EDIT (M, N, C) (SKIP, R (M3));
```

В данном примере использован общий список форматов в операторе FORMAT. Этот оператор можно применять, так как список данных и список форматов в каждом операторе ввода—вывода обрабатываются попарно слева направо и «лишние» элементы формата автоматически игнорируются.

8.8. Внутренняя передача данных

Рассмотренные формы ввода — вывода с преобразованием значений данных описывали передачу информации между внешними носителями и основной памятью или наоборот. Однако в некоторых случаях необходимо производить перемещение значений данных внутри основной памяти.

Одним из таких способов перемещения является использование оператора присваивания, который позволяет передать значение одной переменной в область памяти другой. Но использование этого оператора достаточно жестко ограничивается правилами преобразования одних типов переменных в другие. Так, например, мы не имеем права присвоить значение строки символов арифметической переменной, и наоборот.

Другим способом перемещения данных внутри основной памяти с их преобразованием из символьной формы представления в ариф-

метическую (или битовую) или наоборот является использование операторов GET и PUT с режимом STRING. Такое перемещение называется внутренней передачей данных, которая выполняется по правилам обычного ввода — вывода потоком. В этом случае операторы передачи данных имеют следующий формат:

$$\begin{Bmatrix}\text{GET}\\ \text{PUT}\end{Bmatrix}\ \text{STRING}\ \begin{pmatrix}\text{переменная типа}\\ \text{строки символов}\end{pmatrix}\ \text{спецификация данных;}$$

При внутреннем вводе значения для элементов списка данных образуются из символов, перечисленных в переменной типа строки символов, и преобразуются в соответствующую форму. Символы, включенные в эту переменную, воспринимаются как поток символов при обычном вводе.

Существенное различие состоит в том, что данные берутся не из буфера ввода, а непосредственно из области памяти указанной переменной. Если атрибут длины этой переменной не обеспечивает числа символов, которые требуются для указанного списка данных, транслятором выдается сообщение об ошибке.

Примеры.

1) Объявлена переменная K со следующими атрибутами:

```
DCL K CHAR (16);
K = ' — 1345——, XYZ — — — — ';
```

по оператору

```
GET STRING (K) EDIT (A, P) (F (6,2), X (2), A (2));
```

переменным A и P будут присвоены следующие значения:

```
A = 13.45; P = 'YZ';
```

2) Область памяти переменной L2 с атрибутами CHAR (9) содержит следующие символы:

```
'ABCDEFGHI'
```

Трем элементам массива F, объявленного как

```
DCL F (3) CHAR (3);
```

по оператору

```
GET STRING (L2) EDIT (F) (A (3));
```

будут присвоены значения

```
F (1) = 'ABC'; F (2) = 'DEF'; F (3) = 'GHI';
```

При внутреннем выводе значения элементов списка данных после преобразования в символьный формат переносятся в область памяти переменной тип строки символов, причем заполнение начинается с первого байта указанной строки. Если количество символов, получаемых на основании списка данных, превышает длину переменной, то выдается сообщение об ошибке. Если длина указанной строки больше, чем требуется, то оставшиеся символы остаются неизменными.

Примеры.

1) Переменные X и Y имеют соответственно значения 33 и 8.56. Используя оператор

```
PUT STRING (K) EDIT (X, Y) (F (3), X (1), F (4,2));
```

в область памяти переменной K с атрибутами CHAR (8) будет введена следующая символьная строка:

'—33—8.56'

2) Переменная Z с атрибутами CHAR (4) имеет значение 'BBBB'. После выполнения оператора

```
PUT STRING (Z) EDIT ('M') (A);
```

она будет иметь значение 'MBBB'.

С помощью внутренней передачи данных можно практически присваивать переменной типа строки символов значение арифметического данного, что нельзя сделать с помощью оператора присваивания.

Одной из важных особенностей использования внутреннего ввода является то, что можно осуществить ввод даже в том случае, когда он идет с внешних устройств, к которым невозможно прямо обратиться при помощи операторов ПЛ/1 (например, ввод или вывод с перфоленты и т. п.). В этом случае необходимо написать подпрограмму ввода — вывода на машинно-ориентированном языке (Ассемблер). При вводе такая программа пересылает входные записи в область переменной типа строки символов, а затем с помощью оператора внутреннего ввода осуществляется присвоение значений переменным. При выводе производится пересылка данных в область памяти переменной типа строки символов с помощью оператора внутреннего вывода, а затем эта область используется подпрограммой, которая и осуществляет непосредственный вывод.

Так, например, при помощи оператора

```
PUT STRING (VARY) EDIT (A, B, C, D) (F (8,1), X (4), 2F (7,3),
                                     F (10));
```

в область памяти, отведенную строке символов VARY, будет передана строка из 36 символов, составленная из элементов, указанных в списке данных. В подпрограмме на Ассемблере эта строка выводится на носитель.

Кроме того, внутренняя передача данных может эффективно использоваться при вводе записей с различной структурой (с помощью некоторого признака, содержащегося в записи), при организации проверки вводимых цифровых данных или при разбивке вводимых числовых значений на отдельные цифры и др.

Примеры.

1) В программе некоторое число используется как строка символов и как арифметическое данное. Ввод его можно осуществить как строковое данное (с форматом A), а затем из этих символов при помощи внутреннего ввода преобразовать его в арифметическое значение.

```
DCL A FIXED (5,2), Z CHAR (5);
```

```
.....
GET EDIT (Z) (A (5));
GET STRING (Z) EDIT (A) (F (5,2));
```

2) При помощи встроенной функции DATA можно получить текущую дату в форме JJMMTT (где JJ — год; MM — месяц; TT — день) Если желательно использовать дату в обычной форме TT.MM.JJ, то это можно сделать с помощью следующих операторов:

```
DCL (Y, M, T) CHAR (2),
    DAT1 CHAR (6),
    DAT2 CHAR (8),
    DATE BUILTIN;
DAT1 = DATE;
GET STRING (DAT1) EDIT (Y, M, T) (A (2));
DAT2 = T || '.' || M || '.' || Y;
```

Эту же задачу можно решить с помощью наложения (атрибут DEFINED).

```
DCL 1 SDAT,
      2 Y CHAR (2),
      2 M CHAR (2),
      2 T CHAR (2),
DAT1 CHAR (6) DEFINED SDAT,
DAT2 CHAR (8),
DATE BUILTIN;
DAT1 = DATE;
DAT2 = T || '.' || M || '.' || Y;
```

3) В программе считываются карты, информация которых может иметь две различные внутренние структуры. Тот или иной вид перфокарты определяется значением символа, который стоит по некоторым соображениям не в начале перфокарты, а в 80-й колонке. Содержимое всей карты при этом запоминается в области памяти переменной, затем проверяется признак и далее в соответствии с его значением производится внутренняя передача с определенным форматом:

```
DCL A CHAR (79),
    B CHAR (1),
    L LABEL;
GET EDIT (A, B) (A (79), A (1));
IF B = '*' THEN L = M1; ELSE L = M2;
GET STRING (A) EDIT (...) (R (L));
M1: FORMAT (...);
M2: FORMAT (...);
```

4) Чтобы исключить ошибки при вводе цифровой информации, можно применить следующий способ. Ввести данные с перфокарт в область памяти переменной типа строки символов и затем проверить каждый символ, является ли он цифрой. Если в перфокарте обнаружена ошибка, то должна вводиться следующая перфокарта.

```
PROG: PROC OPTIONS (MAIN);
      DCL CARD FILE INPUT ENV (F (80) MEDIUM
          (SYSIPT, 6012)),
          Z CHAR (80),
          Z1 (80) CHAR (1) DEF Z,
          B (8) FIXED (10);
      ON ENDFILE (CARD) GOTO M1;
M:    GET FILE (CARD) EDIT (Z) (A (80));
```

```
        DO I = 1 TO 80;
            IF Z1 (I) < '0' THEN GOTO M;
        END;
        GET STRING (Z) EDIT (B) (F (10));
        .....
        /* ОБРАБОТКА МАССИВА B */
        GOTO M;
M1:     END PROG;
```

8.9. Ввод и вывод данных, ориентированный на записи

При передаче, ориентированной на записи, обрабатываемая информация рассматривается, как дискретные записи, в которых данные представлены во внутренней (машинной) форме. Операция ввода информации заключается в том, чтобы скопировать запись, передав ее с внешнего носителя в основную память. Назначение операции вывода — скопировать запись и передать ее из основной памяти на внешний носитель. Поскольку данные, входящие в запись, представлены во внутренней форме, никакого преобразования во время выполнения операторов ввода — вывода не производится. Благодаря этому передача данных записями происходит значительно быстрее, чем потоком.

При передаче записями каждый оператор ввода — вывода оперирует с целой записью. Для неблокированных записей передача данных каждый раз фактически связывает основную память и внешний носитель. При блокированных записях передача информации между основной памятью и внешним носителем осуществляется только тогда, когда завершится обработка предыдущего блока.

В операторах ввода — вывода записями могут использоваться неиндексированные переменные, имена массивов и старших структур, а также переменные типа метки и указателя, которые могут иметь любой класс памяти и не должны объявляться с атрибутом DEFINED.

При передаче данных, ориентированной на записи, используются четыре типа операторов:

— для ввода — оператор READ;

— для вывода — операторы WRITE, REWRITE и LOCATE.

В зависимости от способа организации набора данных эти операторы применяются в различных формах.

Оператор READ (читать) может применяться при обработке файлов с атрибутами INPUT или UPDATE. Он вызывает передачу записи из набора данных в область памяти, отведенную для некоторой переменной, или в буфер ввода. В случае блокированных записей оператор READ вызывает передачу блока записей в область буфера ввода (размер которого определяется длиной блока), а также

одновременно осуществляет передачу одной записи из буфера ввода в область памяти переменной для непосредственной ее обработки. Таким образом, в случае блокированных записей не каждое обращение к оператору READ вызывает физическую передачу записей с внешнего носителя в основную память. Кроме того, при использовании базированной переменной оператор READ с режимом SET позволяет обрабатывать записи непосредственно в буфере ввода.

Оператор WRITE (писать) может использоваться при обработке записей файла с атрибутами OUTPUT или DIRECT UPDATE (но не SEQUENTIAL UPDATE). Он вызывает передачу записи из памяти в набор данных. В случае неблокированных записей эта передача может происходить из области памяти некоторой переменной или из буфера. Если записи блокированы, оператор WRITE помещает запись в буфер вывода и после его заполнения выводит блок записей на внешний носитель.

Оператор REWRITE используется для замещения записей в файле с атрибутом UPDATE. Для файлов SEQUENTIAL UPDATE он указывает, что возвращается на место только что прочитанная и откорректированная запись набора данных, т. е. при обработке последовательного набора данных запись сначала читается с помощью оператора READ, а затем возвращается на место оператором REWRITE. Для файлов с атрибутами DIRECT UPDATE в операторе REWRITE должен задаваться ключ записи, что позволяет заменять любую запись набора данных без предварительного чтения.

Оператор LOCATE используется при обработке файла с атрибутами SEQUENTIAL BUFFERED OUTPUT и позволяет с помощью базированной переменной обрабатывать (создавать) записи в буфере вывода. Если записи не блокированы, то передача их из буфера в набор данных происходит при выполнении операторов LOCATE, WRITE или CLOSE. Если записи блокированы, то они передаются в набор данных после заполнения буфера вывода с помощью одного из указанных выше операторов. Подробно использование оператора LOCATE будет рассмотренно в § 9.4.

8.10. Ввод и вывод информации при использовании последовательно организованных наборов данных

В наборе данных с последовательной организацией записи не имеют никаких ключей. Когда создается набор данных, записи помещаются в него последовательно одна за другой в том порядке, как они поступают. Чтение и изменение записей этого набора также происходят последовательно, начиная с первой до последней (или наоборот, от последней к первой, если задан атрибут BACKWARD).

Для обработки записей последовательного набора данных в программе должен быть объявлен логический файл, связанный с этим набором. Общий формат объявления имеет вид

[DECLARE имя файла FILE RECORD [INPUT / OUTPUT / UPDATE]
[SEQUENTIAL [BUFFERED / UNBUFFERED] [BACKWARDS]
ENVIRONMENT ([F (длина блока [, длина записи]) / V (максимальная длина блока) / U (максимальная длина блока)])
MEDIUM (SYSxxx, *nnnn*) [CONSECUTIVE] [BUFFERS (*n*)]
[CTLASA / CTLYES] [LEAVE] [NOLABEL] [NOTAPEMK] [VERIFY];

Назначение каждого атрибута и режима подробно рассматривалось в § 8.2. Остановимся здесь на некоторых особенностях их использования применительно к последовательно организованному набору данных:

— если при обработке записей набора данных не создается буфер ввода — вывода (атрибут UNBUFFERED), то атрибуты INPUT или OUTPUT могут быть указаны не в объявлении файла, а при его открытии в операторе OPEN;

— при использовании атрибута BUFFERED можно в режиме BUFFERS (*n*) указать количество создаваемых буферов (один или два). Если режим BUFFERS не применен, то по умолчанию создается один буфер;

— атрибут BACKWARDS может применяться только для наборов данных, расположенных на магнитной ленте. При этом обязательно указывается режим LEAVE;

— режим NOLABEL используется при создании рабочих наборов данных на МЛ. Совместно с этим режимом для файлов с атрибутом OUTPUT может быть установлен режим NOTAPEMK (отмена записи маркера ленты при открытии рабочих файлов);

— режим VERIFY указывается только для выходных файлов на магнитных дисках.

Для создания последовательного набора данных используется оператор WRITE в следующей форме:

WRITE FILE (имя файла) FROM (переменная);

В режиме FILE указывается имя файла, связанного с набором данных, в который запись должна быть помещана. Этот файл должен быть описан с одним из атрибутов OUTPUT или UPDATE.

Переменная, стоящая в режиме FROM, может быть неиндексированной переменной, массивом или старшей структурой. Действие оператора WRITE заключается в том, что запись из области памяти переменной, указанной в режиме FROM, переносится либо в набор данных (если записи не блокированы), либо в буфер вывода (при блокированных записях).

Пример Необходимо создать последовательный набор данных на магнитной ленте, вводя записи документов с перфокарт. При этом каждый документ

вводится с отдельной перфокарты. Конец вводимых документов определяется табельным номером, равным 777. Форма документа состоит из следующих граф:

Табельный номер	Должность	Фамилия	Зарплата

На магнитной ленте записи должны блокироваться по 20 записей в блоке. Имя файла SPRAV, логическое устройство SYS005. При выводе используются два буфера. Программа создания набора данных имеет вид

```
PR1:  PROC OPTIONS (MAIN);
      DCL SPRAV FILE RECORD OUTPUT
          ENV (F (820, 41) MEDIUM (SYS005, 5012)
          BUFFERS (2));
      DCL 1 DOKUMENT,
          1 NR PIC'999',
          2 POST CHAR (15),
          2 NAME CHAR (20),
          2 ZP FIXED (5,2);
          OPEN FILE (SPRAV);
M:        GET EDIT (DOKUMENT) (SKIP F(3) A(15),
          A (20), F (5,2));
          IF NR = 777 THEN GOTO M1;
          WRITE FILE (SPRAV) FROM (DOKUMENT);
          GOTO M;
M1:       END PR1;
```

Для чтения записи из набора данных используется оператор READ в следующей форме:

READ FILE (имя файла) INTO (переменная);

В режиме FILE указывается имя файла, связанного с набором данных, из которого запись читается. Переменная, указанная в режиме INTO, задает область в основной памяти, в которую должна быть прочитана запись.

Если записи не блокированы, то независимо от использования буферов каждый оператор READ вызывает передачу записи с внешнего носителя в область памяти, отведенную переменной, указанной в режиме INTO. Если же записи блокированы, то оператор READ вызывает передачу физического блока записей с внешнего носителя в буфер и передачу первой записи этого блока из буфера в область памяти переменной. Передача нового блока с внешнего носителя в буфер происходит после того, как предыдущим обращением к оператору READ введена в область памяти переменной последняя запись из буфера ввода.

Пример. Необходимо переписать записи ленточного набора данных на диск. При этом записи файла на диске расширяются на 3 байта за счет введения в них переменной ND, которая содержит сумму удержания и имеет атрибуты FIXED (4,2). Имя файла на диске SPRAV1, логическое устройство SYS006, число записей в блоке 30. Удержание производится только с сотруд-

ников, имеющих определенные должности (POSTA). Сумма удержания указывается переменной UDER

Эти данные вводятся с перфокарт.

```
PR2:  PROC OPTIONS (MAIN);
      DCL SPRAV FILE RECORD INPUT
          ENV (F (820, 41) MEDIUM (SYS005, 5012)
          BUFFERS (2)),
          1 DOKUMENT,
            2 NR PIC'999',
            2 POST CHAR (15),
            2 NAME CHAR (20),
            2 ZP FIXED (5,2);
      DCL SPRAV1 FILE RECORD OUTPUT
          ENV (F (1320,44) MEDIUM (SYS006,5056)),
          1 DOKUMENT1,
            2 NR1 PIC '999',
            2 POST1 CHAR (1
            2 NAME1 CHAR (20),
            2 ZP1 FIXED (5,2),
            2 UD FIXED (4,2),
          POSTA CHAR (15),
          UDER FIXED (4,2);
      OPEN FILE (SPRAV), FILE (SPRAV1);
        ON ENDFILE (SPRAV) GOTO M1:
      GET EDIT (POSTA, UDER) (SKIP, A (15), F (4,2));
      M: READ FILE (SPRAV) INTO (DOKUMENT);
        IF POST = POSTA THEN DO; UD = UDER;
          NR1=NR; POST1=POST; NAME1=NAME;
          ZP1=ZP;
          WRITE FILE (SPRAV1) FROM (DOKUMENT1);
          GOTO M; END;
      UD=0; NR1=NR; POST1=POST; NAME1=NAME; ZP1=ZP;
      WRITE FILE (SPRAV1) FROM (DOKUMENT1);
      GOTO M;
      M1: CLOSE FILE (SPRAV), FILE (SPRAV1);
      END PR2;
```

Создание последовательных наборов данных на дисках имеет преимущество перед магнитной лентой в том, что содержание записей может быть изменено без перезаписи набора на другой носитель, как это делается при использовании магнитной ленты.

Разумеется, для последовательно организованных наборов данных внесение этих изменений осуществляется последовательным чтением каждой записи, ее корректировкой и помещением обратно на то же место. Изменение содержания записи без предварительного ее чтения невозможно. Для помещения записей в набор данных после их корректировки используется следующий оператор:

REWRITE FILE (имя файла) FROM (переменная);

При этом имя файла, указанное в режиме FILE, должно быть описано с атрибутом UPDATE.

Пример. В набор данных, созданный в предыдущем примере, необходимо внести изменения в зарплате для некоторых работников предприятия. Фамилии сотрудников и сумма новой зарплаты вводится с перфокарт. Последняя перфокарта информации не имеет (пустая перфокарта).

```
PR3: PROC OPTIONS (MAIN);
     DCL SPRAV1 FILE RECORD UPDATE
         ENV (F (1320,44) MEDIUM (SYS006,5056)),
         1 DOKUMENT1,
           2 NR1 PIC '999',
           2 POST1 CHAR (15),
           2 NAME1 CHAR (20),
           2 ZP1 FIXED (5, 2),
           2 UD FIXED (4,2),
         NAME CHAR (20),
         ZP FIXED (5,2);
M:   GET EDIT (NAME, ZP) (SKIP, A (20), F (5,2));
         IF NAME1='⎵' THEN GOTO M2;
         OPEN FILE (SPRAV1);
M1:       READ FILE (SPRAV1) INTO (DOKUMENT1);
         IF NAME1 = NAME THEN DO; ZP1=ZP;
        EWRITE FILE (SPRAV1) FROM (DOKUMENT1);
        GOTO M; END;
        GOTO M1;
M2:     CLOSE FILE (SPRAV1); END PR3;
```

При применении буферов для обновления последовательных наборов данных может быть использована возможность корректировки записей непосредственно в буфере ввода. Эта возможность реализуется с помощью базированной переменной и будет рассмотрена подробно в § 9.4.

Вывод на печать или на перфокарты. При выводе данных на устройство печати или выходной перфоратор программист может управлять переходом со строки на строку (или страницу) при печати или подачей перфокарт в определенный карман при выводе на перфоратор. Для этого необходимо в атрибуте ENVIRONMENT указать один из режимов CTLASA или CTLYES, тогда первый символ выводимой записи будет интерпретироваться как управляющий символ. Сам управляющий символ не выводится на внешний носитель; печатаются или перфорируются только символы, следующие за управляющим, которые собственно и составляют запись данных. Можно не указывать режим управления печатью или перфоратором, тогда каждая запись будет печататься на новой строке, а вывод перфокарт будет происходить в стандартный приемный карман. Значения управляющих символов для режимов CTLASA и CTLYES приведены в приложении 2.

Пример. Распечатать по записям набор данных, расположенный на магнитной ленте. При написании программы учесть следующие условия:

— длина строки печати 120 символов;

— использовать управляющие символы режима CTLASA;

— размер записи на МЛ равен 80 байтов, причем блок содержит 10 записей;

— первые шесть символов каждой записи представляют собой номер, который не должен печататься;

— набор данных заканчивается записью данных с номером 999999.

Программа будет иметь вид

```
PR4: PROC OPTIONS (MAIN);
DCL TAP FILE RECORD INPUT ENV (F (800, 80)
    MEDIUM (SYS011, 5012));
```

```
DCL PRESS FILE RECORD OUTPUT
    ENV (F (121) MEDIUM (SYSLST, 7030) CTLASA),
 1  TREG,
    2 NR PIC '(6)9',
    2 TEXT1 CHAR (37),
    2 TEXT2 CHAR (37),
  1 PREG,
    2 ST CHAR (1),
    2 DAT,
      3 INF1 CHAR (37),
      3 EMPTY CHAR (46),
      3 INF2 CHAR (37);
    ST='_'; EMPTY='_';
    OPEN FILE (TAP), FILE (PRESS);
M:  READ FILE (TAP) INTO (TREG);
    IF NR=999999 THEN GOTO M1;
    INF1=TEXT1; INF2=TEXT2;
    WRITE FILE (PRESS) FROM (PREG);
      GOTO M;
M1: CLOSE FILE (TAP);
END PR4;
```

Переменная ST в структуре выводимой записи служит для размещения управляющего символа (в данном случае это пробел, который указывает о переходе на следующую строку). Таким образом, с помощью управляющего символа программист управляет размещением выводимого текста по вертикали. Для управления размещением выводимого текста по горизонтали используется возможность вставки в строку символов пробела. В нашем примере для этого предназначена переменная EMPTY, которой в начале программы присвоена пустая строка.

На листе, который будет напечатан в результате работы программы PR4, строки будут следовать одна за другой без всякого разделения, игнорируя даже разделяющую линию между страницами. Поэтому во время вывода на печать при передаче, ориентированной на записи, программист должен сам следить за тем, чтобы переход на новую страницу происходил после печати определенного числа строк.

Изменим программу так, чтобы после печати каждых пятидесяти строк происходил переход на новую страницу

```
PR4: PROC OPTIONS (MAIN);
     .....
     OPEN FILE (TAP), FILE (PRESS);
     I=0;
M:   READ FILE (TAP) INTO (TREG);
     IF NR=999999 THEN GOTO M1;
     I=I+1;
     INF1=TEXT1; INF2=TEXT2;
     IF I=51 THEN DO; ST='1'; I=1; END;
        ELSE ST=' ';
     WRITE FILE (PRESS) FROM (PREG);
     GOTO M;
M1:  END PR4;
```

В данном примере мы не использовали оператор CLOSE для явного закрытия файлов TAP и PRESS, поэтому они будут закрываться автоматически по умолчанию при окончании программы.

Пример. Необходимо отперфорировать по записям набор данных, расположенный на дисках, соблюдая следующие условия:

— для управления выбором приемного кармана должны быть использованы управляющие символы CTLYES;

— должны перфорироваться только первые 60 символов каждой записи;

— первый байт каждой записи содержит символ, определяющий выбор соответствующего кармана перфоратора. Если этот символ есть 'A', карта должна быть помещена в карман 1; все другие перфокарты помещаются в карман 2. Последняя запись набора данных начинается с символа '*'.

Программа будет иметь вид

```
PR5:    PROC OPTIONS (MAIN);
  DCL DSK FILE RECORD INPUT
     ENV (F (600, 120) MEDIUM (SYS010, 5056));
  DCL CARD FILE RECORD OUTPUT
     ENV (F (61) MEDIUM (SYSPCH, 7010)
     CTLYES),
     1 DREG,
       2 Z CHAR (1),
       2 TEXT CHAR (119),
     1 CREG,
       2 ST CHAR (1),
       2 DAT CHAR (60);
  OPEN FILE (DSK), FILE (CARD);
M:   READ FILE (DSK) INTO (DREG);
  IF Z='*' THEN GOTO M1;
  IF Z = 'A' THEN
     UNSPEC (ST)='00000001'B;
  ELSE
     UNSPEC (ST)='01000001'B;
  DAT=STRING (DREG);
  WPITE FILE (CARD) FROM (CREG);
  GOTO M;
M1: END PR5;
```

При написании программы мы воспользовались псевдопеременной UNSPEC для назначения управляющего символа выбора соответствующего кармана. Встроенная функция STRING позволила сцепить элементы структуры DREG в одну строку.

При выводе на перфокарты в данном случае будут перфорироваться первые 60 колонок. Оставшиеся 20 колонок будут свободны на всех перфокартах. Каждое обращение к оператору WRITE не только осуществляет передачу одной записи, но и выводит целую перфокарту независимо от указанной длины записи.

8.11. Ввод и вывод информации при использовании региональных и индексно-последовательных файлов

При программировании некоторых инженерных, а особенно информационных или экономических задач приходится иметь дело с большим объемом данных, которые необходимо обработать в программе. Использование последовательного способа организации данных в этом случае приводит к неоправданно большим затратам машинного времени для их обработки. С появлением запоминающих устройств прямого доступа (магнитных дисков) открылась возможность значительно сократить время обработки больших объемов информации, непосредственно выбирая или помещая в набор данных любую запись.

В связи с возможностью прямого доступа к записям набора данных появились две новые формы их организации:

— региональная организация набора данных;

— индексно-последовательная организация набора данных.

В рассматриваемом подмножестве языка ПЛ/1 региональные наборы данных разрешается строить и обрабатывать только прямым методом доступа. В зависимости от способа размещения записей в этих наборах данных они имеют два способа организации: REGIONAL (1) и REGIONAL (3). В этих наборах могут использоваться только неблокированные записи фиксированной длины.

Индексно-последовательные наборы данных дают возможность использовать при обработке записей как прямой, так и последовательный методы доступа. Однако создать такие наборы данных можно только последовательным методом доступа. В индексно-последовательных наборах данных могут использоваться блокированные и неблокированные записи фиксированной длины.

8.12. Создание и обработка наборов данных с организацией REGIONAL (1)

При создании и обработке записей регионально организованного набора данных в программе должен быть объявлен логический файл, связанный с этим набором. Формат объявления имеет следующий вид:

DECLARE имя файла FILE RECORD {INPUT / OUTPUT / UPDATE}

DIRECT [KEYED] ENVIRONMENT (F (длина записи)

{REGIONAL (1) / REGIONAL (3)} MEDIUM (SYSxxx, 5056)

[VERIFY] [KEYLENGTH (*n*)] [EXTENTNUMBER (*n*)]);

Назначение каждого атрибута и режима подробно было рассмотрено в § 8.2. Здесь необходимо еще раз напомнить о некоторых особенностях их использования:

— при указании атрибута DIRECT атрибут KEYED можно не указывать, так как он предполагается по умолчанию;

— режим VERIFY указывается только для выходных файлов, имеющих атрибут OUTPUT;

— режим KEYLENGTH (*n*) указывается только для файлов REGIONAL (3);

— режим EXTENTNUMBER (*n*) для этих файлов может указываться в том случае, если записи набора данных размещаются в различных областях магнитного диска и занимают при этом более трех отдельных участков. Например, набор данных занимает на 10-м цилиндре 6 дорожек (первый участок), весь 52-й цилиндр (второй участок), 3 дорожки на 102-м цилиндре (третий участок) и весь

104-й цилиндр (четвертый участок). В этом случае указывается режим EXTENTNUMBER (4).

Напомним, что при такой форме организации набора данных записи должны быть неблокированными, фиксированной длины.

При создании такого набора записи заносятся в определенные области, к которым затем может быть получен прямой доступ с помощью ключа. Каждая область имеет номер. Для каждого набора области нумеруются последовательно, начиная с нуля. Дисковая операционная система предполагает, что номер области представлен в операторе ввода — вывода как восьмизначное целое число в форме строки символов. Обычно номер области в программе описывают с помощью атрибута PICTURE '(8)9'. Имя этой области указывается в режимах KEY или KEYFROM операторов READ, WRITE или REWRITE.

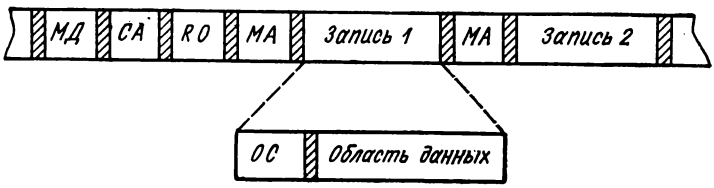
Перед созданием набора данных эти области должны быть подготовлены программистом при помощи специальной программы чистки дисков (CLRDSK). При использовании этой программы создаются фиктивные записи, заполненные символами, указанными программистом в параметрах этой программы. Длина фиктивной записи равна длине записи создаваемого регионального набора данных. Во время создания набора фиктивные записи заменяются действительными.

Подробное описание использования программы чистки дисков (CLRDSK) будет сделано в § 11.5.

Для наборов REGIONAL (1) ключ записи, используемый в программе, идентифицирует записи только внутри программы. На носителе (диске) записи идентифицируются номером области, в которой они находятся относительно начала набора данных (область — одна запись). Поэтому область ключа для записей этих наборов не формируется. Физическая система управления вводом — выводом (СУВВ) исходя из номера определяет область относительно начала набора и вычисляет адрес записи на дисках. Поэтому можно читать как действительные, так и фиктивные записи набора данных.

При работе с региональными файлами REGIONAL (1) могут оставаться пустые области, которые впоследствии заполняются новыми записями. Кроме того, необходимо следить, чтобы при формировании ключей для разных записей не было бы одинаковых ключей, т. е. разным записям не была бы назначена одна и та же область.

Формат дорожки дисков для таких наборов данных имеет следующий вид:



где МД — маркер дорожки, который указывает физическое начало дорожки;

СА — собственный адрес, который определяет положение дорожки на пакете дисков;

RO — запись, которая содержит информацию, описывающую дорожку;

МА — маркер адреса, который служит для указания начала каждой записи данных;

ОС — область счетчика, которая содержит соответствующий номер области.

Для создания регионального набора данных REGIONAL (1) используется оператор WRITE в следующей форме:

WRITE FILE (имя файла) FROM (переменная)
KEYFROM (выражение);

Переменная, стоящая в режиме FROM, может быть неиндексированной переменной, массивом или старшей структурой. Выражение в режиме KEYFROM определяет значение ключа, который указывает номер области размещенной записи в наборе данных. Действие оператора WRITE заключается в том, что запись из области памяти переменной, указанной в режиме FROM, переносится в набор данных именно в ту область, номер которой указан в режиме KEYFROM.

Пример. Находящийся на перфокартах набор данных о сотрудниках одного предприятия переписать с использованием организации REGIONAL (1) на магнитные диски так, чтобы к каждой записи данных можно было получить доступ по табельному номеру (NR). Кроме того, требуется выполнить следующие условия:

— записи, читаемые с перфокарт, имеют длину 80 символов. На магнитные диски они должны быть переписаны неизменными;

— используемый для вычисления номера области (PROV) табельный номер (NR) находится в первых пяти позициях каждой перфокарты прочитанной записи. Последняя перфокарта файла отмечается табельным номером 99999. Эта конечная запись не должна переноситься на магнитные диски.

Программа создания набора данных REGIONAL (1) будет иметь вид

```
PR6:    PROCEDURE OPTIONS (MAIN);
  DECLARE CARD FILE RECORD INPUT
      ENV (F (80) MEDIUM (SYSIPT, 6012));
  DECLARE REG1 FILE RECORD OUTPUT DIRECT
      KEYED ENV (F (80) MEDIUM (SYS012,
      5056) REGIONAL (1)
      EXTENTNUMBER (1)),
      1 REF,
        2 NR PICTURE '(5)9',
        2 INF CHARACTER (75),
      PROV PICTURE '(8)9';
    OPEN FILE (REG1);
M: READ FILE (CARD) INTO (REF);
      IF NR=99999 THEN GOTO M1;
      PROV=NR - 1;
      WRITE FILE (REG1) FROM (REF) KEYFROM
      (PROV);
      GOTO M;
```

```
M1:    CLOSE FILE (REG1);
       END PR6;
```

В нашем примере оператор WRITE вызывает занесение в набор данных REG1 записи, расположенной в структуре REF. Ключ PROV, используемый в программе, служит для вычисления номера области, в которой размещается запись. Поэтому построение набора данных может, вообще говоря, начинаться с любой записи. В данном примере ключ записи (PROV) вычисляется исходя из табельного номера.

Для чтения записи из набора данных используется оператор: READ FILE (имя файла) INTO (переменная) KEY (выражение).

Значение выражения, указанного в режиме KEY, преобразованное в строку символов, определяет номер области набора данных, из которой должна быть прочитана запись.

Переменная, указанная в режиме INTO, задает область в основной памяти, в которую должна быть прочитана запись. Она может быть неиндексированной переменной, массивом или старшей структурой.

Пример. Информация об определенных сотрудниках учреждения должна быть прочитана из набора данных, построенных в предыдущем примере, и напечатана вместе с текущим номером. Из информации о сотрудниках известны только табельные номера. Табельные номера избранных сотрудников будут прочитаны с перфокарт оператором GET (последняя из перфокарт помечена табельным номером 99999).

```
PR7: PROCEDURE OPTIONS (MAIN);
       DECLARE REC1 FILE INPUT RECORD DIRECT
       KEYED ENV (F(80) MEDIUM (SYS012,5056)
       REGIONAL (1));
     DECLARE PROV PICTURE '(8)9',
       1 REF,
         2 NR PICTURE'(5)9',
       2 INF CHAR (75),
       ST CHAR (1);
      OPEN FILE (REG1);
      I=0;
M:    GET EDIT (NR, ST) (F (5), X (74), A (1))
       IF NR=99999 THEN GOTO M1:
       PROV=NR-1;
       READ FILE (REG1) INTO (REF) KEY (PROV);
       I=I + 1;
       PUT EDIT (I, STRING (REF)) (SKIP, F (4), X (2),A);
       GOTO M;
M1:    CLOSE FILE (REG1);
       END PR7.
```

В данном примере:

— переменная ST использована только для ограничения перфокарты с табельным номером сотрудника, чтобы иметь возможность каждый табельный номер выбивать на отдельной перфокарте;

— встроенная функция STRING позволяет сцепить элементы структуры REF в одну строку символов и вывести ее на печать с форматом A.

Если необходимо произвести корректировку записей набора данных, то файл, связанный с этим набором, должен быть объявлен с атрибутом UPDATE. При этом каждое изменение в записи данных всегда связано с чтением ее в основную память, корректировкой и

последующим помещением на старое место. Чтение записи производится с помощью оператора READ, а возвращение на старое место — с помощью оператора REWRITE:

REWRITE FILE (имя файла) FROM (переменная) KEY (выражение);

Пример. В программе должен быть обработан набор данных с информацией о сотрудниках, имеющий способ организации REGIONAL (1). Во время обработки должны быть выполнены следующие условия:

— информацию о некоторых сотрудниках нужно извлечь с помощью его табельного номера и вывести на печать;

— набор данных нужно расширить новыми записями (за счет новой группы сотрудников);

— в записях данных изменить фамилию сотрудника (например, в результате бракосочетания);

— информация, предназначенная для обработки, должна читаться с перфокарт. Управляющие перфокарты, которые содержат последовательности символов 'LOOK', 'ADDITION', 'CHANGE', или 'ENDE', лежат перед соответствующими перфокартами данных;

— для чтения записей из набора данных использовать только табельные номера сотрудников (управляющая карта 'LOOK');

— для обновления информации с перфокарточного ввода будут читаться совершенно новые записи (управляющая карта 'ADDITION');

— в процессе обновления данных наряду с табельными номерами должны будут в соответствии с постановкой задачи читаться также новые фамилии (управляющая карта 'CHANGE');

— последовательность появления управляющих перфокарт с соответствующими картами данных произвольна. Однако карта ENDE всегда должна быть прочитана последней

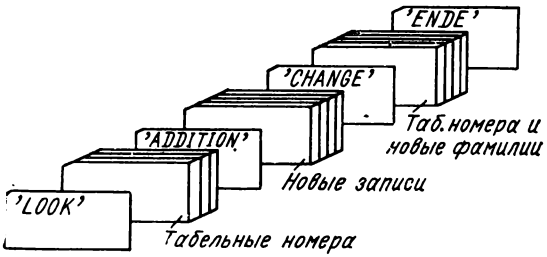
Программа обновления набора данных имеет вид

```
PR8: PROCEDURE OPTIONS (MAIN);
     DECLARE REG1 FILE RECORD UPDATE
         DIRECT KEYED ENV (F (80) MEDIUM (SYS012,
         5056) REGIONAL (1)),
         PROV PICTURE '(8)9',
         MARK LABEL,
         SNAME1 CHAR (30),
         1 REF,
           2 NR PICTURE'(5)9',
           2 SNAME CHAR (30),
           2 INF CHAR (45),
           COLL CHAR (80) DEFINED REF;
     OPEN FILE (REG1);
M:   GET EDIT (COLL) (A (80));
     IF SUBSTR (COLL, 1,4) = 'LOOK' THEN
          DO; MARK = M1; GOTO M; END;
        IF SUBSTR (COLL, 1,8) = 'ADDITION' THEN
        DO; MARK=M2; GOTO M; END;
     IF SUBSTR (COLL, 1,6)='CHANGE' THEN
        DO; MARK=M3; GOTO M; END;
     IF SUBSTR (COLL, 1,4)='ENDE' THEN GOTO M4;
     PROV=NR — 1; GOTO MARK;
M1:  READ FILE (REG1) INTO (REF) KEY (PROV);
     PUT EDIT (STRING (REF)) (SKIP, A);
     GOTO M;
M2:  WRITE FILE (REG1) FROM (REF) KEYFROM (PROV);
     GOTO M;
```

```
M3: SNAME 1=SNAME;
    READ FILE (REG1) INTO (REF) KEY (PROV);
    SNAME=SNAME1;
    REWRITE FILE (REG1) FROM (REF) KEY (PROV);
    GOTO M;
M4: CLOSE FILE (REG1);
    END PR8;
```

В данном случае при объявлении переменной COLL использован атрибут DEFINED, с помощью которого может быть определена общая область памяти для двух или нескольких переменных (в нашем случае для переменной COLL и структуры REF).

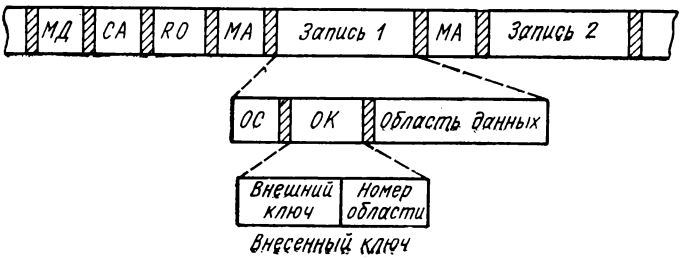
Перфокарты с информацией, подлежащей обработке, формируются в следующем порядке:



8.13. Создание и обработка наборов данных с организацией REGIONAL(3)

При создании наборов данных с организацией REGIONAL (3) каждая область представляет собой одну дорожку, которая может содержать несколько записей. Эти области (дорожки) также нумеруются системой относительно начала набора данных. Первая область (дорожка) имеет номер 0, вторая — 1 и т. д.

Так как на одной дорожке может быть размещено несколько записей, необходимо каждую запись идентифицировать, кроме номера области (дорожки), на которой она расположена, еще каким-то дополнительным признаком для поиска ее на дорожке. Таким признаком является логический (внешний) ключ, который в совокупности с номером области (дорожки) образует внесенный ключ записи. Этот ключ при написании соответствующей записи помещается в область ключа. Формат дорожки для таких наборов данных имеет вид



где ОС — область счетчика; ОК — область ключа.

При создании набора данных внесенный ключ записи указывается в режиме KEYFROM оператора WRITE. При чтении или обновлении записей этих наборов данных должны использоваться те же ключи, что и при их создании. Эти ключи указываются в режиме KEY операторов READ или REWRITE.

Внесенный ключ записывается на дорожке перед записью данных в область ключа. Система требует, чтобы внесенный ключ был в виде строки символов и имел длину не менее 9 символов. При этом последние 8 символов строки предсталяют собой номер области и содержат десятичные цифры без пробелов. Внесенные ключи получаются сцеплением внешнего ключа и номера области. Все ключи файла должны быть одной длины, которая задается в режиме KEYLENGTH (*n*) атрибута ENVIRONMENT описания файла.

В связи с тем, что создание регионального набора данных REGIONAL (3) можно начать с любой записи (атрибут DIRECT), мы должны так же, как перед созданием набора данных REGIONAL(1), «форматизовать» используемые дорожки магнитных дисков с помощью программы CLRDSK. После форматизации системе известно, сколько записей может быть записано на одной дорожке, т. е. в одной области.

Для создания файла используется оператор WRITE в виде

WRITE FILE (имя файла) FROM (переменная)
KEYFROM (выражение);

Действие оператора WRITE заключается в том, что запись, состоящая из внесенного ключа (указанного в режиме KEYFROM) и данных, записывается на «первое свободное место» дорожки, адресуемой номером области. При этом под «первым свободным местом» на дорожке всегда понимается псевдозапись, которая следует непосредственно за последней уже записанной на этой дорожке записью набора данных. Записи с одинаковыми номерами областей будут последовательно, по мере поступления, записываться на одну и ту же дорожку, пока оне не будет заполнена. Дальнейшие попытки занесения записей с тем же номером области будут приводить к прерыванию программы (исключительная ситуация KEY). Попытка записи с внесенным ключом, уже имеющимся в наборе, также вызывает прерывание программы (ситуация KEY). Перезапись, как и в случае файлов REGIONAL (1), с помощью оператора WRITE невозможна.

Пример. С использованием организации REGIONAL (3) на магнитном диске должен быть создан литературный каталог. В качестве логического (внешнего) ключа нужно использовать фамилию автора. Все записи, относящиеся к фамилиям авторов с одинаковыми начальными буквами, должны быть записаны в одной области. Записи содержат название нового произведения автора, фамилия автора не входит в состав записи.

Фамилия автора (30 символов) и соответствующее название произведения (120 символов) должны содержаться попарно на двух идущих друг за другом перфокартах и читаться с помощью оператора GET. Последняя пер-

фокарта содержит строку символов '*ENDE*' в шести первых позициях. В соответствии с количеством букв в латинском алфавите для набора данных используется 26 дорожек (номера областей от 0 до 25).

Программа создания файла REGIONAL (3) будет иметь вид

```
PR9: PROCEDURE OPTIONS (MAIN);
     DECLARE REG3 FILE RECORD OUTPUT DIRECT
        KEYED ENV (F(120) REGIONAL (3) KEYLENGTH
        (38) MEDIUM (SYS013, 5056));
     DECLARE REG RETURNS (FIXED DECIMAL (8)),
        TITLE CHAR (120),
     1 MATTER,
          2 AUTHOR CHAR (30),
          2 NR PICTURE'(8) 9',
        MAT CHAR (38) DEFINED MATTER;
     OPEN FILE (REG3);
M:   GET EDIT (AUTHOR, TITLE) (A(30), X (10),A(120));
     IF AUTHOR='* ENDE*' THEN GOTO M1;
     NR=REG (AUTHOR),
     WRITE FILE (REG3) FROM (TITLE) KEYFROM   (MAT);
     GOTO M;
  M1: CLOSE FILE (REG3);
     END PR9;
```

Для вычисления номера области по первой букве фамилии автора вызывается процедура—функция с именем REG

```
REG: PROCEDURE (X) FIXED (8);
     DECLARE X CHARACTER (30);
     I=UNSPEC (SUBSTR (X, 1, 1)) — 193;
     IF I > 24 THEN K = 15;
        ELSE IF I > 8 THEN K = 7;
             ELSE K = 0;
     RETURN (I — K);
     END REG;
```

В данной процедуре использованы встроенные функции, с помощью которых сначала выделяется первая буква фамилии автора (SUBSTR), а затем она преобразуется во внутреннее представление (UNSPEC), которое используется для вычисления номера соответствующей области (дорожки).

Так, например, если фамилия автора начинается с буквы А, то должно быть возвращено значение нуль. Внутреннее представление A = '11000001'B. При вычислении I это значение А будет преобразовано в двоичное число с фиксированной точкой, которое равно $A = 11000001_2 = 193_{10}$, и когда из него будет вычитаться число 193, то результат будет равен 0.

Для буквы J внутреннее представление 11010001_2, т. е. 209_{10}. Тогда I = 209 — 193 = 16, а номер области равен RETURN (16 — 7), т. е. 9, что и требуется, так как буква J является десятой буквой алфавита.

Чтение записи из набора данных производится с помощью оператора

READ FILE (имя файла) INTO (переменная) KEY (выражение).

Ключ, указанный в режиме KEY, называется исходным ключом. При чтении записи этот исходный ключ должен точно совпадать с внесенным ключом записи, уже имеющейся в наборе данных. Действие оператора READ заключается в том, что прежде всего из последних восьми позиций ключа определяется номер области, т. е. устанавливается соответствующая дорожка, а затем на этой дорожке

последовательно (начиная от начала дорожки) на основании исходного ключа отыскивается запись. При этом внесенные ключи всех записей в области сравниваются с исходным ключом искомой записи до тех пор, пока не произойдет совпадение. Совпадение означает, что запись найдена; сразу же после этого происходит передача данных.

Для замены записи, уже имеющейся в наборе данных, используется оператор REWRITE:

REWRITE FILE (имя файла) FROM (переменная) KEY (выражение);

Переменная, стоящая в режиме FROM, представляет собой запись, которая должна заменить уже имеющуюся в наборе данных.

Пример. Набор данных, построенный в предыдущем примере, должен быть обновлен следующим образом: некоторые содержащиеся в записях названия произведений должны быть заменены новыми.

На печать должны выдаваться вместе с фамилией автора как старое, так и новое названия его произведения. Названия новых произведений вместе с фамилиями авторов должны читаться парами с перфокарт. В этом случае программа обновления набора данных будет выглядеть так:

```
PR10: PROCEDURE OPTIONS (MAIN);
        DECLARE REG3 FILE RECORD UPDATE DIRECT
          KEYED ENV (F (120)
          MEDIUM (SYS013,5056) REGIONAL (3)
          KEYLENGTH (38));
        DECLARE REG RETURNS (FIXED (8)),
          TITLE CHAR (120),
          1 MATTER,
            2 AUTHOR CHAR (30),
              2 NR PICTURE '(8)9',
            MAT CHAR (38) DEFINED MATTER;
        OPEN FILE (REG3);
    M1: GET EDIT (AUTHOR) (A (30));
        IF AUTHOR=' * ENDE*' THEN GOTO M1;
        NR=REG (AUTHOR);
        READ FILE (REG3) INTO (TITLE)  KEY (MAT);
        PUT EDIT (AUTHOR, TITLE) (SKIP, A(30),SKIP (2),
        A (120));
        GET EDIT (TITLE) (X (10), A (120));
        PUT EDIT (TITLE) (SKIP, A (120));
        REWRITE FILE (REG3) FROM (TITLE) KEY (MAT);
        GOTO M;
     M1: CLOSE FILE (REG3);
END PR10;
```

Так как в нашем примере должно печататься старое название произведения, обновляемая запись читается в основную память после того, как автор известен. Если бы не нужно было печатать старое название произведения, оператор READ был бы излишним. Для вычисления номера области вызывается уже описанная процедура-функция с именем REG.

8.14. Создание и обработка индексно-последовательных наборов данных

Способы организации наборов данных, которые были рассмотрены ранее, не позволяли получить к записям одновременно и прямой, и последовательный доступ. В случае последовательно организованного файла к записям возможен только последовательный доступ, а в случае регионального файла — только прямой.

Индексно-последовательная организация набора данных позволяет использовать оба метода доступа.

При обработке записей индексно-последовательного набора данных в программе должен быть объявлен логический файл, связанный с этим набором. Формат объявления:

DECLARE имя файла FILE RECORD {INPUT / OUTPUT / UPDATE}

[SEQUENTIAL / DIRECT] KEYED [BUFFERED]

ENVIRONMENT (F(длина блока [, длина записи])

INDEXED MEDIUM (SYSxxx, 5056)

[VERIFY] KEYLENGTH (*n*) [KEYLOG (*n*)]

[OFLTRACKS (*n*)] EXTENTNUMBER (*n*)

[BUFFERS (*n*)] [INDEXMULTIPLE]

[INDEXAREA (*n*)] [ADDBUFF (*n*)];

Назначение каждого атрибута и режима было подробно рассмотрено выше (§ 8.2). Напомним особенности использования некоторых режимов:

— режим VERIFY указывается только для выходных файлов, имеющих атрибут OUTPUT;

— режим KEYLOG (*n*) указывается только при использовании блокированных записей;

— режим INDEXAREA (*n*) уменьшает время обработки набора данных и может использоваться только для файлов с атрибутом DIRECT;

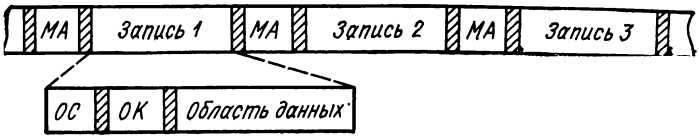
— режим ADDBUFF (*n*) используется для файлов с атрибутами DIRECT UPDATE. Он позволяет повысить эффективность обработки записей набора данных. Записи располагаются в наборе данных последовательно в соответствии с возрастанием некоторого признака упорядочивания, который может быть использован в качестве ключа. Таким образом, каждая запись в индексно-последовательном наборе данных должна иметь ключ, который может содержать любые символы. Он может иметь длину от 1 до 255 символов. Записи,

которые используются при создании набора данных, должны поступать в логически возрастающей последовательности своих ключей.

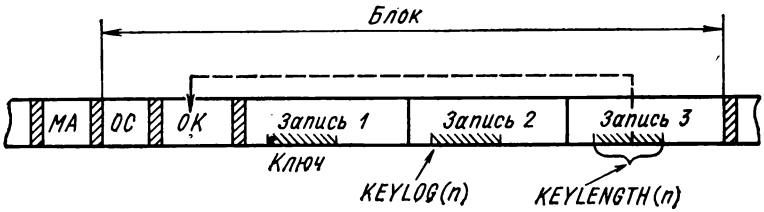
В индексно-последовательном наборе данных могут использоваться блокированные и неблокированные записи фиксированной длины. В случае блокированных записей ключ размещается внутри самой записи, положение его начала указывается с помощью режима KEYLOG, а его длина с помощью KEYLENGTH.

Форматы записи на дорожке для этого набора данных имеют следующий вид:

а) для неблокированных записей фиксированной длины



б) для блокированных записей



Значение ключа последней записи в блоке переносится в область ключа в начало блока.

Индексно-последовательный набор данных характеризуется тем, что расположение его записей определяется с помощью индексов. Индекс сам является частью набора данных, т. е. он представляет некоторую специальную запись, которая располагается на носителе. При создании набора данных он автоматически строится системой, и в дальнейшем система сама заботится об его обновлении, которое необходимо в процессе добавления записей. Существует индекс дорожки, индекс цилиндра и главный индекс. Индекс дорожки строится для каждого цилиндра, занимаемого набором данных, а для всего набора (если он занимает несколько цилиндров) строится индекс цилиндра. Индекс цилиндра находится в отдельной области пакета дисков, а индекс дорожки располагается с начала нулевой дорожки каждого цилиндра.

При создании очень больших наборов данных может использоваться так называемый главный индекс, который задается програм-

мистом с помощью указания режима INDEXMULTIPLE в атрибуте ENVIRONMENT.

Использование главного индекса позволяет системе производить поиск нужного индекса на одной из дорожек индекса цилиндра, а не на всех дорожках. Причем главный индекс рекомендуется использовать, когда индекс цилиндра занимает более трех дорожек. Главный индекс и индекс цилиндра располагаются на диске перед основной областью набора данных.

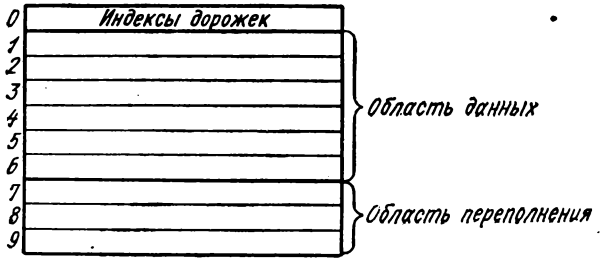


Рис. 8.1. Распределение дорожек цилиндра.

Отличительной особенностью рассматриваемого набора данных является также то, что десять дорожек каждого набора цилиндра, образующего одну область на пакете дисков, разделяются на три части (рис. 8.1):

— индексы дорожек цилиндра (содержат сведения о заполнении дорожек цилиндра);

— область данных (содержит записи набора);

— область переполнения.

В индексе дорожки для каждой дорожки помещаются две записи (рис. 8.2).

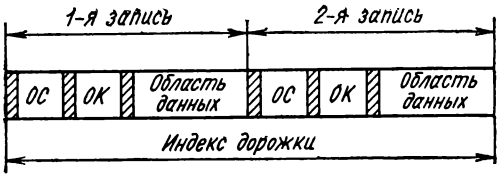


Рис. 8.2. Строение индекса дорожки.

Первая запись определяет основную дорожку области данных файла:

— в области счетчика указывается последовательный номер записи на нулевой дорожке (включая записи, входящие в индексы);

— в области ключа размещается значение максимального ключа записи на дорожке (т. е. ключа записи, которая является последней на дорожке);

— в области данных, которая всегда имеет длину 10 байтов, размещается адрес первой записи на дорожке в форме CCHHR.

Вторая запись определяет дорожку переполнения:

— в области счетчика указывается следующий по порядку номер записи;

— в области ключа размещается ключ последней записи, которая была на дорожке в момент создания файла;

— в области данных указывается адрес на дорожке переполнения, куда была перенесена эта запись.

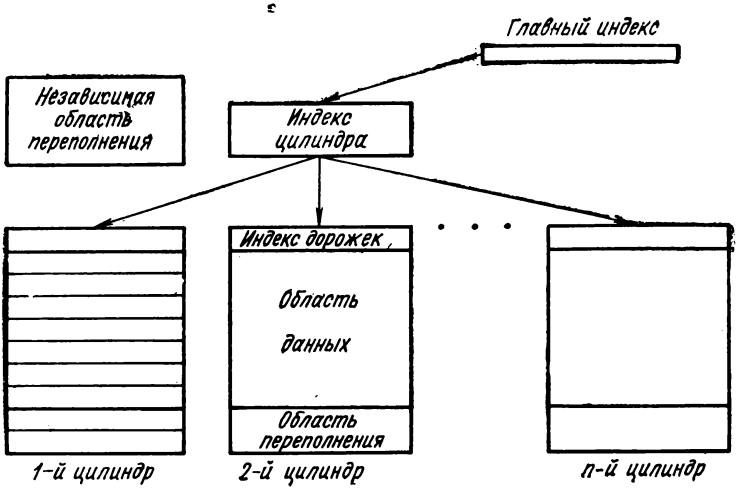
Следует отметить, что при переносе записи на дорожку переполнения размер области данных увеличивается на 10 байтов. Эти дополнительные байты называются полем связи (SL). В этом поле устанавливается адрес записи с большим ключом, перенесенной в область переполнения с той же дорожки (для первой записи, перенесенной в область переполнения, поле связи заполняется нулями). Это поле обеспечивает реализацию принципа сцепления при последующих обращениях к набору данных, т. е. просмотр записей в области переполнения в порядке возрастания их ключей, где физическое расположение записей может быть произвольным.

Индекс цилиндра содержит записи, аналогичные по структуре записям индекса дорожек, но характеризующие каждый цилиндр, отведенный набору данных. Для каждого цилиндра строится одна запись в индексе, содержащая в поле данных адрес индекса дорожек для этого цилиндра (всегда адрес нулевой дорожки), а в поле ключа — максимальное значение ключа записи, расположенной на этом цилиндре. Индекс цилиндра для набора данных должен обязательно располагаться на отдельных цилиндрах тома перед цилиндрами отведенными для записей. Участок для индексов цилиндра описывается с помощью отдельного оператора языка управления заданиями EXTENT (с типом участка 4).

Записи главного индекса описывают дорожку индекса цилиндров и содержат в поле данных адрес этой дорожки, а в поле ключа — наибольший ключ записи на цилиндре. Он также строится на отдельном участке диска, который должен обязательно непосредственно предшествовать участку индекса цилиндров. Он описывается отдельным оператором EXTENT (с типом участка 3).

При создании набора данных записи помещаются только в область данных. Область переполнения будет заполняться позднее при обновлении набора; сюда помещаются записи из области данных для того, чтобы освободить место для записей, которые будут добавляться в набор данных.

Общая структура индексно-последовательного набора данных имеет вид



Набор данных состоит из нескольких областей (областей прямого доступа), занимающих целое число цилиндров. При создании набора записи помещаются в него по возрастанию логических ключей. Создание набора начинается с области данных первого цилиндра, назначенного для него. Для каждой дорожки имеется запись с максимальным ключом. Этот ключ заносится в индекс дорожки (при создании набора данных области ключей индекса имеют одинаковое значение). После заполнения цилиндра максимальный ключ записи, которая находится в области данных, записывается в индекс цилиндра. Если набор данных велик и индекс цилиндра занимает много дорожек (более трех), программист может указать системе о необходимости создания главного индекса (режим INDEX MULTIPLE).

Расширение набора данных происходит путем внесения новых записей на дорожку в соответствии с ее логическим ключом. При этом запись с наибольшим на дорожке ключом может быть вытеснена с этой дорожки и перенесена в область переполнения на первое свободное место. Одновременно происходят необходимые изменения в индексах.

Имеются два типа областей переполнения:

— область переполнения цилиндра, для которой отводится несколько последних дорожек каждого цилиндра. Для одного цилиндра обычно достаточно двух дорожек переполнения. Количество дорожек переполнения устанавливается с помощью режима OFLTRACKS атрибута ENVIRONMENT;

— независимая область переполнения, которая является дополнительной областью на дисках и описывается в отдельном операторе EXTENT, в котором значение операнда «тип участка» равно 2. Участок для этой области может располагаться либо перед цилинд-

рами, либо после цилиндров, отведенных для записей набора данных.

Перед созданием индексно-последовательных наборов данных записи должны быть рассортированы в логически возрастающей последовательности их ключей. Несмотря на то, что эти наборы могут быть построены только последовательно, в операторе всегда должен быть указан режим KEYFROM. Причина этого в том, что при выводе в область ключа должно быть записано его значение. Этот ключ указывается значением выражения в режиме KEYFROM.

Пример. Необходимо создать индексно-последовательный набор данных, содержащий некоторые сведения о сотрудниках предприятия. В качестве признака упорядочивания должна быть использована фамилия сотрудника (SNAME). Исходные данные о сотрудниках находятся в последовательном наборе на магнитной ленте. Они упорядочены по фамилиям сотрудников. Длина записи 80 байтов, в блоке 10 записей.

Записи данных наряду с фамилиями и табельными номерами должны содержать информацию, состоящую из 54 символов. Последняя запись на ленте содержит в позициях имени последовательность символов'* ENDE *'.

Для распознавания одинаковых фамилий разных сотрудников в ключе после фамилии располагается счетчик (переменная COUNT).

Индексно-последовательный набор данных размещается на трех цилиндрах, на каждом из которых отведено по две дорожки для области переполнения.

Программа создания файла INDEXED будет иметь вид

```
PR11:   PROCEDURE OPTIONS (MAIN);
        DECLARE      TAPE FILE RECORD INPUT
          ENV (F (800, 80) MEDIUM (SYS0014, 5010));
        DECLARE      INDSE FILE RECORD OUTPUT KEYED
          SEQUENTIAL ENV (F (80) MEDIUM (SYS015, 5056)
          KEYLENGTH (26) EXTENTNUMBER (2)
          OFLTRACKS (2) INDEXED),
          1 REF,
            2 FAMILY,
            3 SNAME CHAR (25),
            3 COUNT PICTURE '9',
            2 INF CHAR (54),
          PROV CHAR (26);
        OPEN FILE (TAPE), FILE (INDSE);
    M:  READ FILE (TAPE) INTO (REF);
        IF SNAME='* ENDE*' THEN GOTO M1;
        PROV=STRING (FAMILY);
        WRITE FILE (INDSE) FROM (REF) KEYFROM (PROV)
        GOTO M;
    M1: CLOSE FILE (TAPE), FILE (INDSE),
        END PR11;
```

В данном примере индексно-последовательный набор данных содержит неблокированные записи. При этом фамилия сотрудника с отличительным признаком (подструктура FAMILY) содержится как в самой записи, так и в ключе. Однако, как было указано выше, индексно-последовательный набор данных может содержать и блокированные записи. В этом случае ключ располагается только внутри записи, а его начальная позиция в ней определяется режимом KEYLOG.

Для создания набора данных с блокированными записями (10 записей в блоке) нашем примере необходимо объявить файл следующим образом:

```
DCL IND FILE RECORD OUTPUT SEQUENTIAL
        KEYED ENV (F (800, 80) INDEXED
        MEDIUM (SYS015, 5056) KEYLOG (1)
        KEYLENGTH (26) OFLTRACKS (2)
        EXTENTNUMBER (2));
```

Чтение записей индексно-последовательного набора данных производится двумя способами:

— прямым методом доступа (атрибуты объявления файла DIRECT INPUT или UPDATE) с помощью оператора:
READ FILE (имя файла) INTO (переменная) KEY (выражение);

В этом случае из набора данных читается и передается в область памяти переменной, указанной в режиме INTO, та запись, ключ которой определен выражением режима KEY;

— последовательным методом доступа (атрибуты файла SEQUENTIAL INPUT или UPDATE) с помощью оператора:
READ FILE (имя файла) INTO (переменная) [KEY (выражение)];

В данном случае с помощью режима KEY может быть найдена и прочитана запись, ключ которой указан в этом режиме, а использование оператора READ без режима KEY позволяет последовательно читать все остальные записи набора данных.

Пример.

```
          .........
READ FILE (F1) INTO (ST) KEY (PROV);
          .........
            /* обработка найденной записи */
M: READ FILE (F1) INTO (ST);
   .........
              /* последовательная обработка всех остальных записей
                 набора */
           GOTO M;
```

Если оператор READ сразу используется без режима KEY, то осуществляется последовательное чтение набора данных, начиная с его первой записи.

Может применяться оператор ввода в следующей форме:
READ FILE (имя файла) INTO (переменная) KEYTO (переменная);

Использование такой формы оператора READ позволяет при каждом чтении записи из набора данных считывать ее ключ в область памяти переменной, указанной в режиме KEYTO. Сравнивая его с заранее известным ключом, можно ограничить число читаемых записей.

Рассмотрим на следующем примере использование оператора с режимом KEYTO.

Пример. Применяя последовательный доступ, необходимо извлечь из индексно-последовательного набора данных сведения о сотрудниках и напечатать все записи о сотрудниках, фамилии которых начинаются с определенной буквы (соответствующая начальная буква должна быть прочитана с перфокарты).

Программа имеет вид

```
PR13:    PROCEDURE OPTIONS (MAIN);
          DECLARE  INDSE FILE RECORD INPUT KEYED
        SEQUENTIAL ENV (F (800, 80) MEDIUM (SYS015,
        5056) KEYLOG (1) KEYLENGTH (26)
        EXTENTNUMBER (2) INDEXED),
        CELL CHAR (80),
        LETTER CHAR (1),
        PROV CHAR (26);
M: OPEN FILE (IN )S );
     GET EDIT (LETTER) (A(80));
     READ FILE (INDSE) INTO (CELL) KEYTO (PROV);
     DO WHILE (SUBSTR (PROV, 1, 1) = LETTER);
     READ FILE (INDSE) INTO (CELL) KEYTO(PROV);
     END;
     DO WHILE (SUBSTR (PROV, 1,1) = LETTER);
         PUT EDIT (CELL) (SKIP, A (80));
         READ FILE (INDSE) INTO (CELL) KEYTO (PROV);
     END;
     GOTO M;
     CLOSE FILE (INDSE);
   END PR13;
```

Чтение начинается с начала файла, так как первая запись, ключ которой начинается с определенной буквы, неизвестна. После выполнения первого оператора READ первая запись набора данных будет прочитана в область переменной CELL, а ключ этой записи—в область переменной PROV. Затем с помощью встроенной функции SUBSTR будет выделен первый символ ключа (фамилия сотрудника) и проверен на неравенство с введенным значением переменной LETTER. Таким образом с помощью выражения в режиме WHILE организован циклический процесс последующего чтения записей набора данных.

Индексно-последовательные наборы данных позволяют как последовательное, так и произвольное обновления записей. При этом имя файла должно быть объявлено с атрибутом UPDATE.

При необходимости заменить большое количество записей без добавления новых наиболее эффективен последовательный доступ к записям набора данных.

При последовательном обновлении новые записи должны вводиться в логически возрастающей последовательности их ключей. Перед заменой запись должна быть предварительно прочитана в основную память оператором READ (с режимом KEY или без него).

Для помещения обновленной записи в набор данных используется оператор REWRITE в следующей форме:

REWRITE FILE (имя файла);

Пример.

```
        .....
M: READ FILE (F1) INTO (ST) KEY (PROV);
        .....
     /* корректировка записи */
     REWRITE FILE (F1);
     GOTO M;
```

При обновлении набора данных прямым методом доступа для чтения записи используется оператор READ с режимом KEY, а для

помещения записи в набор данных операторы:

WRITE FILE (имя файла) FROM (переменная) KEYFROM (выражение);

REWRITE FILE (имя файла) FROM (переменная) KEY (выражение);

Рассмотрим применение этих операторов на следующем примере.

Пример. В программе индексно-последовательный набор данных должен быть обработан таким образом, чтобы выполнялось следующее:

— указав фамилию сотрудника, вывести на печать информацию о нем;

— учесть возможности расширения набора данных новыми записями (за счет вновь принятых сотрудников);

— в набор данных должны вноситься изменения об отдельном сотруднике (например, изменение адреса или занимаемой должности).

Вся необходимая информация должна читаться с перфокарт. При этом используются четыре различные управляющие перфокарты, которые в первых колонках содержат следующие знаковые строки:

— 'LOOK' указывает, что следующие перфокарты содержат только фамилии сотрудников, данные о которых надо найти;

— 'ADDITION' указывает, что перфокарты последовательности содержать полные записи данных с информацией о новых сотрудниках;

— 'CHANGE' указывает, что перфокарты последовательности содержат полные записи данных с новой информацией о сотрудниках;

— '* ENDE *' указывает, что перфокарты данных дальше не следует.

Порядок, в котором могут появиться управляющие перфокарты с соответствующими им последовательностями перфокарт, произвольный.

Программа обновления файла INDEXED при помощи прямого метода доступа будет иметь вид

```
PR14:   PROCEDURE OPTIONS (MAIN);
        DECLARE      IND FILE RECORD UPDATE DIRECT
             KEYED ENV (F (800,80) MEDIUM (SYS015,
            5056) EXTENTNUMBER (2)  KEYLENGTH (26)
            OFLTRACKS (2) INDEXED),
            1 REF,
              2 FAM,
              3 SNAME CHAR (25),
              3 COUNT PICTURE '9',
            2 INF CHAR (54),
            MARK LABEL;
        OPEN FILE (IND);
M:      GET EDIT (REF) (A (25), F (1), A (54));
        IF SNAME=' LOOK' THEN
           DO; MARK=M1; GOTO M; END;
        IF SNAME='ADDITION' THEN
           DO; MARK = M2; GOTO M; END;
        IF SNAME=' CHANGE' THEN
            DO; MARK = M3; GOTO M; END;
        IF SNAME='* ENDE *' THEN GOTO M4;
             GOTO MARK;
M1:     READ FILE (IND) INTO (REF) KEY  (STRING (FAM));
        PUT EDIT (SNAME, INF) (SKIP, X (10), A, X (5), A);
        GOTO M;
M2:     WRITE FILE (IND) FROM (REF)  KEYFROM
        (STRING (FAM));
        GOTO M;
M3:  REWRITE FILE (IND) FROM (REF) KEY (STRING (FAM));
M4:  CLOSE FILE (IND);
     END PR14;
```

Глава 9

ИСПОЛЬЗОВАНИЕ БАЗИРОВАННЫХ ПЕРЕМЕННЫХ

9.1. Общие понятия о базированной переменной и указателе

Как известно, распределение памяти для используемых в программе переменных производится транслятором на основании атрибутов, указанных программистом при их объявлении. Однако объявление переменной, как правило, не имеет отношения к ее адресу в памяти, т. е. программист знает, какой объем памяти будет выделен для каждой переменной, но адрес области памяти ему неизвестен. В большинстве случаев в этом и нет необходимости. Тем не менее, в языке ПЛ/1 имеется средство, с помощью которого программист может непосредственно управлять размещением переменных в основной памяти машины, т. е. указывать транслятору, где он должен располагать некоторые переменные. Таким средством управления является переменная, называемая указателем. Присваивая указателю определенный адрес основной памяти, программист может распределить память для переменной, связанной с этим указателем.

Переменные, расположение которых в памяти машины определяется указателями, называются базированными переменными (класс памяти BASED). Каждая базированная переменная связывается со своим указателем с помощью явного объявления:

DECLARE идентификатор BASED (указатель);

Указатель также должен быть явно объявлен в программе с помощью атрибута POINTER (сокращенно PTR):

DECLARE идентификатор [(атрибут размерности)] POINTER;

Указатель принадлежит к классу данных управления программой. Допустимыми операциями для него являются только операции сравнения = и ¬ = .

Прежде чем ссылаться на базированную переменную, необходимо присвоить значение связанному с ней указателю. Это можно сделать одним из следующих способов:

— использованием встроенной функции ADDR;
— использованием встроенной функции NULL;
— другим указателем, который уже имеет значение;
— использованием псевдопеременной UNSPEC;
— использованием оператора LOCATE или оператора READ с режимом SET.

Напомним, что встроенная функция ADDR (X) возвращает значение, которое является адресом переменной, заданной в качестве аргумента. Аргумент X может быть скалярной переменной, индексированной переменной, элементом структуры, массивом или структурой. Это данное может быть любого типа и иметь любой класс памяти.

Присваивая указателю с помощью этой функции адрес размещения некоторой переменной, мы назначаем это место для размещения базированной переменной, связанной с этим указателем. Использование базированной переменной таким образом аналогично наложению одной переменной на другую с помощью атрибута DEFINED с той разницей, что в процессе выполнения программы базированная переменная может занимать область памяти не одной (как при использовании атрибута DEFINED), а различных переменных.

Пример. Имеются переменные A и B, объявленные явно. В программе используется переменная C, которая может располагаться либо в области памяти переменной A, либо в облати памяти переменной B. Это можно реализовать с помощью базированной переменной следующим образом:

```
DECLARE (A, B) FIXED (5,2),
        C FIXED (5,2) BASED (Z),
        Z POINTER;
        .....
Z = ADDR (A); /*  УКАЗАТЕЛЬ ПОЛУЧАЕТ ЗНАЧЕНИЕ АДРЕСА
                  ПЕРЕМЕННОЙ A */
        .....
Z = ADDR (B); /* УКАЗАТЕЛЬ ПОЛУЧАЕТ ЗНАЧЕНИЕ АДРЕСА ПЕ-
                 РЕМЕННОЙ B */
```

В первом случае базированная переменная C, связанная с указателем Z, будет размещаться в области памяти A; во втором случае — в области памяти B.

Встроенная функция NULL используется для обнуления указателя. Она присваивает указателю значение, которое не является адресом какой-либо переменной в памяти, а только инициализирует указатель. Этим значением является шестнадцатиричное 00FFFFFF.

Пример. Если в условиях предыдущего примера не желательно, чтобы переменная C располагалась в области памяти B (т. е. не имела бы определенного адреса), то нужно использовать оператор

```
Z = NULL;
```

После выполнения этого оператора базированная переменная не будет иметь области памяти для размещения, т. е. на нее нельзя в дальнейшем ссылаться в программе.

Если указатель, связанный с одной базированной переменной, получил адрес с помощью встроенной функции ADDR, то и другие

базированные переменные могут быть размещены в области памяти этой же переменной. В этом случае значение адреса может быть присвоено указателю с помощью другого указателя, уже имеющего адрес.

Пример.

```
DCL A FIXED,
    B FIXED BASED (Z),
    C FIXED BASED (P);
.....;.....
Z = ADDR (A);
.....
P = Z;
```

После выполнения первого оператора присваивания в области памяти переменной A будет размещаться базированная переменная B, а после выполнения второго оператора там же будет размещаться базированная переменная C. Во втором случае указатель P получает значение адреса области памяти переменной A с помощью указателя Z, которому уже присвоено значение.

Псевдопеременная UNSPEC (X) позволяет в качестве аргумента использовать переменную типа указателя. В этом случае значение, которое присваивается псевдопеременной, должно представляться в виде строки битов. Эта строка битов и будет определять абсолютное значение адреса области в основной памяти, которое и присваивается указателю. Длина этой строки должна быть равна 32 битам (так как переменная типа указателя размещается в одном слове), хотя для указания адреса используются только первые 24 бита этой строки.

Базированной переменной, связанной с этим указателем, будет выделена область памяти, адрес которой присвоен указателю с помощью псевдопеременной UNSPEC.

Использование такого способа размещения базированной переменной необходимо применять осторожно, т.е. при указании абсолютного адреса программист должен быть уверен, что область памяти с этим адресом свободна.

Пример.

```
DCL A FIXED BASED (Z),
    Z POINTER;
.....
UNSPEC (Z) = '00000000000000000100000000000000'B;
```

Этим оператором указателю Z присвоено значение, которое указывает шестнадцатиричный адрес 4000. Ведущие нули в строке битов опускать нельзя, так как при выполнении оператора присваивания более короткая основная строка дополняется нулями справа.

Последний способ присваивания значения указателю с помощью операторов READ или LOCATE с режимом SET будет подробно рассмотрен в § 9.4.

Использование базированных переменных значительно расширяет возможности обработки информации и позволяет более гибко описывать сложные алгоритмы. К основным случаям применения базированных переменных следует отнести:

— обработку записей различных типов;
— моделирование массива структур;
— обработку записей в буфере ввода и вывода.

9.2. Обработка записей различных типов

При обработке записей различных типов можно использовать базированную переменную, что позволит уменьшить объем памяти программы и повысить эффективность их обработки.

Обработку записей разных типов с помощью базированной переменной рассмотрим на следующих примерах.

Пример. Допустим, что необходимо ввести и обработать 100 записей, имеющих различную структуру и находящихся в наборе данных на магнитной ленте. Длина каждой записи 80 байтов. Записи имеют два различных формата.

Формат записи первого типа:

INF			PR	TEXT	
INF1	INF2	INF3		TEXT1	TEXT 2
CHAR (31)	PIC '9999'	CHAR (16)	CHAR (1)	CHAR (18)	CHAR (10)

Формат записи второго типа:

INF			PR	ARITH			
INF1	INF2	INF3		Z1	Z2	Z3	Z4
CHAR (31)	PIC '9999'	CHAR (16)	CHAR (1)	PIC 'SS99V999'			

Как видно из форматов, структура записей отличается второй частью (после признака PR). Обработка записей заключается в следующем:

— первый тип записей выводится на печать без изменений;

— для записей второго типа информация второй части предварительно обрабатывается и затем выводится на печать. Обработка представляет собой суммирование переменных по формулам:

$$SUM\ (1) = Z1 * Z2;$$
$$SUM\ (2) = Z3 * 0.5 + Z4;$$

Решение данной задачи можно реализовать, используя базированную переменную (в нашем примере структуру ARITH). Программа в этом случае будет иметь вид

```
PR15:  PROCEDURE OPTIONS (MAIN);
       DECLARE VOL FILE RECORD INPUT
          ENVIRONMENT (F (80) MEDIUM (SYS010, 5056));
       DECLARE  P POINTER,
         1 LIST,
           2 INF,
             3 INF1 CHAR (31),
             3 INF2 PICTURE '(4)9'
```

```
      3 INF3 CHAR (16),
    2 PR CHARACTER (1),
    2 TEXT,
      3 TEXT1 CHAR (18),
      3 TEXT2 CHAR (10),
  1 ARITH BASED (P),
    2 Z1 PICTURE 'SS99V999',
    2 Z2 PICTURE 'SS99V999',
    2 Z3 PICTURE 'SS99V999',
    2 Z4 PICTURE 'SS99V999',
  SUM (2) FIXED (7,3);
OPEN FILE (VOL);
P = ADDR (TEXT);
DO I = 1 TO 100;
   READ FILE (VOL) INTO (LIST);
     IF PR =  'T' THEN/*запись формата 1*/
     PUT EDIT (INF, TEXT) (SKIP, 5 (X (3), A));
              ELSE/*запись формата 2*/
              DO; SUM (1) = Z1*Z2;
                  SUM (2) = Z3*0.5 + Z4;
  PUT EDIT (INF, SUM) (SKIP, 3 (X (3), A),
  2(X (4), F (9,3)));
              END;
END;
CLOSE FILE (VOL);
END PR15;
```

Присваивая указателю P адрес подструктуры TEXT, мы накладываем на область памяти этой переменной базированную переменную ARITH. Это позволяет нам в зависимости от значения признака PR по-разному обрабатывать считываемую запись набора данных.

Пример.

На перфокартах находятся данные произведенных измерений (максимум 10 измерений). В колонках 1—4 — номер измерения. Далее следуют результаты измерений: это целые четырехзначные числа и текст — последовательность знаков любой длины. Результат измерения распознается по стоящему впереди знаку « + ». Текст может состоять из любых знаков, за исключением « + ». Например: 0002 понедельник + 4402, вторник + 3401...

В результате обработки надо получить и напечатать сумму и среднее значение каждой группы измерений. Кроме того, необходимо напечатать номер измерения. Колода перфокарт с измерениями завершается перфокартой, которая в позициях 1—4 содержит символы 'ENDE'.

Решение.

```
PR16: PROCEDURE OPTIONS (MAIN);
      DECLARE  E (80) CHAR (1),
               PR CHAR (4) DEFINED E,
               Z (11) POINTER;
M:    GET EDIT (E) (A(80));
      IF PR = 'ENDE' THEN GOTO M1;
      K = 1;  Z = NULL;
      DO I = 5 TO 76;
        IF E (I) = ' + ' THEN
```

```
                DO; Z (K) = ADDR(E(I + 1));
                    K = K + 1;
                END;
            END;
            CALL UR (Z);
UR:         PROCEDURE (P);
            DECLARE  P (11) POINTER,
                     R  POINTER,
                     N  FIXED (2) BASED (R),
                     SUM  FIXED (8);
            SUM = 0;
            DO K = 1 BY 1 WHILE (P (K) ¬= NULL);
               R = P (K);
                 SUM = SUM + N;
               END;
  PUT SKIP EDIT (PR, SUM, SUM/K) (A, F (10)  F (10, 2));
  END UP;
M1: END PR16;
```

В данном примере использованы некоторые из указанных выше способов присвоения значения указателю, а именно:

— в операторе

```
Z = NULL;
```

происходит идентификация массива указателей с помощью встроенной функции NULL, что позволяет в процедуре UR определить условие для окончания обработки группы измерений;

— в операторе

```
Z (K) = ADDR(E(I + 1));
```

с помощью встроенной функции ADDR указатель устанавливается на адрес измерения, размещающегося за символом ' + ', что позволяет в процедуре UR обработать значения этих измерений;

— в операторе

```
R = P (K);
```

указатель R получает значение с помощью другого указателя P (K), который уже имеет значение.

9.3. Моделирование массива структур

В данном подмножестве ПЛ/1 нельзя создать массив, элементами которого были бы структуры. Но с помощью базированных переменных можно смоделировать массив структур. Метод построения таких структур рассмотрим на следующем примере.

Даны 80-символьные записи с информацией о среднемесячном количестве осадков (PRECIPIT) , средней температуре (TEMP) для какого-то района (PLACE) в определенном году (YEAR). Можно описать запись, применив массив, элементами которого являются структуры, как это показано ниже.

```
1 CONTROL,
    2 PLACE CHAR (6),
    2 YEAR PICTURE '99',
    2 VALUE (12),
      3 PRECIPIT PICTURE '999',
      3 TEMP PICTURE 'S99';
```

При описании структуры CONTROL использован элемент VALUE, который является массивом подструктур, что недопустимо в языке данного уровня. Учитывая, что элементы подструктуры PRECIPIT и TEMP, описанные с помощью атрибута PICTURE, имеют общую длину в 6 байтов, можно смоделировать подобную структуру следующим образом:

```
1 FACT,
    2 PLACE CHAR (6),
    2 YEAR PICTURE '99',
    2 VALUE (12) CHAR (6);
```

Образование каждого элемента массива VALUE происходит с помощью базированной переменной PATTERN, которая выглядит так:

```
1 PATTERN BASED(Z),
    2 PRECIPIT PICTURE '999',
    2 TEMP PICTURE 'S99';
```

Предположим, что из значений среднемесячных осадков и температур необходимо найти среднегодовое значение. Значения среднемесячных осадков и температур для каждого района вводятся с перфокарт. Признаком конца исходной информации является набор символов ******, набитых в первых шести позициях перфокарты. Программа реализации данного алгоритма имеет вид

```
PR17:  PROCEDURE OPTIONS (MAIN);
         1  FACT,
            2 PLACE CHAR (6),
            2 YEAR PICTURE '99',
            2 VALUE (12) CHAR (6),
         1  PATTERN BASED (Z),
            2 PRECIPIT PICTURE '999',
            2 TEMP PICTURE 'S99',
         Z  POINTER,
         YDP FIXED (7,2),
         YDT FIXED (7,2);
       PUT EDIT ('МЕСТО', 'ГОД', 'СР. КОЛИЧ. ОСАДКОВ',
            'СРМЕС. ТЕМПЕРАТУРА') (SKIP, X(10), A, X(5),
              A, COLUMN (25), A, COLUMN (45), A);
M:     GET EDIT (FACT) (A (6), F (2), 12A(6));
       IF PLACE = '******' THEN GOTO M1;
       YDP = 0; YDT = 0;
       DO I = 1 TO 12;
          Z = ADDR (VALUE (I));
          YDP = YDP + PRECIPIT;
          YDT = YDT + TEMP;
       END;
```

```
        YDP = YDP/12;  YDT = YDT/12;
        PUT EDIT (PLACE, YEAR, YDP, YDT) (SKIP, X (10),
             A, X (5), F (2), COLUMN (25), F (9,2), COLUMN (45),
             F (9,2));
        GOTO M;
M1 :    END PR17;
```

9.4. Обработка данных в буферах ввода и вывода

Ранее мы рассматривали операторы ввода — вывода, ориентированного на записи, которые вводили значения переменных в область памяти, выделенную для этих переменных, или выводили ее в набор данных. Для экономии места в памяти, используемого для размещения данных, и ускорения процесса их обработки целесообразно эту обработку проводить в буферах ввода или вывода с помощью базированных переменных. Такую обработку записей в самом буфере (или буферах) можно осуществлять для последовательного набора данных, описанного с атрибутами RECORD SEQUENTIAL INPUT (OUTPUT или UPDATE). Однако для такой обработки программист должен иметь возможность каким-то образом указывать место размещения записи в буфере. Это осуществляется следующим образом:

— адрес области памяти буфера, в которой находится запись, присваивается переменной типа указателя;

— структура записи, размещенной в буфере, описывается в программе как базированная переменная, связанная с этим указателем.

Основное требование при обработке записей в буфере предъявляется к их длине:

— для блокированных записей фиксированной длины (указанной в описании файла) длина блока должна без остатка делиться на 8;

— для записей переменной длины — этот остаток при делении должен иметь длину 4 байта.

При обработке записей в буфере ввода оператор ввода имеет следующий формат:

READ FILE (имя файла) SET (указатель);

При вводе неблокированных записей этот оператор указывает, что очередная запись набора данных прочитана в буфер ввода и начальный адрес ее в буфере присвоен указателю, имя которого стоит в круглых скобках за режимом SET. При вводе блокированных записей он определяет, что значение указателя установлено на адрес начала очередной записи внутри буфера.

При помощи базированной переменной, связанной с этим указателем, можно осуществить обращение к различным элементам вводимой записи. При этом базированная переменная практически представляет собой описание составных частей (структуры) записи и применяется так, как будто при помощи атрибута DEFINED про-

изведено наложение областей памяти данной структуры на область памяти буфера ввода.

Применение базированной переменной или ее элементов (если речь идет о структуре) в качестве операнда в каком-либо операторе означает, что идет обращение к соответствующему участку области памяти, отведенной буферу ввода. При этом несколько базированных переменных могут быть связаны с одним и тем же указателем и определяться на одно и то же место в памяти, отведенной буферу ввода. При вводе записей с различной структурой можно описать структуры для каждого типа вводимых записей, как базированные переменные, и связать их с одним и тем же указателем. При этом в каждой вводимой записи нужно определить некоторый признак, указывающий тот или иной тип ее строения, с помощью которого можно использовать соответствующую структуру для обработки этой записи.

Пример. Производится обработка записей различной структуры набора данных с именем SF, расположенного на МЛ в буфере ввода Записи набора данных в зависимости от признака (PR) имеют длину 20 символов (значение признака A) или 18 символов (значение признака B). Они считываются в область ввода и после проверки на признак обрабатываются требуемым образом в соответствии с их строением.

```
PR19: PROCEDURE OPTIONS (MAIN);
      DECLARE SF FILE RECORD INPUT
              ENV (V (28) MEDIUM (SYS009, 5010));
      DECLARE Z POINTER,
              1 INF1 BASED (Z),
                2 PR CHAR (1),
                2 A (3) FIXED (7,2),
                2 B CHAR (2),
                2 C CHAR (1),
                2 K FIXED (7),
              1 INF2 BASED (Z),
                2 PR CHAR (1),
                2 U CHAR (8),
                2 V (2) FIXED (5,2),
                2 W FIXED (5);
      ON ENDFILE (SF) GOTO M1;
      OPEN FILE (SF);
M:    READ FILE (SF) SET (Z);
      IF PR = 'A' THEN
      DO;...
      /*операторы обработки структуры INF1*/
      .....
      END;
          ELSE
      DO;...
      /* операторы обработки структуры INF2*/
      .....
      END; GOTO M;
      M1 : CLOSE FILE (SF);
      END PR19;
```

Оператор READ с режимом SET можно эффективно использовать при обработке последовательных файлов с атрибутом UPDATE.

Это дает возможность экономить память за счет рабочей области. Для этого необходимо прочитанную в буфер и в случае необходимости обработанную запись сразу же записать обратно.

При этом содержимое буфера записывается обратно в набор данных с помощью оператора REWRITE без режима FROM, т. е. REWRITE FILE (имя файла). Этот оператор обращается к предварительно считанной, а затем обработанной записи в буфере и записывает ее на прежнее место в набор данных.

Для создания записей в буфере и их вывода в набор данных используется специальный оператор LOCATE в следующей форме:

LOCATE базированная переменная FILE (имя файла) SET (указатель).

Данный оператор указывает, что заданная базированная переменная является базисом области памяти в буфере вывода при создании файла. Связь базированной переменной с конкретным адресом в буфере осуществляется указателем, использованным в режиме SET.

Последующее присвоение значения базированной переменной или ее элементам приводит к тому, что эти значения запоминаются в буфере и, следовательно, образуется выводная запись и подготавливается вывод. При этом несколько базированных переменных могут быть связаны с одним и тем же указателем, что позволяет при создании записей в буфере образовывать различные структуры записей. В случае применения двух буферов вывода указатель последовательно принимает значения адресов начала этих буферов.

Вывод созданных записей на внешний носитель при неблокированных записях осуществляется с помощью последующих операторов LOCATE, WRITE или CLOSE (при закрытии файла). При блокированных записях вывод осуществляется оператором LOCATE, который освобождает буфер при полностью заполненном блоке и связывает базированную переменную с новым местом памяти буфера.

Применение обработки записи в буфере эффективно прежде всего для наборов данных с очень большими записями (таблицами), а также для наборов данных с большим количеством записей (при этом отсутствует многократная передача данных из буфера в область памяти соответствующей переменной и обратно).

Примеры.

1) Необходимо произвести изменения в определенных записях последовательного набора данных на магнитном диске SF в зависимости от номера этой записи. Номера соответствующих записей и переменная, которая должна быть изменена, считываются с перфокарт (набор данных с именем CARD). Обработка записей набора SF должна производиться в буфере.

```
PR20 : PROCEDURE OPTIONS (MAIN);
    DECLARE SF FILE RECORD UPDATE SEQUENTIAL
            ENV (F (432,36) MEDIUM (SYS013, 5056));
    DECLARE CARD FILE INPUT ENV (F (80) MEDIUM (SYSIPT,
                    6012));
        DECLARE 1 INF BASED (Z),
```

```
                        2 A FIXED (5),
                        2 B CHAR (15),
                        2 G CHAR (10),
                        2 R FIXED (4),
                        2 E FIXED (9,2),
                    Z POINTER,
                    NR FIXED (5),
                    W FIXED (9,2);
  ON ENDFILE (CARD) GOTO M2;
  OPEN FILE (SF);
M: GET FILE (CARD) EDIT (NR, W) (F (5), F (9,2));
M1:     READ FILE (SF) SET(Z);
      IF A = NR THEN
         DO; E = E + W;
             REWRITE FILE (SF); GOTO M;
           END;
           ELSE GOTO M1;
M2 : CLOSE FILE (SF);
   END PR20;
```

2) Необходимо построить набор данных из записей двух различных типов (X1 и X2); содержимое которых в произвольной последовательности считывается вместе с признаком их структуры (переменная KK) с двух следующих друг за другом перфокарт. Выводимые записи должны быть созданы внутри буфера вывода.

```
      PR21: PROCEDURE OPTIONS (MAIN);
      DECLARE SF FILE RECORD OUTPUT
              ENV (V (28) MEDIUM (SYS011, 5056));
      DECLARE CARD FILE INPUT ENV (F (80) MEDIUM (SYSIPT,
                  6012));
      DECLARE 1  X1 BASED (P),
                  2 KZ1 CHAR (1),
                  2 A FIXED (7),
                  2 B CHAR (10),
                  2 C FIXED (5,2),
                  2 D FIXED (11,4),
                1 X2 BASED (P),
                  2 KZ2 CHAR (1),
                  2 U CHAR (5),
                  2 V FIXED (7,1),
                  2 W CHAR (2),
                  2 Z FIXED (5,0),
                 P POINTER, EMPTY CHAR (1),
                 KK CHAR (1);
      ON ENDFILE (CARD) GOTO M1;
      OPEN FILE (SF);
   M : GET FILE (CARD) EDIT (KK, EMPTY) (A (1), X (78), A (1));
      IF KK = '1' THEN
          DO; LOCATE X1 FILE (SF) SET (P);
              GET FILE (CARD) EDIT (A, B, C, D, EMPTY)
              (F (7), A (10), F (5,2) F (11,4), X (55), A (1));
      KZ1 = KK; GOTO M;
          END;
          ELSE
           DO; LOCATE X2 FILE (SF) SET (P);
               GET FILE (CARD) EDIT (U, V, W,
               Z, EMPTY) (A (5), F (7,1), A (2), F (5), X (64), A (1));
       KZ2 = KK; GOTO M;  END;
       M1 : CLOSE FILE (SF);
       END PR21;
```

По первому оператору GET будет прочитана в буфер ввода перфокарта с записью, имеющей формат X1 или X2. Переменной KK (для которой выделена область памяти в один байт) будет присвоено значение признака формата (в нашем примере для формата X1 символ '1', для X2 — любой другой). После проверки признака в операторе IF с помощью оператора LOCATE в буфере вывода резервируется область памяти для записи той или иной структуры. Указатель P связывает эту область с определенной структурой записи, после чего можно использовать соответствующие элементы структур в списке данных последующих операторов GET.

Переменная EMPTY позволяет ограничить каждую вводимую запись размерами одной перфокарты, т. е. дает возможность каждую структуру и каждый признак набивать на отдельной перфокарте.

Перенесение заполненного буфера вывода на внешний носитель осуществляется здесь при соответствующем обращении к оператору LOCATE. Последняя запись будет выведена при выполнении оператора CLOSE.

3) Необходимо прочитать 100 записей файла PLATE длиной 80 байтов и после обработки записать их обратно. Обработка заключается в том, что записи, которые начинаются с символа 'A', должны быть заменены пробелами

```
PR22: PROCEDURE OPTIONS (MAIN);
      DECLARE PLATE FILE RECORD UPDATE
                        ENV (F (80) MEDIUM (SYS009, 5056));
      DECLARE 1 BV BASED (Z),
                2 TEST CHAR (1),
                2 INF CHAR (79),
                Z POINTER;
      OPEN FILE (PLATE);
      DO I = 1 TO 100;
         READ FILE (PLATE) SET (Z);
         IF TEST = 'A' THEN DO; INF = '-';
         REWRITE FILE (PLATE); END;
      END; CLOSE FILE (PLATE);
      END PR22;
```

Следует напомнить, что если файл с атрибутом UPDATE читается оператором READ с режимом SET, то содержимое буфера можно записать обратно на внешний носитель оператором REWRITE без режима FROM

4) Необходимо прочитать в рабочую область 100 записей длиной по 80 байтов и, объединив их в блоки (блок содержит 10 записей), записать на ленту. Если запись начинается с символа A, она должна быть заменена пробелами. Программа будет иметь вид

```
PR23:    PROCEDURE OPTIONS (MAIN);
         DECLARE TAPE FILE RECORD OUTPUT
               ENV (F (800,80) MEDIUM (SYS 010, 5010) BUFFERS (2)),
         1 BV BASED (Z),
           2 TEST CHAR (1),
           2 INF CHAR (79),
             Z POINTER;
OPEN FILE (TAPE);
DO I = 1 TO 100;
     LOCATE BV FILE (TAPE) SET (Z);
     GET EDIT (BV) (A (1), A (79));
     IF TEST = 'A' THEN INF = '-';
END; CLOSE FILE (TAPE);
END PR23;
```

Первый блок в 10 записей будет записан на ленту, когда оператор LOCATE будет выполняться в 11-й раз. Следующий блок — при 21-м выполнении и т. д. Вывод последнего блока будет осуществляться оператором CLOSE.

Глава 10

ПРОГРАММНЫЕ ПРЕРЫВАНИЯ

10.1. Причины программных прерываний и их характеристика

При выполнении программы может произойти прерывание, если в ней возникнет так называемая исключительная ситуация. В общем случае она возникает, если оператор программы, написанный на языке ПЛ/1, не выполним. Это происходит из-за того, что обрабатываемая оператором информация находится в противоречии с заданными в программе условиями или с требованиями операционной системы.

Важнейшие исключительные ситуации, при которых необходимо вмешательство программиста, обрабатываются с помощью средства языка ПЛ/1. Причины таких исключительных ситуаций в языке обозначаются специальными именами. Программное прерывание регистрируется операционной системой, и управление передается программе на ПЛ/1. Вмешательство программиста происходит в отдельной части программы (в той части, в которой обрабатывается ситуация, вызвавшая прерывание) — в программе обработки ситуаций. Если необходимо, прерывание можно игнорировать, тогда при возникновении исключительной ситуации программа будет выполняться дальше, без обработки этой ситуации. Игнорирование прерываний программы происходит с помощью так называемых префикс-ситуаций, которые определяют для отдельного оператора или целого блока, должно ли произойти прерывание при возникновении исключительной ситуации. Если прерывание не игнорируется, то язык ПЛ/1 предоставляет для каждой ситуации несколько возможностей ее обработки. Программист может написать собственную программу обработки ситуации и обработать ее с помощью специального оператора ON. Если такая программа не составлена, то выполняется стандартная программа операционной системы для возникшей ситуации.

В зависимости от причины возникновения исключительные ситуации делятся на три группы:

— вычислительные ситуации;
— ситуации ввода — вывода;
— ситуации действия системы.

Вычислительные ситуации возникают во время выполнения вычислений и присваивания значений. К ним относятся: CONVERSION, FIXEDOVERFLOW, OVERFLOW, UNDERFLOW, SIZE и ZERODIVIDE.

Все вычислительные ситуации, кроме SIZE, всегда включены, т. е. при их возникновении во время выполнения программы будет происходить прерывание программы. Ситуацию SIZE при необходимости можно включать с помощью префикс-ситуации.

Результат операции, при выполнении которой возникла вычислительная ситуация, как правило, не определен. Если программист не предусмотрел в программе определенного действия на соответствующее программное прерывание, то система печатает сообщение об ошибке и вызывает стандартное действие системы (ситуацию ERROR).

Ситуация CONVERSION возникает при попытке выполнить запрещенное преобразование данного типа строки символов. Это может произойти во время внутренней обработки данного (вычисления выражения или присваивания значений) или во время выполнения операции ввода — вывода потоком (нарушение внешнего представления вводимой или выводимой переменной; например, при вводе с помощью оператора GET десятичного числа в строке символов, взятых из потока между цифрами, появился запрещенный символ).

Ситуация FIXEDOVERFLOW возникает, когда разрядность результата арифметической операции с фиксированной точкой превышает допустимый максимум. Для десятичных данных максимум 15 цифр, для двоичных — 31 цифра. Результат операции с фиксированной точкой, при которой возникла эта ситуация, не определен.

Ситуация OVERFLOW возникает, когда результат операции с плавающей точкой превышает допустимый максимум. Этот максимум равен для десятичных чисел $7.237 * 10^{75}$, для двоичных 2^{252}.

Ситуация UNDERFLOW возникает, когда результат операции с плавающей точкой меньше допустимого минимума. Этот минимум равен для десятичных чисел $5.4 * 10^{-78}$, для двоичных 2^{-260}. Данная ситуация не возникает, если вычитаются равные числа. Результату операции, при которой возникла ситуация UNDERFLOW, присваивается значение, равное нулю, и с этим результатом продолжается выполнение программы (если программистом не предусмотрены какие-то специальные действия).

Ситуация SIZE возникает, когда при выполнении операции присваивания или операции ввода — вывода, в которой участвуют данные с фиксированной точкой, происходит потеря старших значащих разрядов (десятичных или двоичных) или знака минус. Это может произойти во время преобразования данных, когда разрядность вычисленного значения переменной превышает объявленную или подразумеваемую по умолчанию разрядность этой переменной.

Ситуация ZERODIVIDE возникает при попытке деления данных на нуль. Эта ситуация возникает как при делении данного с фиксированной точкой, так и при делении данного с плавающей точкой.

Ситуации ввода — вывода могут возникать при выполнении операций ввода — вывода. К этим ситуациям относятся: ENDFILE, ENDPAGE, KEY, RECORD, TRANSMIT.

Все ситуации ввода — вывода всегда включены и не могут быть выключены с помощью префикс-ситуаций. Если программист не предусмотрел действия по обработке этих ситуаций, то после прерывания программы система печатает сообщение об ошибке и выполняет стандартное действие (ситуация ERROR).

Ситуация ENDFILE (имя файла) возникает при выполнении операторов GET или READ при попытке читать запись после окончания файла, т. е. после того, как была прочитана последняя запись. Она возникает только при обращении к файлам с последовательной организацией (атрибуты SEQUENTIAL или SEQUENTIAL INDEXED). После того как произойдет ситуация ENDFILE, файл нужно закрыть.

Ситуация ENDPAGE (имя файла) возникает при попытке с помощью оператора PUT напечатать новую строку за пределами текущей страницы. Размер страницы может быть стандартным или указанным в режиме PAGESIZE оператора OPEN. Если действие программиста не задано, то после прерывания программы печать начнется с новой страницы.

Ситуация KEY (имя файла) может возникнуть только во время передачи записей с использованием ключей.

Для файлов REGIONAL (1) она возникает, если задан слишком большой номер области или исходный ключ содержит недопустимые символы.

Для файлов REGIONAL (3) эта ситуация может возникнуть в следующих случаях:

— при записи данных (оператор WRITE);

— если задан слишком большой номер области или заполнена вся дорожка;

— при выполнении операторов READ или REWRITE;

— если не найдена запись с указанным ключом или указан слишком большой номер области (вне участка, отведенного для набора данных);

— если последние 8 позиций исходного ключа содержат не цифровые символы (для любого оператора ввода — вывода).

Для файлов INDEXED ситуация KEY может возникнуть в следующих случаях:

— если ключи помещаемых записей файла SEQUENTIAL не указываются в возрастающей последовательности (операторы READ или WRITE) или область данных уже заполнена;

— при выполнении оператора WRITE (для файлов с атрибутом DIRECT), если в наборе данных уже есть запись с указанным исходным ключом или полностью заполнена область переполнения;

— при выполнении операторов READ или REWRITE, если не найдена запись с указанным ключом.

Ситуация RECORD (имя файла) может возникнуть только при передаче данных, ориентированной на записи (атрибут RECORD), в следующих случаях:

— длина записи больше длины переменной (для всех форматов F, V или U). В этом случае избыточные данные в записи теряются при чтении и не определены при выводе;

— длина записи меньше длины переменной (для формата F). При этом избыточные данные в переменной не передаются при выводе в набор данных и не изменяются при чтении.

Ситуация TRANSMIT (имя файла) может возникнуть во время любой операции ввода — вывода. Ее причиной является постоянная ошибка передачи, и, как результат, любые передаваемые данные являются потенциально неправильными.

Ситуация действия системы ERROR выполняет стандартное действие при возникновении прерывания программы. Эта ситуация всегда включена, и ее нельзя выключить с помощью префикс-ситуации. Если программистом для этой ситуации не задано действие, то система печатает сообщение, прекращает выполнение программы и передает управление супервизору операционной системы.

Эта ситуация может возникнуть в следующих случаях:

— если программист не указал действия для любой вычислительной ситуации или ситуации ввода — вывода;

— если при выполнении программы произошла ошибка, для которой нет имени исключительной ситуации.

10.2. Префикс-ситуации

Для включения или выключения любой вычислительной ситуации используется так называемая префикс-ситуация. С помощью этого средства программист устанавливает, должно ли после возникновения исключительной ситуации происходить прерывание программы или нет. Если указанная вычислительная ситуация включена, то после ее возникновения делается попытка продолжить выполнение программы. Перед именем вычислительной ситуации, указанной в префиксе, может стоять отрицание NO, которое обозначает, что данная ситуация выключена в области действия префикс-ситуации.

Префикс-ситуация представляет собой список одной или нескольких имен вычислительных ситуаций, заключенных в круглые скобки и отделенных от оператора или метки оператора двоеточием. Имена ситуаций, стоящих в списке, разделяются между собой запятыми. Формат префикс-ситуации имеет вид

(вычислительная ситуация [, вычислительная ситуация]...):

Пример.

```
DCL A FIXED (3,2);
.....
(SIZE): A = A * 10;
```

Здесь (SIZE): — префикс-ситуация. Прерывание программы произойдет при выполнении оператора присваивания, если значение переменной A превысит величину 9,99, т. е. произойдет потеря старшего разряда результата вычисления.

Область действия префикс-ситуации — это та часть программы, на выполнение которой распространяется действие указанной префикс-ситуации. Обычно это оператор или блок, которому предшествует префикс-ситуация. К области действия префикс-ситуации не относятся внешние подпрограммы и функции, которые могут быть вызваны при выполнении этого оператора.

Префикс-ситуации можно использовать перед следующими операторами: PROCEDURE, BEGIN, DO (кроме первого типа), IF, оператором присваивания, GET, PUT, OPEN с режимом PAGESIZE, DISPLAY, CALL и RETURN.

Область действия префикс-ситуации, указанной перед операторами PROCEDURE или BEGIN, распространяется на все операторы блока. Она включает также все вложенные блоки. Действие префикс-ситуации не распространяется только на процедуры, лежащие вне блока, которые могут быть вызваны из данного блока. Поэтому и предусматривается возможность ставить префикс-ситуацию перед оператором CALL при вызове этих процедур.

Включение или выключение ситуаций перед началом блока может быть изменено (переопределено) для внутренних блоков. В этом случае область действия префикс-ситуации будет, в свою очередь, распространяться на этот блок и вложенные в него блоки. Когда управление выходит из области ее действия, переопределение теряет силу.

Пример.

```
(SIZE): FUN: PROCEDURE;
...
(NOSIZE): A: BEGIN;
            ....
            END A;
            ....
            B: BEGIN;
            ....
            END B;
          END FUN;
```

В данном примере префикс-ситуация перед оператором PROCEDURE включает ситуацию SIZE в процедурном блоке FUN, т. е. указывает, что, если при вычислениях в процедурном блоке возникнет ситуация SIZE, произойдет прерывание программы. Область действия этой префикс-ситуации распространяется и на блоки, вложенные в FUN (блок B), кроме тех блоков, для которых состояние этой ситуации переопределено. Так, внутри блока A ситуация SIZE выключена, т. е. прерывания не произойдет, если в блоке A возникнет эта ситуация.

Префикс-ситуация, указанная перед оператором IF, действует только при вычислении выражения, стоящего в этом операторе. Ее действие не распространяется на операторы во фразах THEN и ELSE. Поэтому префикс-ситуации, которые должны относиться к операторам во фразах THEN или ELSE, должны быть записаны непосредственно перед этими операторами.

Пример.

```
(ZERODIVIDE): IF I > A/ (B — C) THEN X = M/ (N — Q);
              ELSE (ZERODIVIDE): Y = P/ (L — M);
```

В данном случае прерывание программы будет обрабатываться только в выражении I > A/ (B — C) и при выполнении оператора Y = P/ (L — M).

То же будет и для префикс-ситуации перед оператором DO. Она относится только к выражению, которое стоит в операторе DO, и не распространяется на операторы DO-группы, а также на присваивание значения управляющей переменной. Поэтому префикс-ситуация перед оператором DO первого типа не имеет смысла. Если необходимо изменить состояние ситуации для оператора внутри DO-группы, то префикс-ситуацию необходимо написать перед этим оператором.

Пример.

```
(NOZERODIVIDE): BEGIN;
DECLARE (C, D, E) FLOAT,
        (I, J, K) FIXED;
          ....
(ZERODIVIDE): DO I = 2 TO J BY (J — 2)/ (K — I);
                ....
              C = D/E;
              END;
            ....
          END;
```

Лишь при делении на нуль в выражении (J — 2)/ (K — I) в операторе DO может произойти программное прерывание, так как при вычислении этого выражения включена ситуация ZERODIVIDE. При делении на нуль в выражении с плавающей точкой D/E прерывание не будет обрабатываться. В этом месте программы ситуация ZERODIVIDE согласно префикс-ситуации перед оператором BEGIN находится в выключенном состоянии.

10.3. Оператор ON

Любое программное прерывание регистрируется операционной системой, и управление в этом случае может быть передано на программу, обрабатывающую это прерывание. В языке ПЛ/1 имеется средство — оператор ON, с помощью которого программист может определить действие при возникновении исключительной ситуации. Действие должно быть выполнено, если в результате возникновения этой ситуации произойдет прерывание программы.

Формат оператора имеет вид

ON имя ситуации спецификация—действия;

Спецификация используется программистом, чтобы установить некоторое действие, которое будет выполнено при возникновении прерывания программы. В качестве спецификации могут использоваться: оператор GOTO, пустой оператор и ключевое слово SYSTEM.

Если используется оператор GOTO, то при возникновении программного прерывания, вызванного ситуацией, имя которой стоит в операторе ON, управление передается оператору, помеченному меткой, которая указана в операторе GOTO. Как правило, этот помеченный оператор (или блок) определяет действие программиста на возникшее прерывание программы.

Пустой оператор может быть указан для любой ситуации, кроме KEY, ENDFILE и CONVERSION. Его использование означает, что не надо ничего делать при возникновении прерывания программы, т. е. фактически прерывание игнорируется.

Таким образом, действие, заданное пустым оператором, напоминает выключение ситуации в префиксе, но не эквивалентно ему по трем причинам:

— пустой оператор может использоваться для любых состояний, но не все состояния могут быть выключены с помощью префикс-ситуации;

— выключение ситуации может сэкономить время, которое тратится на проверку этой ситуации. Так как если установлен пустой оператор, то система должна еще проверить наличие ситуации и передать управление пустому оператору, который вернет управление в точку вызова на продолжение выполнения программы;

— области действия префикс-ситуации и оператора ON различны.

Ключевое слово SYSTEM устанавливает стандартное системное действие при возникновении исключительной ситуации, т. е. при возникновении прерывания программы.

Выполнение оператора ON связывает спецификацию действия с именем исключительной ситуации. Эта связь сохраняется до завершения блока, в котором выполняется оператор ON, или до следующего оператора ON для этой же ситуации. Кроме того, действие оператора ON сохраняется и во всех блоках, которые активизируются из данного блока. Однако в этих активных блоках действие оператора может быть отменено другим оператором ON. Новое действие остается в силе только до конца этого блока. Когда управление возвращается к вызывающему блоку, все действия по прерыванию программы, которые существовали в этой точке, снова устанавливаются. Это делает невозможным для подпрограммы изменение реакции на прерывание, установленной для блока, который вызвал подпрограмму.

Если в некотором блоке задано несколько операторов ON для какой-то исключительной ситуации, то влияние более раннего оператора ON отменяется более поздним оператором.

Примеры.

1)
```
PR: PROC;
      ....
      R1: GET EDIT (A, B) (2F (7,3));
      ON CONVERSION GOTO R2;
      ....
      GET EDIT (X) (F (6));
      R2: ....
    END PR;
```

Если во время выполнения первого оператора GET будет прочитан недопустимый символ, то произойдет стандартная реакция системы. Оператор ON с помощью спецификации указывает действие при этой же ошибке в дальнейшем тексте программы. Таким образом, если ошибка преобразования произойдет при передаче X во втором операторе GET, обычная последовательность операторов будет прервана и произойдет передача управления на метку R2.

2)
```
PR11 :    PROC;
      ....
   S1 :     ON OVERFLOW;
   ....
   CALL  M;
   ....
   M : PROC;
....
   S2 : ON OVERFLOW GOTO M1;
   M1: оператор;
   ....
   S3: ON OVERFLOW SYSTEM;
   ....
   END M;
   ....
 END  PR11;
```

В данном примере рассматривается повторное действие операторов ON. Если переполнение (ситуация OVERFLOW) произойдет до выполнения оператора ON с меткой S1, то возникнет прерывание, сопровождающееся стандартной реакцией системы. Если же прерывание произойдет после выполнения оператора ON с меткой S1, то оно будет игнорироваться системой (использован пустой оператор).

После обращения к процедуре M оператор ON с меткой S1 сохранит свое влияние до тех пор, пока не будет выполнен оператор ON с меткой S2. С этого момента действие оператора с меткой S1 временно отменяется и начинает действовать оператор ON с меткой S2.

Если переполнение произойдет на участке программы между операторами с метками S2 и S3, возникнет прерывание программы и выполнится оператор GOTO M1.

Оператор ON с меткой S3 после выполнения полностью аннулирует действие оператора с меткой S2 (действие оператора ON с меткой S1 будет отложено до передачи управления в основную процедуру). После того как управление из процедуры M вернется в основную процедуру, восстановится действие оператора ON с меткой S1.

10.4. Операторы REVERT и SIGNAL

Иногда бывает необходимо отменить действие, установленное в данном блоке для некоторой ситуации, и при этом восстановить то действие, которое было в силе при активации этого блока. Для

этого используется оператор REVERT, который имеет следующий формат:

REVERT имя ситуации;

Этот оператор может воздействовать только на те операторы ON, которые находятся в одном блоке с ним, и восстанавливать реакцию на прерывание программы, установленную в ближайшем охватывающем блоке.

Оператор REVERT выполняет указанные действия при следующих условиях:

— после активизации блока, в котором выполняется оператор REVERT, выполняется оператор ON, указывающий ту же самую ситуацию и внутренний к этому блоку;

— не выполнялся другой оператор REVERT для этой ситуации.

Если хотя бы одно из условий не выполняется, то оператор REVERT рассматривается как пустой оператор.

Пример.

```
   A: PROC;
      ....
  M1: ON ZERODIVIDE GOTO ENDE;
      ....
      CALL B;
      ....
      B: PROC;
      ....
      M2: ON ZERODIVIDE;
          ....
          REVERT ZERODIVIDE;
          ....
      END B;
      ....
      M3: ON ZERODIVIDE SYSTEM;
          ...
ENDE: END A;
```

Так как не было никаких отмен, исключительная ситуация ZERODIVIDE при выполнении любой программы является включенной. Если деление на нуль встретится до выполнения оператора с меткой M1, то произойдет прерывание, обрабатываемое стандартным образом.

Если деление на нуль встретится после выполнения оператора с меткой M1 и до выполнения оператора с меткой M2, то произойдет прерывание и управление будет передано оператору с меткой ENDE.

Если деление на нуль встретится после выполнения оператора с меткой M2 и до выполнения оператора REVERT, прерывание фактически не вызовет никакой реакции (так как спецификацией оператора ON является пустой оператор). После выполнения оператора REVERT действие оператора ON с меткой M1 снова восстанавливается. Таким образом, если деление на нуль встретится после выполнения оператора REVERT и до выполнения оператора с меткой M3, то произойдет прерывание и управление будет предано на оператор с меткой ENDE. После выполнения оператора с меткой M3 деление на нуль вызывает стандартное действие системы.

Оператор SIGNAL позволяет программисту имитировать возникновение исключительной ситуации, имя которой указано в этом

операторе. Он имеет следующий формат:

SIGNAL имя ситуации;

Оператор SIGNAL вызывает выполнение того действия по прерыванию, которое было установлено последним оператором ON для указанной ситуации. Этот оператор может быть использован программистом при отладке программы для проверки выполнения действия, установленного оператором ON для указанной ситуации.

После выполнения оператора SIGNAL порядок выполнения операторов основной программы изменяется и управление передается на оператор ON с указанным именем ситуации. Если в качестве действия программистом задан пустой оператор, то управление обычно возвращается оператору, непосредственно следующему за оператором SIGNAL. Если ситуация, указанная в операторе SIGNAL, выключена, то этот оператор эквивалентен пустому оператору.

Если в момент выполнения оператора SIGNAL для указанной в нем ситуации не предусмотрена реакция на прерывание,то вступает в действие стандартная реакция системы.

Пример.

```
RP: PROC;
    ....
    M1 : ON ENDFILE (REF) GOTO FIN;
    ....
    M2 : SIGNAL ENDFILE (REF);
    ....
    M3 : ON ENDFILE (REF) SYSTEM;
    ....
    M4 : SIGNAL ENDFILE (REF);
    ....
FIN : END RP;
```

Оператор SIGNAL с меткой M2 вызывает прерывание, которое имитирует ситуацию окончания файла. В этом случае будет выполняться действие, указанное в операторе ON с меткой M1. Когда прерывание вызывается оператором SIGNAL с меткой M4, управление передается оператору ON с меткой M3 и вызывается стандартное действие системы.

Глава 11

ОТЛАДКА И ВЫПОЛНЕНИЕ ПРОГРАММ, НАПИСАННЫХ НА ЯЗЫКЕ ПЛ/1

11.1. Компонент ПЛ/1 и его место в ДОС

Как уже отмечалось выше, программа, написанная на языке ПЛ/1 (так называемый исходный модуль), не может быть непосредственно выполнена на машине. В данной главе рассмотрим компоненты дисковой операционной системы (ДОС), необходимые для обработки исходного модуля, а также последующей обработки получающегося после трансляции объектного модуля.

Компонент ПЛ/1 включает в себя:

а) транслятор ПЛ/1;

б) модули библиотеки ПЛ/1, которые состоят из:

— модулей управляющей программы ПЛ/1;

— модулей ввода—вывода;

— модулей, реализующих встроенные функции и операцию возведения в степень;

— модуля пересылки адресов;

в) транзитные фазы управляющей программы ПЛ/1.

Модули подключаются к программе, написанной на языке ПЛ/1, при редактировании. Особое место среди модулей библиотеки занимают модули управляющей программы ПЛ/1. Она служит для управления рабочей программой (абсолютного модуля), т. е. программой, полученной после редактирования объектного модуля.

Выполнение программы, написанной на языке ПЛ/1, состоит из нескольких основных этапов, на которых используются отдельные компоненты ДОС и ПЛ/1:

— этап трансляции, на котором используется транслятор ПЛ/1. В результате выполнения этого этапа создается так называемый объектный модуль, который не является еще рабочей программой и кодируется на промежуточном языке, понятном системной программе РЕДАКТОР;

— этап редактирования, на котором работает обслуживающая системная программа РЕДАКТОР и используются отдельные модули библиотеки языка ПЛ/1, реализующие некоторые стандартные

действия (например, включение в программу встроенных функций языка или операций преобразования данных). Результатом работы этапа является так называемый абсолютный модуль (рабочая программная фаза). На этом же этапе могут включаться в создаваемую фазу некоторые другие объектные модули, хранящиеся в библиотеке объектных модулей (например, отдельные отлаженные логические блоки программы или стандартные процедуры);

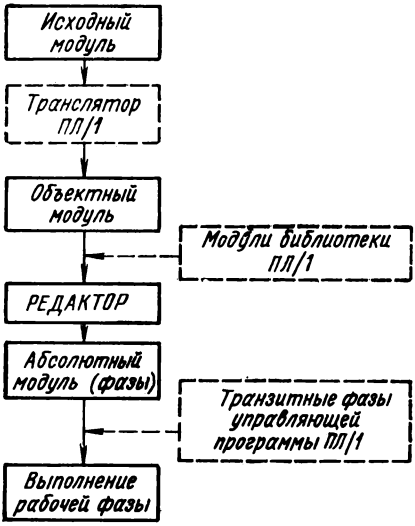


Рис. 11.1. Составные части компонента ПЛ/1.

— этап выполнения, на котором работает системная программа СУПЕРВИЗОР и управляющая программа ПЛ/1, в результате чего создается фактическая связь между рабочей программой и наборами исходных данных, расположенными на внешних носителях.

На рис. 11.1 показано функционирование составных частей компонента ПЛ/1 при трансляции, редактировании и выполнении программы.

При обработке программы, написанной на языке ПЛ/1, используются следующие основные компоненты дисковой операционной системы:

— программа ПЕРВОНАЧАЛЬНАЯ ЗАГРУЗКА;
— программа УПРАВЛЕНИЯ ЗАДАНИЯМИ;
— программа СУПЕРВИЗОР;
— программа РЕДАКТОР.

Прежде чем может быть выполнена трансляция исходного модуля и обработка задания, необходимо привести вычислительную машину и ДОС, которая будет функционировать на ней, в состояние готовности к работе. Это делается с помощью программы ПЕРВОНАЧАЛЬНАЯ ЗАГРУЗКА, которая загружается из резидентного пакета в основную память машины. Эта программа очищает основ-

ную память машины, считывает в нее ядро СУПЕРВИЗОРА и корректирует таблицы физических устройств в ядре СУПЕРВИЗОРА. Работа программы ПЕРВОНАЧАЛЬНАЯ ЗАГРУЗКА завершается обработкой директивы SET, в которой оператор машины указывает системе дату и текущее время дня. Обработав эту директиву, программа ПЕРВОНАЧАЛЬНАЯ ЗАГРУЗКА передает управление СУПЕРВИЗОРУ, который вызывает программу УПРАВЛЕНИЯ ЗАДАНИЯМИ и передает ей управление. Программа УПРАВЛЕНИЯ ЗАДАНИЯМИ управляет потоком заданий, которые должны выполняться на машине.

11.2. Принцип работы транслятора ПЛ/1

Транслятор ПЛ/1 является многофазовым (состоит из 102 фаз) и многопроходным. Он может работать как в фоновом разделе (BG) основной памяти, так и в разделах переднего плана (F1 или F2). При генерации системы может быть создано два варианта транслятора: первый, требующий как минимум 10K основной памяти, и второй, требующий 12K. Первый вариант требует назначения системного логического устройства SYSIPT — на устройство ввода с перфокарт, SYSLST — на устройство печати и SYSPCH — на вы-

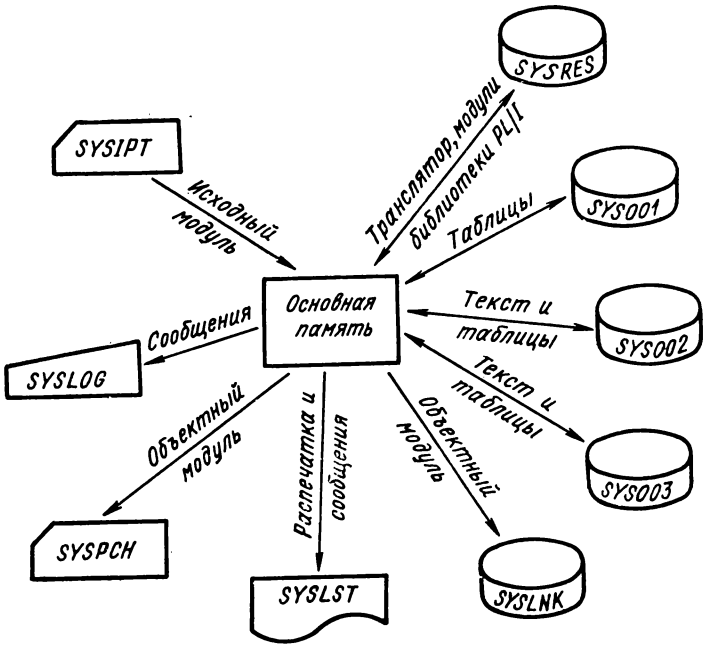


Рис. 11.2. Использование логических устройств при трансляции.

Таблица 11.1

Логическое устройство	Тип устройства	Шифр устройства
SYSIPT	Устройство ввода перфокарточное	ЕС-6012 ЕС-6016
	Накопитель на магнитных лентах	ЕС-5010 ЕС-5012 ЕС-5016 ЕС-5017 ЕС-5019 ЕС-5021
	Накопитель на магнитных дисках	ЕС-5052 ЕС-5055 ЕС-5056 ЕС-5061
SYSRDR	Устройство ввода перфокарточное Накопитель на магнитных лентах Накопитель на магнитных дисках	Те же Те же Те же
SYSRES	Устройство вывода перфокарточное Накопитель на магнитных лентах	ЕС-7010 ЕС-7012 Те же
SYSPCH	Накопитель на магнитных дисках	Те же
SYSLST	Устройство вывода печатающее Накопитель на магнитных лентах Накопитель на магнитных дисках	ЕС-7030 ЕС-7031 ЕС-7032 ЕС-7033 ЕС-7035 Те же Те же
SYSLOG	Пишущая машинка	ЕС-7070 ЕС-7071 ЕС-7072 ЕС-7073 ЕС-7073А ЕС-7074
Y YSLN K	Накопитель на магнитных дисках	Те же
SYS000—SYS222	Накопитель на магнитных дисках Накопитель на магнитных лентах	Те же Те же

ходной перфоратор. Во втором варианте всем этим системным логическим устройствам могут быть назначены магнитные диски.

Память, необходимая для трансляции, распределяется управляющей фазой транслятора и состоит из области загрузки фаз, области ввода—вывода и области таблиц.

Исходная программа (исходный модуль) вводится транслятором с системного логического устройства SYSIPT. Информация для программиста выводится на SYSLST в виде распечаток исходной программы, объектного модуля, таблиц и т. д. Объектный модуль может быть получен на SYSLNK и (или) SYSPCH. Кроме того, для работы транслятору необходимы логические устройства SYS001, SYS002 и SYS003, которые должны быть назначены на диски. Использование логических устройств при трансляции показано на рис. 11.2.

При трансляции исходного модуля управляющая фаза транслятора ПЛ/1 (которая находится в основной памяти весь период обработки исходного модуля) загружает остальные фазы одна за другой. Каждая фаза однократно обрабатывает текст исходного модуля и удаляется из основной памяти. Фазы, предназначенные для обработки тех конструкций языка, которых нет в транслируемой программе, в основную память не вызываются.

Трансляция исходного модуля состоит из следующих этапов:

— организация трансляции;

— синтаксический анализ и обработка объявлений;

— раскрытие программной структуры и создание внутренних макрокоманд;

— создание машинного кода и выдача сообщений об ошибках;

— распределение памяти для рабочей программы, создание и выдача объектного модуля.

Объектный модуль, полученный в результате трансляции, состоит из программного и файлового модулей.

Необходимым условием работы транслятора является назначение логическим устройствам возможных физических устройств машины. Назначение производится при помощи оператора управления заданиями ASSGN. Для выполнения рабочей программы допустимы следующие назначения физических устройств (табл. 11.1).

11.3. Задание на выполнение программы

Каждая работа, заключающаяся в выполнении одной или нескольких программ, последовательное выполнение которых служит единой цели, оформляется в виде задания. Задание определяет, какие программы должны быть выполнены и какие для этого необходимы ресурсы системы (область основной памяти, внешние устройства). Часть задания, связанная с выполнением одной программы, называется шагом задания. Задание может состоять из одного или нескольких шагов.

Управление выполнением задания ведется с помощью операторов УПРАВЛЕНИЯ ЗАДАНИЯМИ, которые считываются с внешнего устройства, соответствующего логическому устройству SYSRDR (ему назначается, как правило, физическое устройство ввода с перфокарт). Каждое задание должно начинаться с оператора:

// _JOB имя задания [комментарии],

где «имя задания» — определенное программистом название задания, содержащее от 1 до 8 символов.

После имени задания могут быть записаны комментарии. При организации работ на вычислительном центре здесь целесообразно указывать фамилию программиста или какую-то специальную информацию.

Оператором конца задания является оператор:

/ & [комментарии]

За знаком /& должны следовать как минимум 11 пробелов, за которыми в этом операторе могут следовать комментарии, аналогичные комментариям, указанным в операторе начала задания.

Начало шага задания (загрузка определенной фазы) и начало его выполнения обеспечиваются следующим оператором:

// _ EXEC [имя фазы],

где «имя фазы» — имя программы, которую нужно вызвать из библиотеки абсолютных модулей и выполнить.

Эта фаза может быть единственной, вызываемой из библиотеки, или только первой фазой, которая сама вызывает другие фазы программы. Если имя фазы не указано, то будет выполняться фаза, созданная РЕДАКТОРОМ при выполнении предыдущего шага задания и временно помещенная в библиотеку абсолютных модулей.

Если для выполнения программы требуются некоторые данные, (размещенные на перфокартах), которые должны читаться с SYSIPT, то они помещаются непосредственно за картой с оператором EXEC. Шаг задания заканчивается оператором EXEC или оператором /*. Этот оператор должен быть последним оператором в любом файле, вводимом с логических устройств SYSRDR и SYSIPT.

Пример. Программистом написана программа на языке ПЛ/1. Она состоит из одной внешней процедуры (исходный модуль). Программа отперфорирована, и теперь программист хочет проверить, не содержит ли его программа синтаксических ошибок и ошибок перфорации. Для этого программа должна быть только протранслирована.

В этом случае задание должно быть оформлено так:

```
// JOB TRANS                                  Иванов отдел 325
// EXEC PL/I
   PR : PROCEDURE OPTIONS (MAIN);
   . ....
   END PS;
/ *
/ &
```

В данном задании из библиотеки абсолютных модулей сначала вызывается фаза с именем *PL/I* (это первая управляющая фаза транслятора), которая затем вызывает следующую фазу и т. д. Данными для работы транслятора является исходный модуль.

В рассматриваемом задании выполняется только один шаг — трансляция исходного модуля.

Вообще говоря, в задании могут присутствовать такие шаги:

— трансляция исходного модуля;

— редактирование объектного модуля (создание программной фазы);

— загрузка полученной фазы в основную память и выполнение этой фазы;

— каталогизация исходного модуля в библиотеку исходных модулей;

— каталогизация объектного модуля в библиотеку объектных модулей;

— редактирование и каталогизация в библиотеку абсолютных модулей;

— выполнение обслуживающих и вспомогательных программ;

— выполнение библиотечных функций;

— выполнение программ пользователя.

Управляющему оператору EXEC могут предшествовать другие операторы программы УПРАВЛЕНИЯ ЗАДАНИЯМИ, которые служат для подготовки системы к выполнению программы, выполняющейся в этом шаге задания.

11.4. Трансляция исходной программы и использование операторов программы УПРАВЛЕНИЯ ЗАДАНИЯМИ

Системная программа УПРАВЛЕНИЯ ЗАДАНИЯМИ загружается в тот раздел основной памяти, где должно выполняться подготовленное задание или шаг задания. Она обеспечивает подготовку системы к выполнению очередного задания или шага задания.

Обычно выполнение программы УПРАВЛЕНИЯ ЗАДАНИЯМИ заканчивается передачей управления СУПЕРВИЗОРУ, который вызывает указанную в задании программу. Эта программа, как правило, загружается в то место основной памяти, где выполнялась программа УПРАВЛЕНИЯ ЗАДАНИЯМИ. По окончании задания или шага задания СУПЕРВИЗОР вновь загружает программу УПРАВЛЕНИЯ ЗАДАНИЯМИ в основную память.

Программа УПРАВЛЕНИЯ ЗАДАНИЯМИ при наличии соответствующих операторов выполняет следующие функции:

— установка режима работы системы и запись информации в область связи СУПЕРВИЗОРА (операторы OPTION, UPSI);

— назначение логическим устройствам конкретных физических устройств (операторы ASSGN и RESET);

— загрузка фазы в основную память (оператор EXEC);

— обработка и запоминание информации о метках наборов данных (операторы DLBL, EXTENT, TLBL, LBLTYP);

— подготовка к запуску и запуск программ с контрольной точки (оператор RESTART);

— некоторые специальные функции (операторы MTC, PAUSE).

Операторы УПРАВЛЕНИЯ ЗАДАНИЯМИ, подготовленные на перфокартах, вставляются в задание. При этом надо иметь в виду, что операторы УПРАВЛЕНИЯ ЗАДАНИЯМИ обычно вводятся с логического устройства SYSRDR, а данные (перфокарты, которые лежат между операторами EXEC и /*), так же как исходный модуль, — с SYSIPT. Как правило, для логических устройств SYSRDR и SYSIPT назначается одно и то же физическое устройство (устройство ввода с перфокарт), поэтому все задание вводится с этого устройства. Операторы УПРАВЛЕНИЯ ЗАДАНИЯМИ могут вводиться также с пишущей машинки. В некоторых случаях с пишущей машинки можно исправлять ошибки в операторах. Основные операторы УПРАВЛЕНИЯ ЗАДАНИЯМИ, используемые при трансляции исходных модулей на языке ПЛ/1 и выполнения программ, приведены в приложении 5. Рассмотрим некоторые из них.

Оператор OPTION позволяет в любом задании установить режим работы системы. Данный оператор имеет следующий формат:

// OPTION режим [,режим] ...

С помощью режимов этого оператора можно получить достаточно большой объем информации о транслируемой программе, что позволяет эффективно и направленно вести отладку программы. Перечислим важнейшие режимы, которые можно указать в этом операторе:

LIST — при указании этого режима транслятор распечатывает исходный модуль. Как правило, является стандартным режимом, устанавливаемым при генерации ДОС;

SYM — при указании этого режима транслятор выводит на печать таблицу всех встречающихся в программе идентификаторов с принадлежащими им атрибутами. Кроме того, при установке этого режима на печать выдается таблица смещений, таблица внешних символических имен и таблица блоков. Целесообразно устанавливать этот режим при отладке программы с целью проверки правильности присвоения желаемых атрибутов идентификаторам, используемым в программе;

ERRS — режим, при котором транслятор выводит на печать список всех ошибок, обнаруженных в исходном модуле. Этот список содержит для каждой ошибки номер оператора, в котором она обнаружена. Указывается степень грубости ошибки и дается текстовое объяснение ее. Как правило, этот режим является стандартным;

DECK — режим, при котором транслятор осуществляет вывод объектного модуля на выходной перфоратор (в виде колоды перфокарт). Этот режим целесообразно устанавливать при окончании отладки программы для получения готового объектного модуля как документа;

LISTX — режим, при котором транслятор выводит на печать объектный модуль в ассемблированной форме, если исходный модуль не содержит ошибок;

LINK — режим, при котором транслятор выводит на SYSLNK объектный модуль для обработки его редактором. РЕДАКТОРУ этот режим указывает, что абсолютный модуль, полученный после редактирования, должен быть временно записан в библиотеку абсолютных модулей. Режим устанавливается обязательно, если в задании предусматривается выполнение программы;

XREF — режим, при котором транслятор выводит на печать таблицу перекрестных ссылок. Эта информация может помочь программисту при анализе логики программы;

DUMP — режим, при котором в случае отмены задания (из-за ошибок) выводится на печать содержимое областей основной памяти, используемых управляющей программой и разделом, в котором выполнялась проблемная программа, а также содержимое общих регистров;

LOG — режим, при котором осуществляется вывод операторов УПРАВЛЕНИЯ ЗАДАНИЯМИ на печать. Как правило, является стандартным режимом

Каждый из перечисленных режимов может отрицаться предлогом NO. Например, режим NODUMP говорит о том, что не требуется вывод содержимого основной памяти на печать в случае ненормального завершения задания Но есть такие режимы, которые не могут употребляться с предлогом NO. К ним относятся:

CATAL — указание этого режима вызывает каталогизацию фазы или программы в библиотеку абсолютных модулей по завершению работы РЕДАКТОРА. Указание режима CATAL автоматически вызывает установку режима LINK;

48C — информирует транслятор о том, что исходная программа написана с применением 48-символьного алфавита;

60C — этот режим сообщает транслятору о том, что исходный модуль написан с применением 60-символьного алфавита.

Во время генерации системы устанавливается набор стандартных режимов для того, чтобы программисту не нужно было указывать все режимы в каждом задании. Однако при необходимости программист может аннулировать любой стандартный режим, указав новый режим в операторе OPTION. Этот новый режим будет действовать весь период выполнения задания (если в задании не используется оператор PROCESS).

Теперь, рассмотрев основные режимы оператора OPTION, составим задание, которое должно состоять из нескольких шагов.

Пример. Программа, состоящая из одной внешней процедуры, должна быть протранслирована, отредактирована и выполнена. При выполнении программы должны быть обработаны перфокарты с исходными данными. Карты читаются с SYSIPT, причем для SYSIPT и SYSRDR назначено одно и то же устройство. При выполнении задания нужно выдать на печать текст программы, таблицу символов и объектный модуль в ассемблированной форме.

Если при выполнении программы обнаружится ошибка, то ее выполнение должно быть прекращено без распечатки содержимого основной памяти.

Решение.

Для решения нашей задачи необходимы режимы SYM, LISTX и NODUMP.

В первом шаге задания исходный модуль должен быть протранслирован. Для этого вызывается транслятор (имя его первой фазы PL/I). Результатом трансляции является объектный модуль. Для дальнейшей обработки он помещается на SYSLNK, для чего используется режим LINK.

Во втором шаге задания РЕДАКТОР (имя первой фазы которого LNKEDT) преобразует объективный модуль в выполняемую программную фазу. Устанавливается адрес загрузки и автоматически присоединяются другие фазы программы и программы библиотеки, которые необходимы для работы.

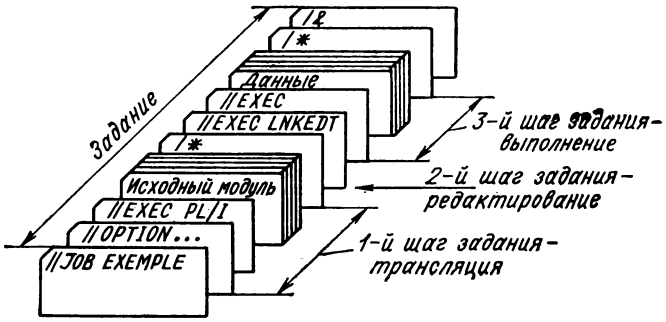


Рис. 11.3. Структура задания на выполнение программы.

В третьем шаге задания выполняется фаза, полученная в результате работы РЕДАКТОРА. В этом случае в операторе EXEC имя фазы не задается.

Итак, для выполнения поставленной задачи требуются следующие управляющие операторы:

```
//—JOB EXEMPLE   ТРАНСЛЯЦИЯ И ВЫПОЛНЕНИЕ
// — OPTION LIST,LINK,SYM,LISTX,NODUMP
//—EXEC PL/I
  |Перфокарты исходного модуля|
/*
//—EXEC LNKEDT
//—EXEC
  |Исходные данные|
/*
/&
```

На рис. 11.3 показана структура задания на выполнение данной программы, подготовленного на перфокартах, в случае, когда логическим устройствам SYSRDR и SYSIPT назначено одно и то же физическое устройство (устройство ввода перфокарточное).

Оператор PROCESS используется для организации трансляции нескольких внешних процедур в одном шаге задания.

Общий формат оператора:

*_PROCESS режим [, режим]...

Символ * должен располагаться в первой колонке перфокарты. За этим символом следует хотя бы один пробел.

Режимы, записанные в этом операторе, не должны выходить за 71-ю колонку перфокарты. Если все требуемые режимы не помещаются на одной перфокарте, то может быть использовано несколько операторов PROCESS.

В этом операторе может быть задано несколько режимов, которые используются в операторе OPTION. К ним относятся: ERRS, LIST, SYM, XRFF, DECK, LISTX, 48C и 60C.

Новыми режимами, которые указываются только в операторе PROCESS, являются: OPT (NOOPT), STMT (NOSTMT), LISTO (NOLISTO), RCATP.

Режим OPT указывает на необходимость оптимизации объектного модуля Оптимизация заключается в удалении ряда команд, которые транслятор считает избыточными в объектной программе. В результате такой оптимизации сокращается объем объектного модуля. По умолчанию подразумевается режим OPT.

Режим STMT указывает на необходимость печатать номера операторов, в которых обнаружена ошибка, во время выполнения рабочей программы. По умолчанию принимается режим NOSTMT. Этот режим эквивалентен использованию оператора //_ UPSI_01 версии 1.3 ДОС ЕС.

Режим LISTO реализует выдачу таблицы смещений, в которой печатаются номера операторов и их смещение в байтах по отношению к началу программного модуля. Этот режим отменяет действие режима LISTX, т. е. если заданы оба режима, то LISTX игнорируется.

Режим RCATP позволяет программисту задать имена для создаваемых транслятором программного и файлового объектных модулей. Общий формат режима:

RCATP [(имя программного модуля [, имя файлового модуля])]

При написании большой программы для эффективной отладки ее можно разбить на отдельные смысловые части и оформить их в виде внешних процедур По мере окончания отладки каждую готовую процедуру целесообразно помещать в библиотеку объектных модулей (БОМ) и хранить там, задавая определенные имена для программного и, если необходимо, файлового модулей. Предварительно перед каталогизацией в БОМ каждый объектный модуль должен быть выведен на перфокарты (устройство SYSPCH) со специальной управляющей картой (управляющим оператором):

CATALR имя модуля.

При использовании режима RCATP и создаются объектные модули с включением в них управляющих карт для каталогизации в БОМ с помощью оператора MAINT программы БИБЛИОТЕКАРЬ.

Возможны следующие варианты:

1) режим задан в форме RCATP.

В этом случае выходная последовательность перфокарт имеет вид

```
[CATALR F—имя          ]
[<файловый модуль>     ]
      CATALR имя
[INCLUDE F—имя]
<программный модуль>
```

где «имя» является именем основной точки входа процедуры, которая описана в программе;

«F—имя» — имя файлового модуля, которое получено из имени транслируемой процедуры присоединением буквы F к ее началу. Файловый объектный модуль создается при объявлении в транслируемой процедуре некоторого обрабатываемого файла;

2) режим задан в форме RCATP (имя программного модуля).

Выходная последовательность перфокарт имеет вид

```
[CATALR имя F...F      ]
[<файловый модуль>     ]
      CATALR имя
[INCLUDE имя F...F]
<программный модуль>
```

где «имя» — это имя транслируемой процедуры;

«имя F...F» — имя файлового модуля, которое получено дополнением к имени процедуры справа буквы F до восьми символов;

3) режим задан в полной форме RCATP (имя программного модуля, имя файлового модуля).

Выходная последовательность перфокарт имеет вид

```
CATALR имя файлового модуля
<файловый модуль>
CATALR имя программного модуля
 INCLUDE имя файлового модуля
<программный модуль>
```

Пример. Необходимо получить в виде колоды перфокарт объектный модуль процедуры FUNK для каталогизации в БОМ с именем PROGR1:

```
//—JOB EXEMPL
//—OPTION DECK
//—EXEC PL/I
 *—PROCESS SYM, RCATP (PROGR1)
  FUNK: PROC OPTIONS (MAIN);
          DCL TAR FILE...;
          ....
  END FUNK;
/*
/&
```

В результате трансляции будут созданы файловый и программный объектные модули и на выходном перфораторе выдана колода перфокарт следующего вида:

```
CATALR PROGR1FF
< файловый модуль >
CATALR PROGR1
INCLUDE PROGR1FF
< программный модуль >
```

С помощью оператора PROCESS можно организовать трансляцию нескольких внешних процедур в одном шаге задания, т. е. за один вызов транслятора. Это позволит сократить время трансляции всего задания и дает возможность изменять значения режимов при трансляции каждой процедуры.

Для такой пакетной трансляции оператор // EXEC PL/I предшествует только первой транслируемой процедуре. Каждой из последующих процедур должен предшествовать хотя бы один оператор PROCESS.

Пример.

```
//—JOB EXEMPL
//—OPTION SYM,LINK
//—EXEC PL/I
*—PROCESS LISTO,STMT
   PR1: PROC OPTIONS (MAIN);
      .....
   END PR1;
 *—PROCESS NOSYM,STMT
   PR2 : PROC;
      ....
   END PR2;
*—PROCESS
   PR3 : PROC;
      ....
   END PR3;
/*
/&
```

11.5. Использование операторов программы УПРАВЛЕНИЯ ЗАДАНИЯМИ для описания наборов данных

При выполнении некоторых программ необходимо связать логический файл, описанный в этой программе, с конкретным набором данных, расположенным на внешнем носителе. Как известно, эта связь осуществляется при открытии файла с помощью оператора OPEN. При этом возникают следующие проблемы:

— установление соответствия между логическими устройствами, которые задаются в режиме MEDIUM атрибута ENVIRONMENT объявления файла, и физическими устройствами, на которых размещается набор данных:

— передача информации для создания и проверки меток наборов данных на магнитных лентах и дисках;

— описание области на магнитном диске, в которой будут располагаться записи набора данных;

— передача информации РЕДАКТОРУ для определения области меток.

При объявлении файла в атрибуте ENVIRONMENT программист с помощью режима MEDIUM связывает имя файла с некоторым логическим устройством. Однако эта связь достаточно общая, т. е. она определяет только тип логического устройства, не указывая конкретного адреса этого устройства (а их на машине может быть несколько). Непосредственная связь объявленного в программе файла с конкретным физическим устройством, на котором расположен набор данных, осуществляется с помощью оператора ASSGN, имеющего следующий формат:

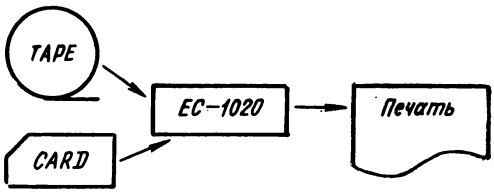
```
//_ ASSGN_SYSxxx,адрес,
```

где SYSxxx — символическое имя логического устройства;

«адрес» — адрес физического устройства в шестнадцатиричном коде X'Cuu';

C — номер канала; uu — номер устройства.

Пример. В программе используются три набора данных, которые расположены на устройствах, показанных на рисунке:



Эти файлы объявляются следующими операторами DECLARE:

```
DECLARE TAPE FILE INPUT RECORD
        ENV (F (120) MEDIUM (SYS009, 5010) NOLABEL));
DECLARE CARD FILE INPUT RECORD
        ENV (F (80) MEDIUM (SYSIPT, 6012));
DECLARE LIST FILE PRINT ENV (F (120) MEDIUM
        (SYS010, 7030);
```

Какими должны быть операторы ASSGN, при помощи которых логическим устройствам, указанным в режимах MEDIUM, назначаются конкретные физические устройства?

Предположим, что файл TAPE находится на магнитной ленте с адресом '282'; файл CARD — на перфокарточном устройстве ввода с адресом '00G', а файл LIST должен быть выведен на устройство печати с адресом '00F'.

Тогда нам потребуются следующие операторы:

```
//_ASSGN_SYS 009,X'282'
//_ASSGN_SYS010,X'00F'
```

Оператор

```
//_ ASSGN_SYSIPT,X'00C'
```

не требуется, если для SYSIPT назначено системное устройство перфокарточного ввода с адресом X'00C'.

В объявлении файлов лучше употреблять системные логические устройства SYSIPT, SYSPCH, SYSLST, назначение физических устройств которым делается программой ПЕРВОНАЧАЛЬНАЯ ЗАГРУЗКА. Следовательно, файл LIST лучше объявить так:

```
DECLARE LIST FILE PRINT ENV (F (120) MEDIUM
(SYSLST, 7030));
```

Тогда оператор ASSGN для назначения логическому устройству SYS010 физического устройства с адресом '00F' не потребуется, так как оно является стандартным.

Указав с помощью оператора ASSGN конкретное устройство, на котором расположен набор данных, программисту необходимо сообщить операционной системе характеристики этого набора. К таким характеристикам относятся:

— метка набора данных, в которой содержатся сведения о его учетной информации;

— характеристика конкретных участков магнитного диска, на которых расположены записи набора данных.

Создание меток набора данных осуществляется с помощью операторов TLBL и DLBL.

Оператор TLBL определяет информацию о метке набора данных на магнитной ленте.

Он имеет следующий формат:

```
// TLBL  имя файла,
         ['идентификатор набора данных'],
         [дата],
         [регистрационный номер набора данных],
         [порядковый номер тома],
         [порядковый номер набора данных],
         [номер поколения],
         [номер версии поколения]
```

Рассмотрим структуру и назначение основных параметров:

«имя файла» может содержать от 1 до 7 символов и должно быть тождественным имени файла, используемому в операторе DECLARE, который описывает этот файл;

«идентификатор набора данных» может содержать не более 17 символов, заключенных в кавычки. Идентификатор присваивается набору данных программистом и является его именем. Если этот параметр опущен для выводного набора данных, в качестве идентификатора набора берется имя файла. Если он опущен для вводного набора данных, то проверка идентификатора не производится;

«дата» определяет срок хранения набора данных и может задаваться в одной из двух форм: либо в виде числа, содержащего от одного до четырех цифр и указывающего количество дней хранения набора, либо в виде абсолютной даты истечения срока хранения, со-

держащей последние две цифры года и номер дня в году (формат ГГ/ДДД). В каждой метке набора данных содержится дата создания и дата истечения срока его хранения. Дата создания сообщается программе ПЕРВОНАЧАЛЬНАЯ ЗАГРУЗКА при создании набора. Дата истечения срока переносится из оператора TLBL. Для вводных наборов этот параметр значения не имеет. Для выводных наборов сравнивается дата истечения срока годности и текущая дата. Если срок хранения истек, то нужно установить другую катушку магнитной ленты;

«регистрационный номер набора данных» может содержать от 1 до 6 алфавитно-цифровых символов, идентифицирующих первую (или единственную) катушку набора на магнитной ленте. Если параметр содержит менее шести символов, он дополняется нулями слева. Если параметр опущен, будет использованы регистрационный номер тома первой (или единственной) катушки набора. Если параметр пропущен для вводного набора данных, контроль регистрационного номера не производится.

Описание остальных параметров рассматривается в курсе операционной системы ДОС/ЕС.

При размещении набора данных на магнитном диске создание метки задается с помощью оператора DLBL.

Оператор DLBL имеет формат

```
//_ DLBL  имя файла,
          ['идентификатор набора данных'],
          [дата], [коды],
```

где «имя файла» может содержать от 1 до 7 символов и должно быть тождественным имени файла, используемому в операторе DECLARE, который описывает этот файл;

«идентификатор набора данных» может содержать не более 44 символов, заключенных в кавычки. Идентификатор присваивается набору данных программистом, является его именем и может включать номер поколения и номер версии поколения. Если этот параметр опущен, в качестве идентификатора набора берется имя файла;

«дата» имеет то же назначение, что и в операторе TLBL;

«коды» — двух- или трехсимвольное поле, идентифицирующее тип организации набора данных:

SD — последовательная организация;

DA — региональная организация;

ISG — индексно-последовательная организация (используется для создания набора данных);

ISE — индексно-последовательная организация (используется для обработки набора данных).

Если параметр «коды» опущен, то принимается код SD.

Оператор DLBL дает общую информацию о наборе данных. Конкретная характеристика размещения записей набора данных на участках магнитного диска осуществляется с помощью оператора

тается ошибочным. Для последовательных и региональных наборов данных с атрибутом INPUT этот параметр может быть опущен. Для подсчета адреса участка используется формула

адрес участка = 10 х номер цилиндра + номер дорожки на цилиндре.

Например, для участка, начинающегося на седьмой дорожке сотого цилиндра, адрес участка равен

$$10 \times 100 + 7 = 1007;$$

«количество дорожек» — это количество дорожек, занимаемых участком (от одной до пяти десятичных цифр). Для последовательного набора данных этот параметр можно опустить. Для набора данных, разделяющих цилиндры с другими наборами, количество дорожек представляет произведение количества цилиндров, отведенных набору, на количество дорожек, занимаемых им на каждом из цилиндров;

«дорожка разделения цилиндра» — номер последней из дорожек, занимаемых участком на каждом из цилиндров в случае разделения цилиндров для последовательных наборов. Номер первой дорожки этого участка берется из адреса участка.

При обработке набора данных в проблемной программе метка, расположенная на внешнем носителе и характеризующая набор данных, для сокращения времени ее обработки должна быть переписана в основную память. Выделение для этого некоторого объема памяти в области проблемной программы осуществляет РЕДАКТОР, а информация РЕДАКТОРУ задается программистом с помощью оператора LBLTYP. Оператор LBLTYP должен использоваться только в том случае, если программа обрабатывает ленточные наборы данных с метками или наборы данных на дисках, отличные от последовательных (так как для последовательных наборов данных место для метки выделяется автоматически в области памяти СУПЕРВИЗОРА).

Оператор LBLTYP может иметь одну из двух форм:

— если программа обрабатывает наборы данных на магнитной ленте с метками и не работает ни с индексно-последовательными файлами, ни с региональными файлами, то

```
// _ LBLTYP TAPE
```

— если программа обрабатывает индексно-последовательные наборы данных или наборы с прямым доступом, то независимо от того, какие еще типы файлов обрабатываются в программе,

```
// _ LBLTYP LSD (nn)
```

В этой форме *nn* указывает максимальное количество участков, используемых одним набором данных.

EXTENT. Так как практически набор данных может размещаться не на одном, а на нескольких отдельных участках (под участком понимается одна или несколько подряд идущих дорожек, занятых записями набора данных), то за каждым оператором DLBL может следователь несколько операторов EXTENT, характеризующих эти участки. За каждым оператором DLBL должен обязательно следовать хотя бы один оператор EXTENT.

Оператор EXTENT имеет следующий формат:

```
// — EXTENT [SYS xxx], [регистрационный номер],
                [тип], [номер участка],
                [адрес участка], [количество дорожек],
[дорожка разделения цилиндра],
```

где SYSxxx — символическое имя логического устройства, которому назначен накопитель на магнитных дисках. Этот параметр должен совпадать с логическим устройством, заданным в режиме MEDIUM атрибута ENVIRONMENT. Если этот параметр опущен, то берется имя, указанное в предшествующем операторе EXTENT.

«Регистрационный номер» — это регистрационный номер пакета дисков (тома), на котором находится описываемый в EXTENT участок (состоит из 1—6 символов). Если этот параметр опущен, то используется номер, указанный в предшествующем операторе EXTENT. Если этот параметр опущен во всех операторах EXTENT, то во время открытия набора данных не производится контроль на правильность установки нужного диска (тома) и программист сам несет ответственность за использование неверного носителя;

«тип» — тип участка; этот параметр записывается в виде одной из цифр 1, 2, 4, 8, где 1 — определяет участок, предназначенный для размещения данных (неразделенные цилиндры); 2 — участок для области переполнения (для индексно-последовательного набора данных); 4 — участок для размещения индексов (для индексно-последовательного набора данных); 8 — участок для размещения данных (разделенные цилиндры).

Если параметр опущен, то принимается тип 1.

«Номер участка» — десятичное число от 0 до 255, определяющее порядковый номер участка для набора данных, расположенного на нескольких участках. Номер участка должен быть указан для каждого индексно-последовательного и регионального набора данных. Первый или единственный участок регионального или последовательного набора имеет номер 0. Для участка, используемого для размещения главного индекса индексно-последовательного набора, должен быть указан номер 0. Если индексно-последовательный набор данных не имеет главного индекса, то его первый участок должен иметь номер 1.

«Адрес участка» занимает от 1 до 5 цифр, определяет относительный номер первой дорожки участка (относительно нулевой дорожки на нулевом цилиндре). Если адрес участка опущен для индексно-последовательного набора данных, то такой оператор счи-

В потоке оператор LBLTYP должен ставиться перед вызовом управляющей фазы РЕДАКТОРА, т. е. перед картой

```
// — EXEC LNKEDT
```

Рассмотрим на примерах применение описанных выше операторов.

Примеры.

1) В программе используются два набора данных, описанные следующим образом:

```
DECLARE TAPE1 FILE RECORD OUTPUT
        ENV (F (400, 50) MEDIUM (SYS010, 5010);
DECLARE CARD FILE RECORD INPUT
        ENV (F (80) MEDIUM (SYS009, 6012));
```

Необходимо составить операторы УПРАВЛЕНИЯ ЗАДАНИЯМИ для трансляции программы и ее выполнения. Перфокарты должны читаться с устройства с адресом '00C'; магнитная лента должна располагаться на устройстве с адресом '281'. Порядковый номер файла на ленте 6;

Идентификатор набора данных 'СКЛАД — 1'. Набор данных должен существовать до пятидесятого дня 1977 года.

Решение.

```
//—JOB EXEM1
//—ASSGN SYS009,X'00C'
//—ASSGN SYS010,X'281'
//—OPTION LINK,SYM
//—EXEC PL/I
|Исходный модуль|
/*
//— LBLTYP TAPE
//—EXEC LNKEDT
//—TLBL TAPE1,'СКЛАД — 1',77/050,,,000006
//—EXEC
|Исходные данные|
/*
/&
```

2) В программе при использовании входного ленточного набора данных создается выходной индексно-последовательный набор данных. Файлы, обрабатывающие эти наборы данных, описаны в операторе DECLARE следующим образом:

```
DECLARE GRAD FILE RECORD INPUT
        ENV (F (400, 50) MEDIUM (SYS010, 5010));
DECLARE PLATE FILE RECORD OUTPUT KEYED
        SEQUENTIAL ENV (F (50) MEDIUM (SYS013, 5056)
        INDEXED KEYLENGTH (10) EXTENTNUMBER (2)
        OFLTRACKS (2));
```

Магнитная лента установлена на устройстве с адресом '280'. Регистрационный номер набора данных на ленте 12, идентификатор набора 'КОНСТАНТЫ'. Время хранения набора данных до 100-го дня 1977 года.

Магнитный диск располагается на устройстве с адресом '191'. Идентификатор индексно-последовательного набора данных 'КОНСТАНТЫ ДАННЫХ', регистрационный номер тома 777777.

Набор данных располагается на 12 цилиндрах, начиная со 180-го цилиндра. Для индексов цилиндра выделяется четыре дорожки на 179-м цилиндре. Дата истечения срока хранения — 150-й день 1977 года.

Решение.

```
//—JOB EXEMPLE
//—ASSGN SYS013,X'191',
//—ASSGN SYS010,X'280'
//—OPTION LINK
//—EXEC PL/I
|Исходный модуль|
/*
//—LBLTYP NSD (02)
//—EXEC LNKEDT
//—TLBL GRAD, 'КОНСТАНТЫ',77/100,000012
//—DLBL PLATE, 'КОНСТАНТЫ ДАННЫХ',77/150,ISC
//—EXTENT SYS013,777777,4,1,1790,4
//—EXTENT SYS013,777777,1,2,1800,120
//—EXEC
/&
```

В данном примере создается индексно-последовательный набор данных, поэтому «код» в операторе DLBL указан ISC. Ввиду того, что набор данных размещается на нескольких цилиндрах, необходимо создать индекс цилиндра, который с помощью первого оператора EXTENT размещается на 179-м цилиндре и занимает 4 дорожки. «Тип» участка обозначается цифрой 4, а номер участка — 1 (так как не создается главный индекс).

Второй участок (его номер 2), который представляет собой область памяти, выделяемой для размещения данных и области переполнения располагается со 180-го цилиндра и занимает 120 дорожек.

3) Для набора данных с организацией REGIONAL (3) необходимо подготовить участок диска, а затем после трансляции и редактирования программы, в которой этот набор используется, выполнить ее.

Набор данных занимает 18 дорожек, начиная с 58-го цилиндра. Магнитный диск располагается на устройстве с адресом '191' и имеет регистрационный номер 000666.

Имя файла REF, идентификатор набора данных 'СВЕДЕНИЯ О ВЫПОЛНЕНИИ ПЛАНА', символическое имя устройства SYS015. Длина записи 80 байтов, длина ключа 10 байтов. Дата истечения срока хранения файла 50-й день 1978 года.

Для того чтобы область диска подготовить для приема записей данных при использовании прямого доступа (файлы REGIONAL (1) и REGIONAL (3)), используется вспомогательная программа CLRDSK. Управляющий оператор программы, в котором задаются различные параметры, имеет формат

$$//—\text{UCL B} = (\text{K} = mmm,\ \text{D} = nnn),\ \begin{Bmatrix}\text{C'c'}\\ \text{X'xx'}\end{Bmatrix},\ \begin{Bmatrix}\text{OY}\\ \text{ON}\end{Bmatrix},$$

где *mmm* — длина ключа в байтах (для файлов REG (1), для которых ключи на носителе не создаются, длина ключа должна равняться 0);

nnn — длина области данных в байтах.

С помощью C'c' или X'xx' задаются символы, которыми заполняется область данных (псевдозаписи). При этом в первом случае символ заполнения—любой символ языка, а во втором—символ в шестнадцатиричном виде.

При указании параметров OY или ON производится создание набора данных с контролем операции записи или без контроля.

Последним управляющим оператором для CLRDSK является оператор конца задания //END. Необходимо заметить, что программа чистки использует символическое имя файла UOUT. Кроме того, необходимо иметь в виду, что задание кода в операторе DLBL является лишним, так как форматизация происходит последовательно, а задание кода SD является значением по умолчанию. Во время выполнения программы CLRDSK на печать выводятся сообщения, информирующие программиста о режиме и результатах ее работы.

Решение.

```
//_JOB EXEM3
//_ASSGN SYS015,X'191'
//_DLBL UOUT,'СВЕДЕНИЯ О ВЫПОЛНЕНИИ ПЛАНА',78/050
//_EXTENT SYS015,000666,1,0,580,18
//_EXEC CLRDSK
//_UCL B = (K = 10, D = 80), C'A', OY
//_END
//_EXEC PL/I
```

Исходный модуль

```
/*
//_LBLTYP NSD (01)
//_EXEC LNKEDT
//_DLBL REF, 'СВЕДЕНИЯ О ВЫПОЛНЕНИИ ПЛАНА',78/050,DA
//_EXTENT SYS015,000666,1,0,580,18
//_EXEC
```

Перфокарты с данными

```
/*
/&
```

11.6. Информация, выдаваемая на печать во время выполнения задания

Во время выполнения задания на печать (устройство SYSLST) выдается достаточно обширная и подробная информация о программе в виде сообщений и таблиц, которая может значительно помочь программисту при отладке. Общий объем информации, выдаваемой на печать, показан на рис. 11.4. Всю информацию, выдаваемую на печать во время выполнения задания, можно разделить на три части:

— печать операторов УПРАВЛЕНИЯ ЗАДАНИЯМИ и некоторой другой информации об обработке управляющих операторов программой УПРАВЛЕНИЯ ЗАДАНИЯМИ;

— печать сообщений транслятора, которые относятся к программе, написанной на языке ПЛ/1;

— печать о редактировании программы РЕДАКТОРОМ.

Перед началом выполнения каждого задания производится распечатка операторов УПРАВЛЕНИЯ ЗАДАНИЯМИ, используемых для трансляции исходного модуля. В первой строке вместе с информацией управляющего оператора JOB об имени задания печатается информация о времени и дате выполняемого задания в виде

```
// JOB EXEMPL ВАСИЛЬЕВА ОТД. 325 21.36.20 DATE 20/11/76
// OPTION LINK,SYM
// EXEC PL/I
```

Если заданы определенные режимы, транслятор выводит на печать информацию о транслируемой программе, которая включает

— список режимов;
— распечатку исходного и объектного модулей программы;
— таблицы символических и внешних имен;
— сообщения об ошибках (диагностические сообщения);
— таблицу смещений;
— таблицу блоков;
— сообщения об окончании трансляции.

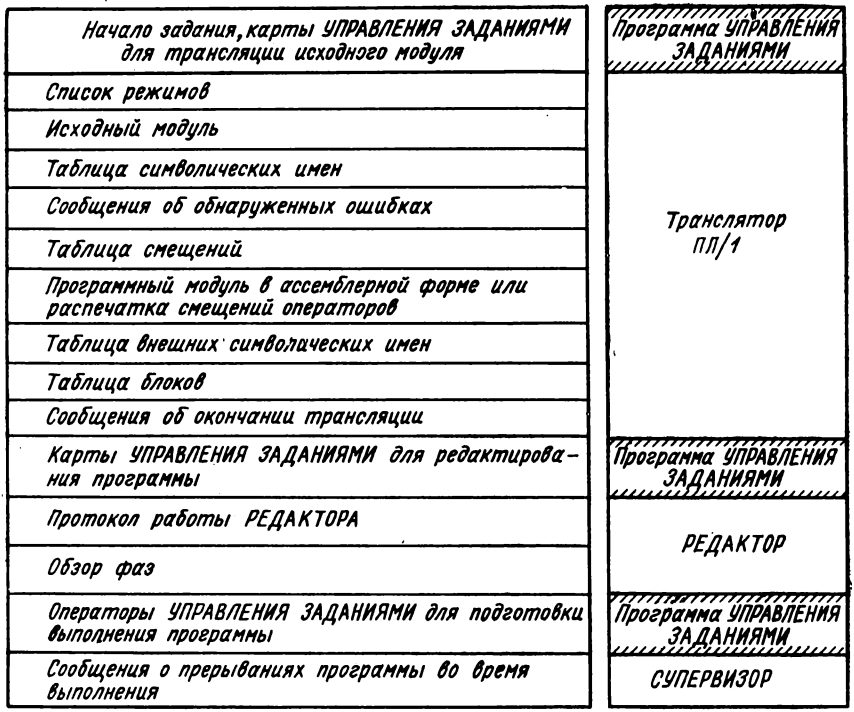


Рис. 11.4. Обзор информации, выводимой на SYSLST

Каждый лист распечатки, выдаваемой транслятором, начинается с заголовка листа, который содержит сведения об используемом трансляторе, имени задания, дате и номере листа распечатки.

Список режимов. После обработки транслятором всех операторов PROCESS (если они использованы программистом) на печать выдается список режимов, действующих при трансляции данной программы. Сначала распечатываются все операторы PROCESS с указанными в них режимами, а затем список режимов, заданных в операторе OPTION, и стандартных режимов транслятора. Список режимов предворяется заголовком OPTIONS LIST.

Пример.

```
DOS/EC PL/I V.M.2.0. <имя задания> 9/11/76 PAGE 002
                    OPTIONS LIST
* PROCESS STMT, LISTX
* PROCESS XREF, NOOPT, LISTO
OPTIONS TAKEN ARE 60G, LIST, SYM, ERRS, LINK, LOG
```

Исходный модуль. Если задан режим LIST (или он является стандартным), то все перфокарты исходной программы распечатываются на SYSLST, при этом содержимое каждой перфокарты печатается в отдельной строке.

При трансляции производится нумерация исходной программы. Номера операторов печатаются в начале той строки, где находится оператор. Если в одной строке содержится несколько операторов, то печатается номер первого.

Заметим, что если комментарии, константы типа строки символов или спецификации шаблона атрибута PICTURE в исходном тексте определены неправильно, то нумерация исходных операторов может быть ошибочной, а иногда и неполной. Причинами этого могут быть:

— признак конца комментария неправильно отперфорирован;
— признак конца комментария расположен в колонках 73—80;
— кавычка, обозначающая конец константы типа строки символов или спецификации шаблона, отсутствует или неправильно отперфорирована.

Первая перфокарта исходного модуля печатается на каждой странице. Поэтому целесообразно, чтобы она содержала текст, характеризующий программу. Например,

```
/* ОПРЕДЕЛЕНИЕ ДИНАМИЧЕСКИХ НАГРУЗОК */
 PR: PROCEDURE OPTIONS (MAIN);
 . . . . . .
```

В этом случае на каждой странице выдаваемого на печать текста будут помещены комментарии, указанные на первой перфокарте исходного модуля.

При распечатке текста исходной программы для повышения ее читабельности и наглядности имеется возможность регулировать интервалы между строками на печати. Для этого используется первая колонка перфокарты (которая не должна использоваться для записи текста программы). В этой колонке могут записываться следующие символы:

_	— продвижение бумаги на 1 строку;
0	— продвижение бумаги на 2 строки;
—	— продвижение бумаги на 3 строки;
1	— переход на новую страницу.

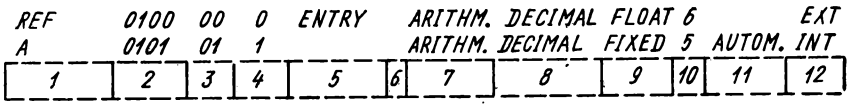
Любые другие значения символов на первой колонке перфокарты воспринимаются, как ошибочные. Такая перфокарта трансля-

тором не обрабатывается, а на печать для нее выдается 14 звездочек и содержимое колонок 2—80 перфокарты.

Для приведенного примера (если при набивке исходного модуля имя точки входа набито с 1-й колонки) в распечатке будет выдано:

```
/* ОПРЕДЕЛЕНИЕ ДИНАМИЧЕСКИХ НАГРУЗОК *)
**·····* R: PROCEDURE OPTION (MAIN);
```

Таблица символических имен. Если указан режим SYM, то все идентификаторы, используемые в программе, печатаются в таблице символических имен (SIMBOL TABLE LISTNG). После заглавия таблицы символических имен печатается информация о назначении и атрибутах используемых идентификаторов от протранслированной программы. Эту информацию можно условно разделить на 12 граф, каждая из которых имеет следующее значение:



1 — идентификатор, используемый программистом в исходной программе;

2 — внутреннее имя (четыре шестнадцатиричных цифры);

3 — номер блока, в котором объявляется идентификатор (две десятичных цифры);

4 — номер уровня блока, т. е. глубина вложенности блока (одна десятичная цифра);

5 — графа пустая или содержит

ARRAY — если идентификатор является именем массива;

STRUC — если идентификатор является именем старшей структуры, подструктуры или элемента структуры;

ENTRY — если идентификатор является именем точки входа;

BUILTIN — если идентификатор является встроенной функцией;

6 — графа пустая или содержит номер уровня структуры (одна десятичная цифра);

7 — графа пустая или содержит:

ARITHM — для арифметических переменных;

PICTURE — для цифровых знаковых данных;

LABEL — для меток;

POINTER — для переменной типа указателя;

STRING — для строки символов;
FILE — для имени файла;

8 — графа пустая или содержит:
ALIGNED (UNAL) — атрибут расположения в памяти элементов данных;
BINARY (DECIMAL) — для переменных с двоичным или десятичным основанием;
CONST. — для констант типа метки;
VARIAB. — для переменных типа метки;

9 — графа пустая или содержит:
BIT (CHAR) — атрибут строки битов или строки символов;
FIXED (FLOAT) — атрибут способа представления для арифметических данных;

10 — атрибут разрядности или пробел;

11 — графа пустая или содержит:
AUTOM (STATIC) — атрибут класса памяти для переменной;
BASED — для базированной переменной;
PARAM. — для параметров;
DEFIN. — для идентификаторов, имеющих атрибут DEFINED;

12 — EXT (INT) — атрибут области действия.

Если в процессе трансляции были обнаружены ошибки в объявлении имен, то для них в таблице символических имен печатаются сообщения об ошибках. В этом случае таблица символических имен выводится даже тогда, когда задан режим NOSYM. В табл. 11.2 приведен перечень сообщений об ошибках.

Таблица перекрестных ссылок. Если задан режим XREF, то на печать выдается таблица перекрестных ссылок (CROSS REFERENCE LISTING), в которой в алфавитном порядке распечатываются внешние и внутренние имена переменных. Для каждого имени выдается на печать следующая информация:

— внутренний код имени (четыре шестнадцатиричные цифры). Они совпадают с внутренними именами, указанными в графе 2 таблицы символических имен;

— номер уровня блока, в котором встречается имя переменной (совпадает с графой 3 таблицы символических имен);

— список номеров операторов программы (кроме оператора DECLARE), в которых встречается данное имя.

Пример.

```
CROSS  REFERENCE LISTING
A      0103 02 11 12 15 22
ATR    0101 01  2 16 22
```

Сообщения об ошибках во время трансляции. Ошибки, вызванные несоблюдением синтаксических правил языка или нарушениями ограничений данной реализации, обнаруживаются транслятором.

Таблица 11.2

Код ошибки	Текст сообщения об ошибке
01	SYNTACTICAL DECLARE ERROR Синтаксическая ошибка при объявлении переменной
02	CONFLICTING ATTRIBUTES Противоречивые атрибуты
03	PRECISION IS MISSING OR WRONG Отсутствует или неправилен атрибут разрядности
04	BASE VARIABLE ITSELF IS DEFINED OR BASED Идентификатор базы в атрибуте DEFINED сам является базированным или имеет атрибут DEFINED
05	BASE OR POINTER INCORRECT Базированная переменная или указатель объявлен неправильно
06	ATTRIBUTES OF SECONDARY ENTRY CONFLICT WITH THOSE OF PRIMARY ENTRY Атрибуты дополнительной точки входа противоречат атрибутам основной точки входа
07	MULTI — DECLARED IDENTIFIER Многократно объявленный идентификатор
08	ENTRY RETURNS VALUE WITH CONFLICTING ATTRIBUTES Атрибуты возвращаемого значения в вызывающей процедуре противоречат атрибутам объявления имени входа в процедуре-функции
09	INVALID STRUCTURE Неправильное строение структуры
OA	ARRAY TOO LONG Массив слишком длинный
OB	STRUCTURE TOO LONG Структура слишком длинная
OG	POINTER IN BASED STRUCTURE Структура с атрибутом BASED содержит переменную типа указателя
OD	TOO MANY ARRAYS Слишком много массивов в транслируемой внешней процедуре (максимально может быть 32)
OE	INVALID PICTURE Ошибка в атрибуте PICTURE
OF	STRUCTURE LEVEL DEEPER THAN EIGHT Уровень вложенности структуры больше 8
10	NAME EXCEEDS 31 CHARACTERS IN LENGTH Длина идентификатора более 31 символа
11	EXTERNAL NAME EXCEES 8 CHARACTER IN LENGTH Длина внешнего имени больше 8 символов
12	MULTIPLE DECLARATION IF EXTERNAL NAME INCONSISTENT Несколько объявлений внешнего имени противоречивы

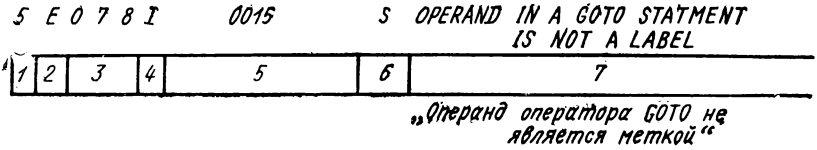
В этом случае для каждой обнаруженной ошибки печатается диагностическое сообщение, которое содержит степень грубости ошибки (классификация их приведена в табл. 11.3), номер оператора, десятичный код и текст ошибки.

Таблица 11.3

Классификация ошибок по степеням грубости

Степень грубости	Пояснение	Замечание
W (предупреждение)	Транслятор предполагает ошибку	Программист должен проверить исходный текст на синтаксическую правильность
E (ошибка)	Программа неправильна. Транслятор производит корректировку программы	Программист должен проверить, устраивает ли его исправление программы, сделанное транслятором
S (серьезная ошибка)	Программа содержит ошибки, которые транслятор не может исправить. Трансляция продолжается, обработка объектного модуля РЕДАКТОРОМ не производится	При ошибке степени грубости S может производиться распечатка программного модуля в ассемблерной форме, таблицы смещений и таблицы внешних символических имен. Программа не выполняется
T (ошибка вызывает прекращение трансляции)	Прекращение трансляции и окончание шага задания	

Сообщения об ошибках (диагностические сообщения) выдаются, начиная с нового листа распечатки. Каждое диагностическое сообщение можно условно разбить на несколько частей (граф), в которых содержится информация о значении и номере сообщения, номере ошибочного оператора, степени грубости ошибки и текста сообщения:



Каждая из граф сообщения означает:

1 — цифра 5, которая указывает, что дает сообщение об ошибке транслятор с языка ПЛ/1;

2 — вид ошибки;

A — ошибка в назначении устройств, в режимах и в карте PROCESS;

C — ошибка в операторе DECLARE;

E — указывает нарушение синтаксических правил языка ПЛ/1;

C — указывает на несоблюдение существующих ограничений;

3 — десятичный номер сообщения, состоящий из трех цифр;

4 — символ I, который обозначает, что сообщение об ошибке является информационным сообщением для программиста;

5 — номер ошибочного оператора в исходном тексте, состоящий из четырех десятичных цифр;

6 — степень грубости ошибки;

7 — текст сообщения.

Все ошибки, обнаруживаемые транслятором при обработке оператора PROCESS, имеют степень грубости W, т. е. транслятор продолжает свою работу. Неверный оператор PROCESS или неверно заданный режим этого оператора игнорируется. Сообщения об ошибках в операторе PROCESS печатаются в списке режимов (OPTIONS LIST) после распечатки операторов PROCESS.

Печать диагностических сообщений вида C и E происходит в следующем порядке. Сначала после заголовка (DECLARE STATEMENT DIAGNOSTICS) печатаются сообщения об ошибках в операторе DECLARE. Затем после заголовка (DIAGNOSTIC MESSAGES) печатаются синтаксические ошибки, обнаруженные транслятором в программе.

Пример.

```
DOS/ES PL/I V.M.2.0    <имя задания>    <дата>    PAGE 005
DECLARE STATEMENT DIAGNOSTICS
5CO761 0003 W RECORDSIZE OF RECORD NOT DIVISIBLE BY 8 IN
                              FILE BAND
DOS/ES PL/I V.M.2.0  <имя задания>  <дата> PAGE 006
DIAGNOSTIC MESSAGES
5E0971 0005 E (MISSING) INSERTED IN FORMAT LIST
```

В конце списка ошибок печатается результирующее сообщение, текст которого зависит от характера имеющихся ошибок (табл. 11.4). Результирующее сообщение выводится также на пишущую машинку оператора.

Таблица смещений. Если задан режим SYM, то выдается на печать таблица смещений (OFFSET TABLE), в которой вместо идентификаторов, употребляемых программистом, указываются их внутренние имена. Таблицу смещений условно можно разбить на четыре графы.

В каждой графе содержатся следующие значения:

INTERNAL NAME — внутреннее имя проблемного данного или данного управления программой;

OFFSET — смещение элемента данных относительно начала SSA или DSA соответствующего блока;

Таблица 11.4

Степень грубости ошибки	Результирующее сообщение
W или E	5E031 POSSIBLE ERRORS IN SOURCE PROGRAM Трансляция закончена. Возможны ошибки в объектном модуле
S	5E021 LINK OPTION RESET Трансляция закончена. Редактирование отменено, т. е. объектный модуль не передан на SYSLNK
T	5E011 JOBSTER PL/I TERMINATED, LINK OPTION RESET Шаг задания на PL/I снимается. Объектный модуль не строится

TYPE указывает принадлежность данных управления программой и проблемных данных к классу памяти STATIC или AUTOMATIC Этим определяется, в какой области (статической или динамической) отведена память для элемента данных;

MODULE OFFSET указывает смещение элемента данных относительно начала модуля. Для данных с классом памяти AUTOMATIC это смещение не указывается.

Для всех проблемных данных и данных управления программой (переменные типа метки, указателя и точки входа), которые объявлены в программе и занимают область памяти в области динамической памяти (DSA) или в области статической памяти (SSA), указываются смещения (в шестнадцатиричной системе счисления) относительно начала DSA или SSA.

Пример.

```
INTERNAL NAME  OFFSET TYPE      MODULE OFFSET OFFSET TABLE
     0102        0038  STATIC                 000308
     0103        0114  AUTOMATIC
```

Для определения абсолютных адресов элементов данных сначала используется таблица символических имен, в которой указываются внутренние имена переменных, объявленных в исходном модуле. Используя эти внутренние имена , в таблице смещений можно отыскать адреса элементов данных относительно начала DSA или SSA.

Для определения абсолютного адреса данного, который имеет класс памяти STATIC, необходимо знать следующее:

— смещение проблемного данного относительно начала SSA, которое определяется из таблицы смещений;

— начальный адрес SSA относительно программного модуля главной процедуры. Начальный адрес SSA можно определить из распечатки программного модуля в ассемблерной форме;

— адрес загрузки программного модуля главной процедуры, который определяется из обзора фаз, выдаваемого РЕДАКТОРОМ.

Таким образом, абсолютный адрес некоторого проблемного данного A, который обозначим через ADR (A), вычисляется следующим образом:

ARD (A) = относительный адрес SSA в программном модуле + + смещение A относительно начала SSA + адрес загрузки программного модуля.

Определение адреса данного управления программой происходит иначе. Адрес данного управления программой (обычно адрес имени точки входа), которое имеет класс памяти STATIC, определяется с помощью указателя адресов. Указатель адреса — это адрес области статической памяти, в которой содержится адрес данного управления программой. Для определения указателя адреса необходимо знать следующее:

— адрес загрузки программного модуля главной процедуры, который определяется из обзора фаз, выдаваемого РЕДАКТОРОМ;

— адрес области статической памяти, в которой находится адрес точки входа, выдаваемый в таблице смещений (MODULE OFFSET).

Таким образом, указатель адреса данного управления программой вычисляется так:

ADR (данного) = адрес загрузки программного модуля + + адрес области статической памяти, в которой содержится «данное».

Для определения абсолютного адреса проблемного данного, которое не является параметром и имеет класс памяти AUTOMATIC, должно приниматься во внимание следующее:

— к какому блоку принадлежит данное, что выбирается из распечатки таблицы символических имен (графа 3);

— длина DSA данного блока, которая выбирается из таблицы блоков (LENGTH OF DSA);

— последовательность активизации блоков.

Пример

1) Рассмотрим вычисление абсолютного адреса проблемного данного VAR, которое относится к блоку уровня 03. Главная процедура HP вызывает процедуру UP1, которая, в свою очередь, вызывает процедуру UP2 (структура DSA для данных процедур показана на рис. 11.5.). Начальный адрес DSA блока 1-го уровня можно получить следующим образом:

— из обзора фаз, выдаваемого РЕДАКТОРОМ, получаем адрес последнего байта фазы (графа HICORE);

— адрес начала области для DSA блока первого уровня определяем с границы следующего двойного слова.

Абсолютный адрес проблемного данного VAR, объявленного в процедуре UP2, вычисляется следующим образом:

ADR (VAR) = начальный адрес DSA (01) + + длина DSA (01) + длина DSA (02) + + адрес VAR относительно начала DSA (03).

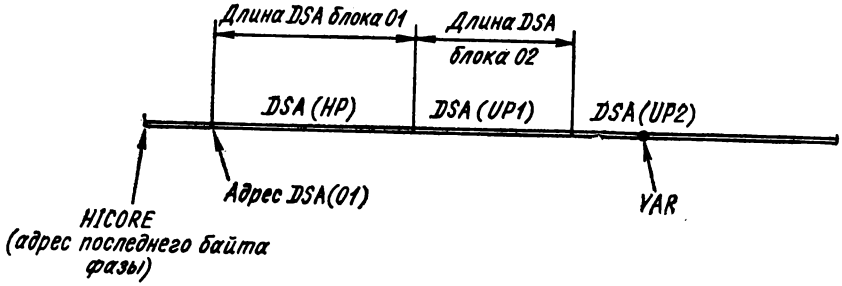


Рис. 11.5. Структура DSA для процедур HP, UP1 и UP2.

2) Рассмотрим вычисление абсолютного адреса проблемного данного, которое относится к блоку уровня 02. Пусть HP — главная процедура, в которой вызываются процедуры UP1 и UP2 (внутренние или внешние). Структура DSA для процедур HP, UP1 и UP2 в этом случае показана на рис. 11.6.

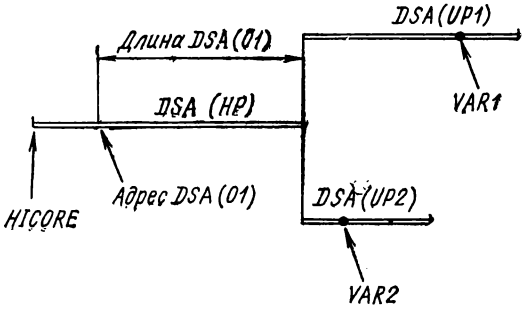


Рис. 11.6. Структура DSA для процедур HP, UP1, UP2.

Абсолютные адреса проблемных данных VAR1 и VAR2, объявленных в процедурах UP1 и UP2, вычисляются следующим образом:

ADR (VAR1) = начальный адрес DSA (01) + длина DSA (01) + + адрес VAR1 относительно DSA (02);
ADR (VAR2) = начальный адрес DSA (01) + длина DSA (01) + + адрес VAR2 относительно DSA (02).

Печать программного модуля в ассемблерной форме. Если задан режим LISTX и синтаксические ошибки степени грубости T отсутствуют, транслятор выдает распечатку программного модуля в ассемблерной форме. Распечатка программного модуля состоит из нескольких частей (граф). После стандартного заголовка листа печатается строка, содержащая название каждой графы:

LOG — относительный адрес, который указывается в шестнадцатиричной системе счисления;

OBJECT CODE — объектный код оператора;

LABEL содержит символическую метку или остается пустой;

OP — мнемонический код операции;

OPERANDS — операнды.

В последней графе печатается номер оператора исходного текста программы. Он устанавливает соответствие между исходным оператором программы и полученным в результате трансляции этого оператора объектным кодом. Кроме того, в этой графе могут печататься следующие комментарии:

BEGIN OF BLOCK *nn* указывает начало блока с номером *nn*;

LENGTH OF DSA OF BLOCK *nn* — размер динамической памяти (DSA) блока *nn* указан в графе OBJECT CODE и соответствует размеру, указанному в таблице блоков;

END OF BLOCK указывает конец блока;

STATIC STORAGE указывает начало статической памяти (SSA), символическая метка которой L'FFFF'.

Пример.

```
DOS/ES PL/I V. M. 2. 0 <имя задания>  <дата> PAGE 008
LOC. OBJECT  CODE LABEL  OP. OPERANDS
000000   05F0                BALR F,0
000002             L'010E' BEGIN OF BLOCK 01
```

Таблица внешних символических имен. Если задан режим SYM, то транслятор выдает на SYSLST таблицу внешних символических имен (EXTERNAL SYMBOL TABLE). Эта таблица содержит имена программных секций объектного модуля (файлового и программного), а также внешние ссылки и имена дополнительных точек входа, обнаруженных внутри модуля. Каждая графа таблицы внешних символических имен имеет следующее значение:

SYMBOL — имя программного или файлового модуля, для которых создается программная секция (SD), или имя модуля, которое используется как внешняя ссылка (ER), или имя дополнительной точки входа (LD), с помощью которой можно обратиться к данному модулю;

TYPE — обозначение SD, ER или LD;

ESID — нумерация внешних символических имен внутри каждой из двух частей объектного модуля (для типов SD и ER);

ADDR — указывается для типов SD и LD, причем для SD — X'000000', а для LD — это смещение относительно начала программной секции;

LENGTH — длина файлового или программного модуля (указывается для типа SD);

ESID — номер программной секции, к которой относится дополнительная точка входа (для типа LD).

Пример.

```
                                EXTERNAL SYMBOL TABLE
SYMBOL     TYPE  ESID  ADDR    LENGTH  ESID
TAPE       ED    0001  000000  00017A
IYFFZZZZ   ER    0002
CARD       SD    0001  000000  000144
```

```
IJJFCIZD    ER    0002
IJKTXCF     ER    0003
IJKTXCW     ER    0004
```

Из таблицы можно получить, в частности, следующую информацию:

— для файлов TAPE и CARD, объявленных в программе, имеются программные секции длиной 17A и 144 байтов соответственно;

— внешняя ссылка (ER) IYFFZZZZ из программной секции TAPE указывает на модуль для работы с магнитной лентой (MTMOD). Используя имя IYFFZZZ, можно определить некоторые параметры MTMOD;

— в программной секции CARD генерируются внешние имена IJJFCIZD, IJKTXCF и IJKTXCW, которые относятся к модулю логической CYBB и модулям библиотеки ПЛ/1:

а) IJJFCIZD — для обращения к модулю при работе с файлом CARD;

б) IYKTXCF — имя подпрограммы обработки конца файла:

в) IYKTXCW — имя подпрограммы обработки ошибок, возникающих при передаче данных.

Распечатка смещений операторов. Если задан режим LISTO, то для каждого оператора исходной программы печатается его номер (STNU) и относительный адрес (LOC) начала следующего оператора в объектном модуле. Режим LISTO аннулирует режим LISTX, т. е. если заданы оба режима, то режим LISTX игнорируется. Смещения операторов печатаются после таблицы смещений.

Пример.

```
DOC/ES PL/I V.M.2.0  ‹имя задания›  ‹дата›  PAGE 00)
STNU LOC STNU LOC STNU LOC
STNU 1 00003C STNU 6 000048 STNU 7 000066
STNU 9 00007E STNU 10 000094 STNU 11 0000D2
```

Таблица блоков. Если задан режим SYM, транслятор выдает на печать таблицу блоков (BLOCK TABLE). Для каждого программного блока исходного модуля в таблице печатается номер блока — графа BLOCK и размер соответствующей области динамической памяти (DSA) — LENGTH OF DSA.

Пример.

```
BLOCK LENGTN OF DSA BLOCKTABLE
01      0IBO
02      0080
```

Сообщения об окончании трансляции. Конец трансляции указывается одним из двух сообщений. Одно сообщение

5W01I SUCCESSFUL COMPILATION

указывает, что транслятор не обнаружил ошибок в исходной программе и трансляция завершена успешно.

Если при трансляции были обнаружены ошибки степени грубости не выше S, то трансляция завершается следующим сообщением.

5W02I COMPILATION IN ERROR

Оно означает, что трансляция закончена, но в объектном модуле есть ошибки. Сообщения о завершении трансляции печатаются с нового листа распечатки после стандартного заголовка. Это сообщение является последним сообщением транслятора ПЛ/1.

11.7. Функции РЕДАКТОРА и редактирование объектного модуля

Объектный модуль, полученный в результате работы транслятора, должен быть обработан РЕДАКТОРОМ, который создает рабочую программу — так называемый абсолютный модуль. Вызов управляющей фазы РЕДАКТОРА производится с помощью оператора УПРАВЛЕНИЯ ЗАДАНИЯМИ:

```
// EXEC LNKEDT
```

Оператору EXEC для вызова РЕДАКТОРА могут предшествовать управляющие операторы РЕДАКТОРА: PHASE, INCLUDE ENTRY и ACTION. Работа РЕДАКТОРА возможна, если в операторе УПРАВЛЕНИЯ ЗАДАНИЯМИ OPTION задан режим LINK или CATAL. Эти режимы отменяются в случае грубой ошибки в исходной программе.

В простейшем случае задается только режим LINK, а операторы РЕДАКТОРА не указываются. РЕДАКТОР ВЫПОЛНЯЕТ при этом следующие действия:

— используя объектный модуль, помещаемый транслятором на SYSLNK, создает рабочую фазу и временно помещает ее с именем PHASE*** в библиотеку абсолютных модулей. Во время редактирования разрешаются все внешние ссылки и автоматически включаются все необходимые модули библиотеки ПЛ/1 и логический CYBB;

— определяет адрес загрузки созданной фазы, который зависит от конечного адреса программы СУПЕРВИЗОР и длины области меток;

— определяет пусковой адрес фазы, который задается с помощью основной точки входа главной процедуры, содержащейся в фазе;

— выдает на печать (устройство SYSLST) информацию для программиста о ходе и результатах редактирования. Эта информация включает протокол РЕДАКТОРА, диагностические сообщения и обзор фаз. Временно помещенная в библиотеку абсолютных модулей фаза может быть загружена и выполнена только сразу же после редактирования с помощью оператора EXEC без операнда.

Рассмотрим более подробно порядок задания и функции операторов РЕДАКТОРА, которые вводятся, как и операторы УПРАВЛЕНИЯ ЗАДАНИЯМИ, с устройства SYSRDR.

Если рабочая программа, полученная после работы РЕДАКТОРА, будет выполняться несколько раз, то существует возможность отредактированную программу поместить (каталогизировать) в библиотеку абсолютных модулей и вызывать ее оттуда для выполнения. Кроме того, в библиотеке абсолютных модулей каталогизируется программа, состоящая из нескольких отдельных фаз.

Если фазы должны быть каталогизированы или программа состоит из нескольких фаз, то в задании нужно указать один или не-

сколько операторов PHASE. Каждый такой оператор вызывает построение фазы. Оператор PHASE должен быть помещен перед первым объектным модулем соответствующей фазы. Последующие модули, вплоть до следующего оператора PHASE, будут принадлежать данной фазе. Несколько операторов PHASE не могут следовать непосредственно друг за другом.

Оператор PHASE, который служит для указания имени и адреса загрузки фазы, имеет следующий формат:

␣[␣...] PHASE имя фазы, адрес загрузки,

где «имя фазы» — символическое имя, с которым фаза будет каталогизирована в библиотеку абсолютных модулей. Оно должно содержать от пяти до восьми буквенно-цифровых символов, первым из которых не должен быть символ *. Первые четыре символа имен фаз многофазовой программы должны быть одинаковы;

«адрес загрузки» —это адрес основной памяти, начиная с которого отредактированная фаза должна загружаться для выполнения.

«Адрес загрузки» может задаваться одним из следующих выражений:

S — указывает, что если программа будет выполняться в разделе BG, то адресом загрузки будет адрес первого двойного слова после конца области управляющей программы. В адрес загрузки добавляется длина области меток, если она есть. Если программа будет выполняться в разделах переднего плана F1 или F2, то адресом загрузки будет адрес соответствующего раздела, увеличенный на длину области сохранения;

* — указывает, что в качестве адреса загрузки фазы РЕДАКТОР должен использоваться адрес первого двойного слова после конца предыдущей фазы. Если символ * используется в первом для данной программы операторе PHASE, то он рассматривается РЕДАКТОРОМ как элемент S;

ROOT указывает, что формируемая фаза является корневой фазой, т. е. фазой, которая будет находиться в основной памяти в течение всего времени выполнения отредактированной программы. Адрес загрузки корневой фазы соответствует заданию элемента S. ROOT разрешается использовать только для первой фазы оверлейной программы;

«имя» — указывает, что фаза с именем «имя—фазы» должна загружаться с адреса, равного адресу загрузки фазы с именем «имя», которое должно быть ранее определено, т. е. обе фазы загружаются на одно и то же место.

Оператор ACTION предназначен для задания режимов работы РЕДАКТОРА. Этот оператор должен предшествовать всем другим операторам РЕДАКТОРА, но не является обязательным оператором.

С помощью одного оператора ACTION можно задать только один режим, для задания нескольких режимов потребуется подготовить соответствующее количество операторов. Если в операторе ACTION

обнаружена ошибка, то игнорируются все последующие операторы ACTION в данном шаге задания.

Оператор ACTION, используемый в задании на редактирование объектного модуля ПЛ/1, имеет следующий формат:

$$\text{ACTION}\left\{\begin{matrix}\text{CLEAR}\\ \text{MAP}\\ \text{NOMAP}\\ \text{CANCEL}\\ \text{F1}\\ \text{F2}\end{matrix}\right\},$$

где CLEAR указывает РЕДАКТОРУ, что перед выполнением редактирования свободная область библиотеки абсолютных модулей должна быть заполнена нулями. Однако необходимо помнить, что этот режим значительно увеличивает время редактирования;

MAP указывает, что во время редактирования на SYSLST должна быть выдана информация обзора фаз и диагностические сообщения РЕДАКТОРА;

NOMAP указывает РЕДАКТОРУ, что устройство SYSLST недоступно для вывода информации. В этом случае обзор фаз не печатается вообще, а диагностические сообщения РЕДАКТОРА печатаются на SYSLOG. По умолчанию предполагается режим MAP;

CANCEL указывает, что задание должно быть прекращено после печати обзора фаз, если при редактировании были обнаружены ошибки;

F1 или F2 указывает РЕДАКТОРУ, что программа должна быть отредактирована для выполнения в разделах переднего плана F1 или F2 соответственно. Оператор ACTION с операндами F1 и F2 может использоваться только в системе с мультипрограммированием, в противном случае этот оператор игнорируется.

Оператор INCLUDE используется для того, чтобы включить в фазу модуль или программную секцию. Этот оператор имеет следующий формат:

INCLUDE [имя модуля] [,(список имен программных секций)],

где «имя модуля» — заданное имя модуля, которое использовалось при каталогизации этого модуля в библиотеку объектных модулей. Это имя содержит от одного до восьми символов;

«список имен программных секций» — список имен программных секций, которые принадлежат указанному в операнде «имя модуля» модулю. Количество имен программных секций, заданных в одном операторе INCLUDE, не должно превышать пяти.

Чтобы включить в фазу более пяти секций из одного модуля, нужно использовать несколько операторов INCLUDE со вторым операндом. Если указанная секция не найдена, сообщение об этом не выдается.

Входная информация РЕДАКТОРА заканчивается оператором ENTRY. Если этот оператор написан программистом, то он дол-

жен следовать после всей информации, необходимой РЕДАКТОРУ для построения фаз. Оператором ENTRY можно задать точку входа первой фазы программы. Этот оператор имеет следующий формат:

ENTRY точка входа,

где «точка входа»— символическое имя основной или дополнительной точки входа главной процедуры. Если операнд опущен или не указан оператор ENTRY, то в качестве адреса точки входа принимается основная точка входа главной процедуры.

При отсутствии оператора ENTRY конец входной информации РЕДАКТОРА определяется оператором

/_ EXEC LNKEDT.

11.8. Печать, выполняемая РЕДАКТОРОМ

Во время своей работы РЕДАКТОР выдает на печать информацию о результатах редактирования. К этой информации относятся:

— протокол РЕДАКТОРА;
— обзор фаз;
— сообщения об ошибках во время работы РЕДАКТОРА.

Протокол РЕДАКТОРА выдается на печать, если в управляющем операторе ACTION не указан операнд NOMAP (если оператор ACTION отсутствует, подразумевается режим MAP).

Из протокола РЕДАКТОРА можно узнать, все ли модули, имена которых указаны в таблице внешних символических имен, включаются в фазу. Возможность автоматического включения модулей обеспечивается режимом РЕДАКТОРА AUTOLINC, который, если он не отменяется управляющим оператором ACTION NOAUTO, действует всегда.

После заголовка листа, который содержит имя задания и дату, печатаются операнды управляющего оператора ACTION, указанные явно или подразумеваемые по умолчанию. После этого оператора печатается список всех включенных модулей. В каждой строке печатаются LIST, AUTOLINK и имя присоединенного модуля.

Пример.

```
JOB  <имя задания>  <дата>  DISK LINKAGE EDITOR DIAGNOSTIC
                  OF  INPUT
ACTION  TAKEN  MAP
LIST  AUTOLINK  IJFFZZZZ
LIST  AUTOLINK  IJJFCIZD
LIST  AUTOLINK  IJKSYSA
```

Если РЕДАКТОР не может разрешить все внешние ссылки, он выдает в протоколе сообщение об этом.

Обзор фаз печатается по окончании редактирования, если не был указан режим NOMAP. В обзоре фаз указываются имя фазы, имена программных секций, точек входов и адреса загрузки.

Обзор фаз имеет следующие графы:

PHASE — имя фазы. Если в операторе PHASE имя фазы не указывается, то печатается PHASE***;

XFR_AD — точка входа (пусковой адрес) фазы;

LOCORE — адрес основной памяти, с которого располагается фаза (адрес загрузки фазы);

HICORE — адрес последнего байта фазы в основной памяти;

DSK_AD — дисковый адрес фазы в библиотеке абсолютных модулей в форме CCHHP, где CC — номер цилиндра; HH — номер дорожки; P — номер записи;

ESDTYPE — признак информации, характеризующей программу; CSECT — управляющая секция;

ENTRY — точка входа (ENTRY со знаком * означает неиспользованное имя входа); EXTRY — неразрешенная внешняя ссылка; COM — общая область;

LABEL — имя программной секции отредактированной программы, имя точки входа или неразрешенное внешнее имя;

LOADED — адрес загрузки программной секции или адрес точки входа;

REL_FP — значение перемещения программной секции для строк типа CSECT или длина общей области для строк типа COM.

Пример.

PHASE	XFR_AD	LOCORE	HICORE	DSK_AD	ESDTYPE	LABEL
PHASE ***	003318	003050	005AA7	30 02 2	CSECT	CARD
					CSECT	IJFFZZZZ

Заметим, что общая область указывается для РЕДАКТОРА признаком COM (в языке ПЛ/1 переменная с атрибутами STATIC EXTERNAL). COM появляется только тогда, когда программа на ПЛ/1 использует подпрограмму на Ассемблере, в которой с помощью оператора COM описана общая область.

Сообщения об ошибках во время работы РЕДАКТОРА. Ошибки, обнаруженные РЕДАКТОРОМ, выводятся на печать и подразделяются на две группы:

— ошибки, из-за которых невозможно продолжение редактирования (номера ошибок от 21801 до 21981). Печать таких ошибок сопровождается снятием задания, а перед печатью сообщения об ошибке печатается фраза

LINKAGE EDITOR CANNOT CONTINUS,

которая указывает на то, что редактирование не может быть продолжено;

— ошибки, после которых редактирование продолжается, если в операторе РЕДАКТОРА ACTION не задан режим CANCEL (но-

мера ошибок от 21001 до 21791). Перед первым сообщением о такой ошибке печатается фраза

ERROR HAS OCCURED DURING LINKAGE EDITING,

которая указывает на то, что во время редактирования была обнаружена хотя бы одна ошибка.

11.9. Средства отладки программы

Во время трансляции программы, написанной на языке ПЛ/1, с помощью средств транслятора производится синтаксический контроль, который дает возможность программисту найти и исправить ошибки практически во всех конструкциях языка. Однако эти средства не позволяют произвести отладку алгоритма описанной программистом задачи, т. е. получить информацию во время выполнения программы с конкретными данными.

Тем не менее существуют стандартные подпрограммы, которые могут быть полезны программисту при отладке алгоритма. Они обеспечивают получение информации о ходе выполнения программы (т. е. помогают проверить логику алгоритма) и проследить за промежуточными результатами решения. Следует отметить, что эти средства системозависимы, т. е. относятся только к ДОС версии 2.0 (или 2.1). Вызов этих вспомогательных подпрограмм производится, как вызов процедур-подпрограмм с помощью оператора CALL.

Печать значений переменных — подпрограмма DYNDUMP, с помощью которой можно печатать в шестнадцатиричной форме внутреннее представление заданных в ней аргументов. Вызов подпрограммы осуществляется оператором:

CALL DYNDUMP (аргумент [,аргумент] ...);

В качестве аргументов могут использоваться простые переменные, массивы, структуры, а также скалярные выражения. Максимальное число аргументов не должно быть более 12.

Данную подпрограмму целесообразно использовать для выдачи значений промежуточных результатов во время выполнения программы. При этом следует отметить, что ее выполнение требует минимальной затраты времени. При обращении к ней каждый элемент списка аргументов печатается на отдельной строке. В начале каждой строки печатается шестнадцатиричный адрес левого байта области, которая занята аргументом. Если в качестве аргумента используется имя массива или структуры, то их элементы печатаются непрерывным потоком (на одной или нескольких строках) группами по 8 шестнадцатиричных цифр независимо от длины отдельного элемента.

Пример.

```
DCL A FIXED (4,3),
    B BIT (3),
    C (6) FIXED INIT (1, 2, 3, 4, 5, 6);
A = 0.062; B = '110'B; X = -25; I = 33;
CALL DYNDUMP (A, I, B, X, C);
```

После выполнения подпрограммы будет выдано на печать следующее сообщение

```
PL/I DYNDUMP
006B60  00062C
006B54  00000021
006B62  C0
006B4C  C2190000
006B68   00001C  00 002C  0000 3C  00004C   00005C   00 006C
          1        2        3        4        5        6
```

Печать изменяющихся значений переменных — подпрограмма IJKEXHC. Данная подпрограмма осуществляет печать в шестнадцатиричной форме внутреннего представления заданных аргументов каждый раз, как их значение изменяется в результате выполнения некоторого оператора. Таким образом, с помощью этой подпрограммы можно проверить динамику выполнения алгоритма и проследить за изменениями значений заданных переменных. Обращение к подпрограмме производится оператором:

CALL IJKEXHC (аргумент [, аргумент] ...);

Аргументами могут быть переменные, массивы и структуры. Максимальное число аргументов не должно быть более 12, причем для аргументов с атрибутами BASED или DEFINED учитываются и соответствующие переменные типа указателя или идентификатора базы.

Использование этой подпрограммы требует дополнительного объема основной памяти: на саму подпрограмму приблизительно 1264 байта, для каждой переменной, появляющейся в качестве аргумента, еще 12 байтов плюс поле, содержащее имя переменной, и плюс поле, содержащее ее значение. Все эти поля выделяются в статической области памяти.

При первом вызове этой подпрограммы печатаются идентификаторы переменных (аргументов), внутренние представления их значений в шестнадцатиричной форме и для скалярных переменных их внешнее представление. Переменные с атрибутом AUTOMATIC печатаются всякий раз, когда оператор CALL IJKEXHC выполняется впервые после активизации блока. Переменные с атрибутом STATIC печатаются только тогда, когда подпрограмма выполняется в первый раз, если активизированный блок является внутренним. Если активизированный блок — внешний, переменная печатается каждый раз, когда происходит обращение к этой подпрограмме.

При всех последующих выполнениях оператора обращения к подпрограмме происходит печать значений переменных только тогда, когда это значение изменилось по сравнению с предыдущим.

Если программа содержит несколько обращений к подпрограмме IJKEXHC, то они выполняются независимо друг от друга.

Если транслятору отведено 10K основной памяти, то внутри одного блока с помощью подпрограммы IJKEXHC можно проверить не более 30 имен переменных. Для каждых дополнительных 4 K (вплоть до 46 K) можно проверить еще 30 имен.

Выдача информации на печать производится следующим образом:

— каждый раз, когда происходит обращение к подпрограмме, печатается заголовок PЛ/I EXHIBIT CHANGED, указывающий на изменение значений переменных;

— каждый элемент списка аргументов печатается на отдельной строке. Причем каждая строка печати предворяется заголовком INITIAL VALUE (начальная величина);

— для массивов и структур значения элементов выдаются в шестнадцатиричной форме непрерывным потоком, разделенным на группы по 8 шестнадцатиричных цифр.

Пример.

```
PR: PROC OPTIONS (MAIN);
    DCL A (5) INIT (1, 2, 3, 4, 5),
        B BIT (2),
        C CHAR (3),
        D FIXED (6, 3);
    X = 25; B = '01'B; C = 'OLD';
    DO K = 1 TO 3;
    D = 2.15;
    CALL IJKEXHC (K, X, D, C, A);
    D = 2.753;
    IF K = 3 THEN DO; C = 'NEW'; B = '11'B; END;
    CALL IJKEXHC (D, B, C);
    X = X + 5;
    END;
    END PR;
```

При выполнении данной программы на печать будет выдаваться следующая информация:

```
PL/I EXHIBIT CHANGED
INITIAL VALUE
K = 00000001 = 1
INITIAL VALUE
X = 42190000 = 2.50000E + 01
INITIAL VALUE
D = 0002150C = 2.150
INITIAL VALUE
C = D6D3C4 = OLD
INITIAL VALUE
A = 00000001 00000002 00000003 00000004 00000005
PL/I EXHIBIT CHANGED
INITIAL VALUE
```

```
D = 0002753C = 2.753
INITIAL  VALUE
B = 40 = 01
INITIAL VALUE
C = D6D3C4 = OLD
PL/I  EXHIBIT  CHANGED
K = 00000002 = 2
X = 421E0000 = 3.00000E + 01
PL/I  EXHIBIT  CHANGED
K = 00000003 = 3
X = 42230000 = 3.50000E + 01
PL/I  EXHIBIT  CHANGED
B = C0 = 11
C = D5C5E6 = NEW
```

Печать переходов — подпрограмма IJKTRON позволяет проверить логику выполнения программы.

Обращение к подпрограмме оператором

```
CALL IJKTRON;
```

обеспечивает распечатку информации о логике передачи управления при выполнении операторов программы, а обращение вида

```
CALL IJKTROF;
```

выключает такую распечатку.

Областью действия этих операторов является блок, в котором они явно присутствуют, а также все внутренние блоки, активизированные из данного блока. Внешняя процедура при вызове (при условии, что в момент вызова печать переходов включена) обязательно должна содержать оператор CALL IJKTRON, иначе распечатки переходов не будет.

Если для какого-то блока включена печать переходов, то выдается следующая информация:

— при входе в блок печатается метка или имя входа этого блока. Если блок не имеет метки, то печатается его внутреннее имя, присвоенное транслятором;

— при выходе из блока с помощью оператора END или RETURN выдается сообщение о выходе. Если в операторе PROCESS задан режим STMT, то печатается номер оператора END или RETURN, а также номер того оператора, которому передается управление после выхода из блока;

— для каждого выполняемого оператора перехода происходит печать значения константы или переменной типа метки, указанной в операторе GOTO. Кроме того, адрес области памяти соответствующего оператора печатается во фразе VALUE в форме шестнадцатиричного числа. Если в операторе PROCESS задан режим STMT, то печатается также номер выполненного оператора GOTO и номер оператора, которому передается управление;

— если оператор GOTO является действием программиста в операторе ON, то дополнительно к этому печатается название соответствующей ситуации, вызвавшей этот переход.

Пример.

```
1   HP: PROC OPTIONS (MAIN);
          ....
4         CALL IJKTRON;
5         IF X = 0 THEN GOTO M1;
10  M1:   ......;
11  M2:   BEGIN;
          .....
15        CALL P2 (X);
          .....
21        END;
22        ......
28        BEGIN;
29        ON ZERODIVIDE GOTO M3;
30        DO K = 1 TO L;
          .....
36        N = K/M + 2;
          .....
48        END;
49  M3:   END;
50        CALL IJKTROF;
          ....
56        END HP;
57        P2: PROCEDURE (A);
          ....
62        END P2;
```

При выполнении этой программы будет выдаваться следующая информация:

```
PL/I TRACE BRANCH TO LABEL M1, VALUE = 000040EE,
           FROM STMNT NO 0005 TO STMNT NO 0010
PL/I TRACE CONTROL HAS PASSED ENTRY NAME M2
PL/I TRACE END OF BLOCK, FROM STMNT NO 0021
           TO STMNT NO 0022
PL/I TRACE CONTROL HAS PASSED ENTRY NAME N '012F'
PL/I TRACE BRANCH ON ACCOUNT OF ON-CONDITION
           ZERODIVIDE, VALUE = 00004334, FROM
           STMNT NO 0036 TO STMNT 0049
```

В результате использования подпрограммы IJKTRON мы получили следующую информацию о ходе выполнения данной программы:

— условный оператор выполнен и управление передано оператору с меткой M1, т. е. от оператора перехода с номером 0005 к оператору с номером 0010. Значение, стоящее во фразе VALUE, представляет шестнадцатиричный адрес этого оператора;

— передано управление на точку входа M2 (активизировался обычный BEGIN блок);

— переход к внешней процедуре P2 не указан, так как в ней не присутствует оператор CALL IJKTRON;

— указаны выход из обычного блока и передачи управления от оператора END (номер 0021) к оператору 0022;

— управление передано на внутреннее имя '012F' (активизировался непомеченный BEGIN блок);

— было выполнено прерывание программы вследствие (ON ACCOUN OF ситуации ZERODIVIDE внутри обычного блока (в операторе 0036) и управление передано оператору 0049.

11.10. Сообщения о программных прерываниях во время выполнения программы

При выполнении программы могут возникнуть ситуации, из-за которых продолжение выполнения программы будет неправильным или невозможным. Такие ситуации возникают, например, в следующих случаях:

— при вводе данных, которые не могут быть преобразованы;

— при поиске в файле записи с несуществующим ключом;

— если область, отведенная под динамическую память, является недостаточной для выполнения программы;

— если значения аргументов для встроенных функций выходят за границы допустимых областей по причинам, которые программист заранее не предусмотрел.

Операционная система обрабатывает различные прерывания. Известно, что при помощи префикс-ситуаций можно установить контроль за некоторыми ситуациями (прерываниями). Тогда, используя оператор ON, при возникновении ситуации (прерывания), можно осуществить передачу управления той части программы, которая производит обработку этой ситуации.

Если, несмотря на все принятые меры, возникает какая-то непредвиденная ситуация, происходит прерывание выполнения программы и выдается сообщение об ошибке, имеющее следующий формат:

5L00I *ccdddddd aaaaaa* ERROR.

Если в операторе PROCESS задан режим STMT, то сообщение об ошибке принимает вид

5L00I *ccdddddd aaaaaa* ERROR STATEMENT NUMBER *nnnn*.

где 5L00I — признак для сообщений компонента транслятора ПЛ/1;

cc — код ошибки;

dddddd — адрес блока ввода—вывода, если сообщение об ошибке относится к файлу (в противном случае это нули);

aaaaaa — абсолютный адрес команды, при выполнении которой обнаружена ошибка;

nnnn — номер оператора, при выполнении которого возникла ситуация прерывания.

Стандартным действием системы для большинства ситуаций является вывод сообщения об ошибке и вызов ситуации ERROR. Для

Таблица 11.5

Код	Ситуация	Причина	Стандартное действие системы
01	OVERFLOWW	Результат операции с плавающей точкой превышает максимальное допустимое значение	Вызывается ситуация ERROR и происходит прерывание программы
02	UNDERFLOW	Результат операции с плавающей точкой меньше минимально допустимого значения	Следует продолжение операции, в качестве результата используется нуль в форме с плавающей точкой
03	ZERODIVIDE	Попытка произвести деление на нуль (с фиксированной или плавающей точкой)	Вызывается ситуация ERROR и происходит прерывание программы
04	FIXEDOVERFLOW	Разрядность результата с фиксированной точкой превышает 15 десятичных цифр или 31 двоичную	То же
05	SIZE	При выполнении операции присваивания переменной с фиксированной точкой происходит потеря значащих цифр	»
06	CONVERSION	Попытка совершить недопустимое преобразование над данными типа строки символов	»
09	ERROR	Если возникла исключительная ситуация, при которой нельзя определить точную причину ее возникновения. Как правило, она возникает как результат для большинства исключительных ситуаций (ситуация-следствие)	»
0A	ENDFILE	Попытка читать файл со способом организации CONSECUTIVE или INDEXED при последовательном методе доступа после достижения конца файла	»
0B	KEY	Указанный в операторе ввода или вывода ключ не отвечает специальным требованиям (например, неправильная длина ключа и т. д.)	»

Продолжение табл. 11.5

Код	Ситуация	Причина	Стандартное действие системы
OC	TRANSMIT	Постоянная ошибка передачи	Программное прерывание
OE	RECORD	При вводе—выводе записей действительная длина записи не соответствует длине записи, указанной в атрибуте ENV	Вызывается ситуация ERROR и происходит программное прерывание

ситуаций, обрабатываемых средствами ПЛ/1, сообщение об ошибке выдается только в том случае, если они включены и на них распространяется стандартное действие системы. Сообщения об ошибках выводятся на устройства SYSLST, если в операторе PROCEDURE главной процедуры программы не задан режим ONSYSLOG, в противном случае они выводятся на пультовую пишущую машинку оператора (устройство SYSLOG.)

Пример.

При выполнении программы произошло прерывание и выдано следующее сообщение:

```
5L00I 06000000 0033EC ERROR STATEMENT NUMBER 0011
```

Это сообщение указывает, что при выполнении оператора программы с номером 11 возникла ситуация CONVERSION, что указывается кодом ошибки 06.

Для вычислительных ситуаций и ситуаций ввода—вывода сообщение об ошибке может быть выдано только в том случае, если такая ситуация включена. При этом если в момент программного прерывания в операторе ON задано действие программиста, то сообщение не выдается, а происходит передача управления той части программы, в которой эта ситуация будет обработана. Если оператор ON для данной ситуации не используется или в нем указана спецификация действия SYSTEM, то выдается рассмотренное выше сообщение с соответствующим кодом.

В табл. 11.5 приведены коды сообщений с указанием причины возникновения ошибки и стандартного действия системы.

Некоторые ошибки, допущенные при программировании, при выполнении программы могут вызывать прерывания. Такие прерывания могут возникнуть: при программных прерываниях, ошибках организации, ошибках при обращении к встроенным функциям, ошибках ввода—вывода.

Причины и коды ошибок, вызывающие эти прерывания, приведены в приложении 6.

11.11. Создание структуры с перекрытием

Программа, полученная в результате редактирования, имеет абсолютный формат, т. е. она без дополнительной обработки может быть вызвана из библиотеки абсолютных модулей и выполнена. В простейшем случае абсолютный модуль состоит из одной программной фазы. Однако большие по объему программы целесообразно делить на части, которые могли бы обрабатываться последовательно и сменяли одна другую в основной памяти. В этом случае может быть создана программа из нескольких рабочих фаз, часть из которых находится в библиотеке абсолютных модулей и вызывается оттуда по мере необходимости их выполнения.

Такая структура многофазной программы называется структурой с перекрытием. Использование структур с перекрытием позволяет минимизировать необходимый для программы объем основной памяти. Получение структуры с перекрытием обеспечивается использованием управляющего оператора РЕДАКТОРА—PHASE и специального оператора языка ПЛ/1 — OVERLAY.

На этапе редактирования происходит формирование отдельных частей программы с перекрытием. Для этой цели используется оператор PHASE, в котором указывается имя и адрес загрузки редактируемой программной фазы. Фаза программы, которая загружается операционной системой, получает от нее управление и постоянно находится в основной памяти при выполнении всей программы, называется корневой фазой и имеет адрес загрузки ROOT. Для этой фазы оператор имеет вид

....PHASE имя фазы, ROOT.

Остальные программные фазы загружаются в основную память после корневой фазы и могут сменять друг друга. Они называются фазами перекрытия и имеют адрес загрузки * или «имя». Таким образом, оператор будет записываться следующим образом:

$$\text{....PHASE имя фазы, } \left\{ \begin{matrix} * \\ \text{имя} \end{matrix} \right\},$$

где * указывает, что в качестве адреса загрузки фазы РЕДАКТОР должен использовать адрес первого двойного слова после конца предыдущей фазы;

«имя» означает, что РЕДАКТОР должен использовать в качестве адреса загрузки адрес фазы, имя которой стоит в операторе PHASE.

Для загрузки в основную память нужной фазы в исходной программе на языке ПЛ/1 должен быть использован оператор CALL, имеющий следующий формат:

[метка:] CALL OVERLAY ('имя фазы');

После того как нужная фаза загружена, управление ей может быть передано либо как процедуре-подпрограмме с помощью дру-

гого оператора CALL, либо обращением к ней как к процедуре-функции.

Пример. Пусть для решения некоторой задачи составлена программа, состоящая из главной процедуры HP и двух внешних процедур P1 и P2. Процедуры P1 и P2 вызываются главной процедурой HP и являются в этом смысле зависимыми. Возможен случай, когда область памяти, которую можно отвести для программы, будет недостаточна для размещения этих трех процедур (рис. 11.7, *а*).

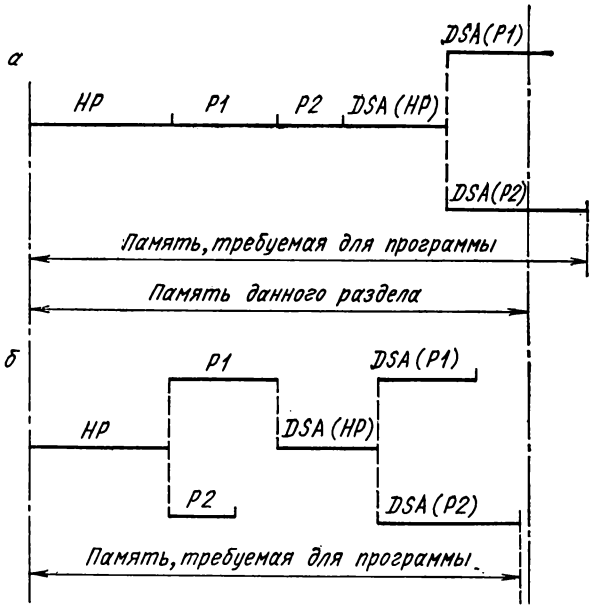


Рис. 11.7. Размещение программы в памяти без перекрытия (*а*) и с перекрытием (*б*).

Предположим, что процедуры P1 и P2 не зависят друг от друга, т. е. процедура P2 может отсутствовать в основной памяти при работе процедуры P1 и наоборот. Следовательно, процедуры P1 и P2 могут сменять друг друга в основной памяти, т. е. перекрывать одна другую. Для осуществления этого процедуры HP, P1 и P2 должны представлять собой отдельные фазы многофазовой программы, причем фазы, соответствующие процедурам P1 и P2, должны быть отредактированы с одного адреса (рис. 11.7, *б*).

Создание структуры с перекрытием следует начинать на стадии программирования, т. е. строить программу таким образом, чтобы отдельные ее части могли быть включены в различные фазы.

В нашем примере корневая фаза содержит главную процедуру HP, а перекрывающие друг друга фазы — процедуры P1 и P2 соответственно. Пусть с помощью оператора PHASE им присвоены следующие имена: PROGH, PROG1, PROG2.

Предположим, что в нашем примере процедура P1 вызывается в главной процедуре как процедура-подпрограмма, а процедура P2 — как процедура-функция. С учетом всех указанных условий задание на выполнение программы с перекрытием будет иметь вид

```
//_JOB  PROG
//_OPTION  LINK
  _PHASE  PROGH,ROOT
//_EXEC  PL/I
 HR:  PROCEDURE  OPTIONS  (MAIN);

      ....
        CALL  OVERLAY  ('PROG1');
        CALL  P1  (список аргументов);
      ...
        CALL  OVERLAY  ('PROG2');
   X = Y + P2  (аргументы);
   END  HP;
/*
  _PHASE  PROG1,*
// _ EXEC  PL/I
  P1:  PROCEDURE  (список параметров);
       ...
  END  P1;
/*
  _ PHASE  PROG2,  PROG1
//_EXEC  PL/I
  P2:  PROCEDURE  (список  параметров)  атрибуты
       функции;
       ...
  RETURN  (выражение);
  END  P2;
/*
//_EXEC  LNKEDT
//_EXEC  PROGH
/&
```

Необходимо отметить, что имена элементов данных, которые используются во всех фазах структуры с перекрытием, должны объявляться явно с атрибутом EXTERNAL.

При использовании файлов надо иметь в виду следующее:

— если файл явно объявлен в фазе перекрытия, но не объявлен в корневой фазе, то он должен быть закрыт в этой фазе прежде, чем она будет перекрыта. Это ограничение не нужно, если файл объявлен также и в корневой фазе;

— если в фазах, отличных от корневой, используются стандартные файлы, эти файлы должны использоваться также или в корневой фазе, или в фазе, которая в дальнейшем не будет перекрыта. Если это не выполняется, то в корневую фазу с помощью оператора INCLUDE необходимо включить соответствующие модули: IJKSYSI (для стандартного вводного файла) и IJKSYSA (для стандартного выводного файла).

Атрибуты результатов основных арифметических операций

Таблица 1

Атрибуты результатов операций сложения и вычитания

Операнд 2	Операнд 1			
	FIXED DEC (p_1, q_1)	FLOAT DEG (p_1)	FIXED BIN (p_1, q_1)	FLOAT BIN (p_1)
FIXED DEC (p_2, q_2)	FIXED DEC (p, q) $p=1+\max(p_1-q_1, p_2-q_2)+\max(q_1, q_2)$, $q=\max(q_1, q_2)$	FLOAT DEC (p) $p=\max(p_1, p_2)$	FIXFD BIN (p, q) $p=1+\max(p_1-q_1, r-s)+\max(q_1, s)$ $q=\max(q_1, s)$, $r=1+p_2$. 3.32 $s=q_2$. 3,32	FLOAT BIN (p) $p=\max(p_1, r)$ $r=p_2$. 3.32
FLOAT DEG (p_2)	FLOAT DEC (p) $p=\max(p_1, p_2)$	FLOAT DEG (p) $p=\max(p_1, p_2)$	FLOAT BIN (p) $p=\max(p_1, r)$ $r=p_2$. 3.32	FLOAT BIN (p) $p=\max(p_1, r)$ $r=p_2$. 3.32
FIXFD BIN (p_2 q_2)	FIXED BIN (p, q) $p=1+\max(r-s, p_2-q_2)+\max(s, p_2)$, $q=\max(s, q_2)$, $r=1+p_1$. 3.32, $s=q_1$. 3.32	FLOAT BIN (p) $p=\max(r, q_2)$, $r=p_1$. 3.32	FIXED BIN (p, q) $p=1+\max(p-q_1, p_2-q_2)+\max(q_1, q_2)$, $q=\max(q_1, q_2)$	FLOAT BIN (p) $p=\max(p_1, p_2)$
FLOAT BIN (p_2)	FLOAT BIN (p) $p=\max(r, p_2)$, $r=p_1$. 3.32	FLOAT BIN (p) $p=\max(r, p_2)$, $r=p_1$. 3.32	FLOAT BIN (p) $p=\max(p_1, p_2)$	FLOAT BIN (p) $p=\max(p_1, p_2)$

Таблица 2

Атрибуты результатов операции умножения

Операнд 2	Операнд 1			
	FIXED DEC (p_1, q_1)	FLOAT DEC (p_1)	FIXED BIN (p_1, q_1)	FLOAT BIN (p_1)
FIXED DEC (p_2, q_2)	FIXED DEC (p, q) $p=p_1+p_2+1$, $q=q_1+q_2$	FLOAT DEC (p) $p=\max(p_1, p_2)$	FIXED BIN (p, q) $p=p_1+r+1$, $q=q_1+s$, $r=1+p_2$. s.32, $s=q_2$. 3.32	FLOAT BIN (p) $p=\max(p_1, r)$, $r=p_2$. 3.32
FLOAT DEC (p_2)	FLOAT DEC (p) $p=\max(p_1, p_2)$	FLOAT DEC (p) $p=\max(p_1, p_2)$	FLOAT BIN (p) $p=\max(p_1, r)$, $r=p_2$. 3.32	FLOAT BIN (p) $p=\max(p_1, r)$, $r=p_2$. 3.32
FIXED BIN (p_2, q_2)	FIXED BIN (p, q) $p=r+p_2+1$, $q=s+q_2$ $r=1+p_2$. 3.32, $s=q_1$. 3.32	FLOAT BIN (p) $p=\max(r, p_2)$, $r=p_1$. 3.32	FIXED BIN (p, q) $p=p+p_2+1$, $q=q_1+q_2$	FLOAT BIN (p) $p=\max(p_1, p_2)$
FLOAT BIN (p_2)	FLOAT BIN (p) $p=\max(r, p_2)$, $r=p_1$. 3.32	FLOAT BIN (p) $r=\max(r, p_2)$, $p=p_1$. 3.32	FLOAT BIN (p) $p=\max(p_1, p_2)$	FLOAT BIN (p) $p=\max(p_1, p_2)$

Таблица 3

Атрибуты результатов операции деления

Операнд 2	Операнд 1			
	FIXED DEC (p_1, q_1)	FLOAT DEC (p_1)	FIXED BIN (p_1, q_1)	FLOAT BIN (p_1)
FIXED DEC (p_2, q_2)	FIXED DEC (p, q) $p=15$, $q=15-((p_1-q_1)+q_2)$	FLOAT DEC (p) $q=\max(p_1, p_2)$	FIXED BIN (p, q) $p=31$, $q=31-((p_1-q_1)+s)$, $s=q_2$. 3,32	FLOAT BIN (p) $p=\max(p_1, r)$, $r=p_2$. 3.32
FLOAT DEC (p_2)	FLOAT DEC (p) $p=\max(p_1, p_2)$	FLOAT DEC (p) $p=\max(p_1, p_2)$	FLOAT BIN (p) $p=\max(p_1, r)$, $r=p_2$. 3.32	FLOAT BIN (p) $p=\max(p_1, r)$, $r=p_2$. 3.32
FIXED BIN (p_2, q_2)	FIXED BIN (p, q) $p=31$, $q=31-((r-s)+q_2)$, $r=1+p_1$. 3.32, $s=q_1$. 3.32	FLOAT BIN (p) $p=\max(r, p_2)$, $r=p_1$. 3.32	FIXED BIN (p, q) $p=31$, $q=31-((p_1-q_1)+q_2)$	FLOAT BIN (p) $p=\max(p_1, p_2)$
FLOAT BIN (p_2)	FLOAT BIN (p) $p=\max(r, p_2)$. $r=p_1$. 3.32	FLOAT BIN (p) $p=\max(r, p_2)$ $r=p_1$. 3.32	FLOAT BIN (p) $p=\max(p_1, p_2)$	FLOAT BIN (p) $p=\max(p_1, p_2)$

Таблица 4

Атрибуты результатов операции возведения в степень

Операнд 1 (основание)	Операнд 2 (экспонента)	Результат (степень)
FIXED DEC (p_1, q_1) (случай 1)	Целая константа без знака со значением n	FIXED DEC (p, q) $p \leqslant 15$ $p = (p_1 + 1)n - 1$ $q = q_1 . n$
FIXED BIN (p_1, q_1) (случай 2)	То же	FIXED BIN (p, q) $p \leqslant 31$ $p = (p_1 + 1) \cdot n - 1$ $q = q_1 . n$
FIXED DEC (p_1, q_1) или FLOAT DEC (p_1) (случай 3)	FIXED DEC (p_2, q_2) или FLOAT DEC (p_2)	FLOAT DEC (p) (если не применим случай 1) $p = \max(p_1, p_2)$
FIXED BIN (p_1, q_1) (случай 4)	FIXED BIN (p_2, q_2) или FLOAT BIN (p_2)	FLOAT BIN (p) (если не применим случай 2) $p = \max p_1$, CELL (p_2. 3.32))

Управляющие символы CTLASA, CTLYES

Управляющие символы CTLASA приведены в табл. 1. Управляющие знаки CTLYES для ввода — вывода перфокарт и для печати приведены в табл. 2 и 3 соответственно.

Таблица 1

Управляющий символ	Функция
␣	Продвижение строки перед печатью
0	Двойное продвижение строки перед печатью
—	Тройное продвижение строки перед печатью
+	Блокировка перевода строки перед печатью
	Продвижение бумаги перед печатью:
1	в дорожке 1
2	в дорожке 2
3	в дорожке 3
4	в дорожке 4
5	в дорожке 5
6	в дорожке 6
7	в дорожке 7
8	в дорожке 8
9	в дорожке 9
A	в дорожке 10
B	в дорожке 11
C	в дорожке 12
V	Выбрать приемный карман 1
W	Выбрать приемный карман 2

Таблица 2

Управляющие знаки CTLYES для ввода—вывода перфокарт

Комбинация битов управляющего знака	Комбинация пробивок	Функция
0000 0001	12, 9, 1	Выбрать приемный карман 1
0100 0001	12, 0, 9, 1	Выбрать приемный карман 2
1000 0001	12, 0, 1	Выбрать приемный карман 3

Таблица 3

Управляющие знаки CTLYES для печати

Комбинация битов управляющего знака	Комбинация пробивок	Функция
0000 0001	12, 9, 1	Печать (нет автоматического продвижения строки)
0000 0001	12, 9, 8, 1	Печать, затем продвижение строки
0001 0001	11, 9, 1	Печать, затем продвижение двух строк
0001 1001	11, 9, 8, 1	Печать, затем продвижение трех строк
1000 1001	12, 0, 9	Печать, затем продвижение до пробивки:
		в дорожке 1
1001 0001	12, 11, 1	в дорожке 2
1001 1001	12, 11, 9	в дорожке 3
1010 0001	11, 0, 1	в дорожке 4
1010 1001	11, 0, 9	в дорожке 5
1011 0001	12, 11, 0, 1	в дорожке 6
1011 1001	12, 11, 0, 9	в дорожке 7
1100 0001	12, 1	в дорожке 8
1100 1001	12, 9	в дорожке 9
1101 0001	11, 1	в дорожке 10
1101 1001	11, 9	в дорожке 11
1110 0001	11, 0, 9, 1	в дорожке 12
0000 1011	12, 9, 8, 3	Простое продвижение строки
0001 0011	11, 9, 3	Пропуск
0001 1011	11, 9, 8, 3	двух строк
1000 1011	12, 0, 8, 3	трех строк
		Продвижение до пробивки:
1001 0011	12, 11, 3	в дорожке 1
1001 1011	12, 11, 8, 3	в дорожке 2
1010 0011	11, 0, 3	в дорожке 3
1010 1011	11, 0, 8, 3	в дорожке 4
1011 0011	12, 11, 0, 3	в дорожке 5
1011 1011	12, 11, 0, 8, 3	в дорожке 6
1100 0011	12, 3	в дорожке 7
1100 1011	12, 0, 9, 8, 3	в дорожке 8
1101 0011	11, 3	в дорожке 9
1101 1011	12, 11, 9, 8, 3	в дорожке 10
1110 0011	0, 3	в дорожке 11
		в дорожке 12

ПРИЛОЖЕНИЕ 3

Атрибуты описания файлов

В данном положении приняты следующие обозначения:
A — выбор из указанных возможностей;
E — указание в этом месте или в операторе OPEN;
O — возможное применение;
S — атрибут по умолчанию;
X — указание обязательно;
cccc — номер перфокарточного устройства;
tttt — номер устройства для файла на МЛ;
pppp — номер печатающего устройства;
dddd — номер устройства для файлов на МД.

Таблица 1

Атрибуты и режимы файлов при передаче, ориентированной на поток

Атрибуты и режимы файлов	Вводной файл			Выводной файл без атрибута PRINT				Выводной файл с атрибутом PRINT		
	Перфокарты	Магнитная лента	Диски	Перфокарты	Печать	Магнитная лента	Диски	Печать	Магнитная лента	Диски
Имя файла	X	X	X	X	X	X	X	X	X	X
FILE	X	X	X	X	X	X	X	X	X	X
STREAM	S	S	S	S	S	S	S	S	S	S
INPUT	X	X	X							
OUTPUT				X	X	X	X			
PRINT								X	X	X
ENVIRONMENT	X	X	X	X	X	X	X	X	X	X
MEDIUM	X	X	X	X	X	X	X	X	X	X
SYSIPT	A	A	A							
SYSPCH				A		A	A			
SYSLST					A	A	A	A	A	A
SYS*nnn*	A	A	A	A	A	A	A	A	A	A
cccc	X			A						
pppp					A			X		
tttt		X				X			X	
dddd			X				X			X
F (длина блока)	X	X	X	X	X	X	X	X	X	X
BUFFERS (1)	S	S	S	S	S	S	S	S	S	S
BUFFERS (2)	0	0	0	0	0	0	0	0	0	0
LEAVE		0				0			0	
NOLABEL		0				0			0	
NOTAPEMK						0			0	0
VERIFY							0			0
COSECUTIVE	S	S	S	S	S	S	S	S	S	S
EXTERNAL	S	S	S	S	S	S	S	S	S	S

Таблица 2

Атрибуты и режимы файлов при передаче, ориентированной на записи
Последовательные файлы с буфером (BUFFERED)

Атрибуты и режимы файлов	Вводной файл				Выводной файл				
	Перфокарты	Магнитная лента	МЛ с оборотным чтением	Диски	Перфокарты	Печать	Магнитная лента	Диски	Файл для обновления на дисках
Имя файла	X	X	X	X	X	X	X	X	X
FILE	X	X	X	X	X	X	X	X	X
RECORD	X	X	X	X	X	X	X	X	X
INPUT	X	X	X						
OUTPUT					X	X	X	X	
UPDATE									X
SEQUENTIAL	S	S	S	S	S	S	S	S	S
BACKWARDS			X						
BUFFERED	S	S	S	S	S	S	S	S	S
ENVIRONMENT	X	X	X	X	X	X	X	X	X
MEDIUM	X	X	X	X	X	X	X	X	X
SYSIPT	A	A		A					
SYSPCH					A		A	A	
SYSLST						A	A	A	
SYS*nnn*	A	A	X	A	A	A	A	A	X
cccc	X				A				
pppp						A			
tttt		X	X				X		
dddd				X				X	X
F (длина блока)	X	A	A	A	X	X	A	A	A

Продолжение табл. 2

Атрибуты и режимы файлов	Вводный файл				Выводной файл				
	Перфокарты	Магнитная лента	МЛ с оборотным чтением	Диски	Перфокарты	Печать	Магнитная лента	Диски	Файл для обновления на дисках
F (длина блока, длина записи)		A	A	A			A	A	A
BUFFERS (1)	S	S	S	S	S	S	S	S	S
BUFFERS (2)	0	0	0	0	0	0	0	0	0
CTLASA/CTLYES					0	0			
LEAVE		0	0				0		
NOLABEL		0	0				0		
NOTAPEMK							0		
COSECUTIVE	S	S	S	S	S	S	S	S	S
EXTERNAL	S	S	S	S	S	S	S	S	S
VERIFY								0	0

Таблица 3

Последовательные файлы без буфера (UNBUFFERED)

Атрибуты и режимы файлов	Вводной файл		Выводной файл		Файл для обновления на дисках
	Магнитная лента	Диски	Магнитная лента	Диски	
Имя файла	X	X	X	X	X
FILE	X	X	X	X	X
RECORD	X	X	X	X	X
INPUT	X	X			
OUTPUT			X	X	
UPDATE					X
SEQUENTIAL	S	S	S	S	S
BACKWARDS	O				
UNBUFFERED	X	X	X	X	X
ENVIRONMENT	X	X	X	X	X
MEDIUM	X	X	X	X	X
SYS*nnn*	X	X	X	X	X
tttt	X		X		
dddd		X		X	X
F (длина блока)	X	X	X	X	X
LEAVE	0		0		
NOLABEL	0		0		
VERIFY				0	0
CONSECUTIVE	S	S	S	S	S
EXTERNAL	S	S	S	S	S

Таблица 4

Региональные файлы

Атрибуты и режимы файла	Файл REGIONAL			Файл REGIONAL		
	вводной	выводной	для обновления	вводной	выводной	для обновления
Имя файла	X	X	X	X	X	X
FILE	X	X	X	X	X	X
RECORD	X	X	X	X	X	X
INPUT	X			X		
OUTPUT		X			X	
UPDATE			X			X
DIRECT	X	X	X	X	X	X
KEYED	X	X	X	X	X	X
ENVIRONMENT	X	X	X	X	X	X
MEDIUM	X	X	X	X	X	X
SYS*nnn*	X	X	X	X	X	X
dddd	X	X	X	X	X	X
F (длина блока)	X	X	X	X	X	X
VERIFY		0	0		0	0
REGIONAL (1)	X	X	X			
REGIONAL (3)				X	X	X
KEYLENGTH (n)				X	X	X
EXTENTNUMBER (n)	0	0	0	0	0	0
EXTERNAL	S	S	S	S	S	S

Таблица 5

Индексно-последовательные файлы

Атрибуты и режимы файлов	Файл с последовательным доступом			Файл с прямым доступом	
	вводной	выводной	для обновления	вводной	для обновления
Имя файла	X	X	X	X	X
FILE	X	X	X	X	X
RECORD	X	X	X	X	X
INTPUT	X			X	
OUTPUT		X			
UPDATE			X		X
SEQUENTIAL	S	S	S		
DIRECT				X	X
KEYED	X	X	X	X	X
BUFFERED	S	S	S		
ENVIRONMENT	X	X	X	X	X
MEDIUM	X	X	X	X	X
SYS*nnn*	X	X	X	X	X
dddd	X	X	X	X	X
F (длина блока)	A	A	A	A	A
F (длина блока, длина записи)	A	A	A	A	A
VERIFY		0	0		0
INDEXED	X	X	X	X	X
KEYLENGTH (*n*)	X	X	X	X	X
EXTENTNUMER (*n*)	X	X	X	X	X
INDEXMULTIPLE	O	O	O	O	O
OFTRACKS (*n*)		O			O
KEYLOG (*n*)	O	O	O	O	O
EXTERNAL	S	S	S	S	S

Ситуация прерывания

Таблица 1

Вычислительные ситуации

Ситуация	Причина	Состояние	Стандартное действие системы	Спецификация действия
CONVERSION	Попытка выполнения неразрешенного преобразования с данными типа символьной строки при внутренней обработке или вводе — выводе потоком	Включено	ERROR как результат ситуации и прерывания программы	Оператор GOTO; элемент данного содержит свое первоначальное значение
		Выключено	Специфицированное состояние не выполняется. Программа продолжается со следующего оператора или следующего данного. Данное содержит свое первоначальное значение	
FIXEDOVERFLOW	Разрядность результата операции с фиксированной точкой превышает разрешенный максимум (15 десятичных или 31 двоичные цифры). Не встречается при GET	Включено	ERROR как результат ситуации и прерывания программы	Оператор GOTO; данное содержит свое первоначальное значение. Пустой оператор: продолжение операции с фиксированной точкой. Лишние цифры слева будут потеряны
		Выключено	Специфицированное действие не выполняется. Продолжение операции описывается как «пустой оператор»	
OVERFLOW	Результат операции с плавающей точкой получает большое значение ($7{,}237 \times 10^{75}$ или 2^{252}). Не встречается при GET	Включено	ERROR как результат ситуации и прерывания программы	Оператор GOTO данное содержит свое первоначальное значение. Пустой оператор: продолжение операции с неопределенным значением с плавающей точкой
		Выключено	Специфицированное действие не происходит Продолжение операции описывается как «пустой оператор»	

Продолжение табл. 1

Ситуация	Причина	Состояние	Стандартное действие системы	Спецификация действия
UNDERFLOW	Результат некоторой операции с плавающей точкой получает малое значение ($5{,}4 \cdot 10^{-78}$ или 2^{-260})	Включено	Продолжение операции со значением переменной, равной нулю	Оператор GOTO: данные содержат свое первоначальное значение. Пустой оператор: стандартное действие системы
		Выключено	Специфицированное действие не выполняется. Продолжение, как при включенном стандартном действии системы	
SIZE	При операции присваивания происходит потеря лишних значащих цифр (не хватает разрядности, указанной в описании переменной)	Включено	ERROR как результат ситуации и прерывания программы	Оператор GOTO: данное содержит свое первоначальное значение. Пустой оператор: продолжение операции со следующими значениями с фиксированной точкой, при FIXED DECIMAL: а) нечетная разрядность — все лишние значащие цифры будут потеряны; б) четная разрядность — правая из лишних значащих цифр еще останется; —FIXED BINARY — точный результат (до 31 цифры); PICTURE — знак минус или лишние цифры теряются. Особый случай при выводе PUT: все позиции, соответствующие формату F, при выводе заполняются звездочками
			Специфицированное действие не выполняется. Продолжение, как при «пустом операторе»	

Продолжение табл. 1

Ситуация	Причина	Состояние	Стандартное действие системы	Спецификация действия
ZERODIVIDE	Попытка выполнения деления данных с фиксированной или плавающей точкой на нуль	Включено	ERROR как результат ситуации и прерывания программы	Оператор GOTO: данные содержат свое первоначальное значение. Пустой оператор: продолжение деления со следующим значением из внутренней области результата: а) деление с плавающей точкой или двоичное деление с фиксированной точкой, делимое стоит в области результата; б) десятичное деление с фиксированной точкой не определено
		Выключено	Специфицированное действие не будет выполняться. Продолжение, как при «пустом операторе»	

Таблица 2

Ситуации ввода—вывода

Ситуация	Причина	Состояние	Стандартное действие системы	Спецификация действия
ENDFILE	Попытка читать файл, описанный с атрибутами CONSECUTIVE или INDEXED после достижения конца файла	Включено	ERROR как результат ситуации и прерывание программы	Оператор GOTO: непосредственно после этой ситуации файл нужно закрыть. Пустой оператор: не разрешен
ENDPAGE	Оператор PUT для файла с атрибутом PRINT пытается печатать новую строку за пределом, указанным для текущей страницы	»	Продвижение на новую страницу и продолжение выполнения оператора	Оператор GOTO: а) печатает итоговую строку и переходит на новую страницу с помощью режимов PAGE или LINE; б) на этой же странице печатается дальше. Пустой оператор: оператор PUT продолжает печатать дальше на этой же странице
KEY	Указанный в операторе ввода — вывода ключ не соответствует специальным требованиям (в зависимости от файла)	»	ERROR как результат ситуации и прерывание программы	Оператор GOTO: заметим, что прерывание программы для одного и того же имени файла можно встретить для следующих операторов: READ, WRITE и REWRITE (пустой буфер), а также для операторов CLOSE или END. Соответственно можно сделать отдельную программу обработки для каждого из этих случаев. Пустой оператор: не разрешен

Продолжение табл. 2

Ситуация	Причина	Состояние	Стандартное действие системы	Спецификация действия
RECORD	При вводе или выводе, ориентированном на записи, длина записи, заданная в соответствующем режиме атрибута ENVIRONMENT (встречается только для файла BUFFERED), не соответствует фактической длине переменной	Включено	ERROR как результат ситуации и прерывание программы	Оператор GOTO: заметим, что следующим оператором ввода — вывода будет обрабатываться новая запись. Будет передана короткая или длинная запись (заполнение неопределенным содержимым или отсечение) Пустой оператор: запись будет передана (смотри оператор GOTO). Продолжение со следующего оператора
TRANSMIT	Постоянная ошибка передачи	Включено	ERROR как результат ситуации и прерывание программы	Оператор GOTO: дальнейшая обработка зависит от специальных случаев. Пустой оператор: а) передача, ориентированная на поток: продолжение оператора GET или PUT со следующего данного; б) передача, ориентированная на записи: продолжение программы со следующего оператора

Основные операторы программы управления заданиями

JOB — начать задание. Оператор JOB указывает начало последовательности управляющих операторов нового задания.

//—JOB имя задания [комментарии],

где «имя задания» состоит из 1—8 символов;

«комментарии» — пояснительный текст (например, целесообразно указывать фамилию программиста).

/& — конец задания. Этот оператор должен быть последним в пакете операторов каждого задания

/& [комментарии]

Между символом и комментариями должно быть не менее одиннадцати пробелов.

EXEC — выполнить программу. Оператор EXEC информирует программу УПРАВЛЕНИЯ ЗАДАНИЯМИ о конце управляющих операторов шага задания и указывает на необходимость начать выполнение проблемной программы. По оператору EXEC загружается первая (или единственная) фаза программы и передается управление в точку входа (заданную оператором EXEC или предполагаемую по умолчанию)

//—EXEC[имя фазы],

где «имя фазы» — фазы из библиотеки абсолютных модулей (от 1 до 8 символов).

Если имя фазы не указано, в основную память загружается и затем выполняется фаза которая непосредственно перед этим была создана РЕДАКТОРОМ и временно помещена в библиотеку абсолютных модулей.

ASSGN — назначить логическому устройству конкретное физическое устройство

$$\text{//—ASSGN SYSxxx, } \begin{Bmatrix} \text{X'}cuu\text{'} \\ \text{UA} \\ \text{IGN} \end{Bmatrix} \text{[,ALT],}$$

где SYSxxx — символическое имя логического устройства;

X'*cuu*' — адрес физического устройства в шестнадцатиричном коде;

c — номер канала;

uu — номер устройства в канале;

UA — назначение для логического устройства отменяется. Попытка выполнить операцию на логическом устройстве, назначение для которого отменено, вызовет снятие задания;

IGN — назначение для логического устройства отменяется. Параметр IGN не допускается для логических устройств SYSRDR, SYSIPT и SYSIN;

ALT — назначение сменной магнитной ленты, которая будет использоваться тогда, когда емкость основной магнитной ленты, назначенной для этого же логического устройства, будет исчерпана. Параметр ALT не допускается для логических устройств SYSRDR, SYSIPT и SYSIN.

OPTION — установить режим. Оператор устанавливает режим работы программы УПРАВЛЕНИЯ ЗАДАНИЯМИ и некоторых обрабатывающих программ (при генерации системы устанавливаются стандартные режимы работы):

//—OPTION режим [, режим [, режим] ...],

где режим — один из следующих параметров:

LIST — режим, при котором транслятор осуществляет вывод исходного модуля на SYSLST;

SYM — режим, при котором транслятор выводит на SYSLST таблицу всех встречающихся в программе идентификаторов с принадлежащими им атрибутами;

ERRS — режим, при котором транслятор выводит на SYSLST список всех ошибок, содержащихся в исходном модуле;

XREF — режим, при котором транслятор выводит на SYSLST таблицу перекрестных ссылок;

DECK — режим, при котором транслятор осуществляет вывод объектного модуля на SYSPCH;

LISTX — режим, при котором транслятор выводит на SYSLST программный модуль в ассемблерной форме;

LINK — режим, при котором транслятор записывает объектный модуль на SYSLNK для обработки его РЕДАКТОРОМ. РЕДАКТОРУ этот режим указывает, что программа должна быть временно записана в библиотеку абсолютных модулей (БАМ);

CATAL — режим, который вызывает каталогизацию программы в библиотеку абсолютных модулей по завершению работы РЕДАКТОРА. Указание этого режима автоматически вызывает установку режима LINK;

DUMP — режим, при котором в случае снятия задания (из-за ошибок) на SYSLST выводится содержимое областей основной памяти, используемых управляющей программой и разделом, в котором выполнялась проблемная программа, и содержимое общих регистров;

LOG — режим, при котором осуществляется вывод операторов УПРАВЛЕНИЯ ЗАДАНИЯМИ на SYSLST.

Каждый из режимов, кроме CATAL, может отменяться с помощью приставки NO.

PAUSE — организовать паузу. Этот оператор организует паузу немедленно, сразу после его ввода. Возобновляется обработка потока управляющих операторов по директиве УПРАВЛЕНИЯ ЗАДАНИЯМИ «конец текста» («КТ»):

```
//—PAUSE [комментарии]
```

RESET — восстановить назначения. Оператор восстанавливает стандартные назначения устройств ввода — вывода:

```
//—RESET {SYS   }
       {PROG  }
       {ALL   }
       {SYSxxx},
```

где SYS восстанавливает стандартные назначения всех системных логических устройств;

PROG восстанавливает стандартные назначения всех логических устройств программиста;

ALL восстанавливает стандартные назначения всех логических устройств;

SYSxxx восстанавливает стандартное назначение указанного логического устройства

DLBL — *запомнить информацию о метках набора данных на дисках.* Оператор сообщает системе информацию, необходимую для создания или проверки меток набора данных на дисках. В нем дается общая характеристика наборов данных.

```
//—DLBL — имя файла, [,идентификатор набора данных,], [дата],
                                                [коды],
```

где «имя файла» является именем файла, заданного в операторе DECLARE;

«идентификатор набора данных» — идентификатор, который присваивается набору данных и проверяется на носителе (от 1 до 44 символов);

«дата» определяет срок хранения набора данных и может задаваться следующим образом:

— в виде числа, содержащего от 1 до 4 цифр, указывающих количество дней хранения набора;

— в виде абсолютной даты истечения срока хранения набора, содержащей последние две цифры года и номер дня в году (ГГ/ДДД);

«коды» — двух- или трехсимвольное поле, идентифицирующее тип организации набора данных:

SD — последовательная организация;

DA — региональная организация;

ISC — создание индексно-последовательного набора данных;

ISE — обработка индексно-последовательного набора данных.

Если параметр «коды» опущен, то принимается код SD.

EXTENT — запомнить информацию об участке набора данных на дисках. Этот оператор сообщает системе информацию об участке, занимаемом набором на дисках.

Участком называется одна или несколько дорожек, идущих друг за другом. Необходимость размещения набора данных на нескольких участках может обусловливаться тем, что пакет дисков занят несколькими наборами данных и остались свободными отдельные дорожки или несколько соприкасающихся друг с другом дорожек (участок). Один или несколько операторов EXTENT должны следовать непосредственно за оператором DLBL.

```
// — EXTENT [SYSxxx], [регистрационный номер],
[тип], [номер участка], [адрес участка],
[количество дорожек], [дорожка разделения цилиндра],
```

где SYSxxx — символическое имя логического устройства, которому назначен диск. Если параметр опущен, то берется имя, указанное в предшествующем операторе EXTENT;

регистрационный номер — регистрационный номер пакета дисков (тома), на котором находится описываемый участок (от 1 до 6 знаков). Если параметр опущен, то используется номер, указанный в предшествующем операторе EXTENT. Если этот параметр опущен во всех операторах EXTENT, то во время открытия набора данных не производится контроль за правильностью установки нужного тома;

«тип» — тип участка (записывается в виде 1, 2, 4 и 8):

1 — участок предназначен для размещения данных (неразделенные цилиндры);

2 — для области переполнения (для индексно-последовательных наборов);

4 — для размещения индексов цилиндра и главного индекса (для индексно-последовательных наборов);

8 — для данных (разделенные цилиндры).

Если параметр опущен, то принимается тип 1;

«номер участка» — десятичное число от 0 до 255, определяющее порядковый номер участка для набора данных, расположенного на нескольких участках. Указывается для каждого индексно-последовательного и регионального набора. Первый или единственный участок регионального или последовательного набора имеет номер 0. Номер можно опустить только в том случае, когда последовательный набор данных занимает один участок памяти на дисках.

Для участка, используемого для размещения главного индекса индексно-последовательного набора данных, должен быть указан номер 0. Если главный индекс не используется, то первый участок должен иметь номер 1. Если номер первого участка индексно-последовательного файла опущен, то оператор EXTENT не воспринимается;

«адрес участка» — от 1 до 5 цифр, определяющих относительный номер первой дорожки участка (относительно нулевой дорожки нулевого цилиндра). Если этот параметр опущен для индексно-последовательного на-

бора данных, то оператор EXTENT считается ошибочным. Для последовательных и региональных наборов с атрибутом INPUT этот параметр может быть опущен. Для подсчета адреса участка используется формула

$$10 * N + Nq,$$

где N — номер цилиндра; Nq — номер дорожки на цилиндре;

«количество дорожек» — количество дорожек, занимаемых участком (от 1 до 5 цифр). Для последовательного набора данных параметр можно опустить. Для наборов, отделяющих цилиндры от других наборов, количество дорожек представляет произведение количества цилиндров, отведенных набору данных, на количество дорожек, занимаемых этим набором на каждом из цилиндров;

«дорожка разделения цилиндра» — номер последней из дорожек, занимаемых участком на каждом из цилиндров в случае разделения цилиндров для последовательных наборов данных

TLBL — *запомнить информацию о метках набора данных на магнитной ленте*. Оператор сообщает информацию, которая необходима для создания или проверки меток наборов данных на магнитной ленте:

```
//—TLBL имя файла, [,идентификатор набора данных'],
[дата], [регистрационный номер набора],
[порядковый номер тома],
[порядковый номер набора],
[номер поколения], [номер версии поколения],
```

где «имя файла» идентично имени файла, объявленному в операторе DECLARE;

«идентификатор набора» — идентификатор, состоящий из 1—17 символов. В случае выводного набора данных он записывается в метке ленты (если он опущен, используется имя файла). Для вводных наборов он сравнивается с информацией в метке ленты;

«дата» аналогична оператору DLBL. Если параметр опущен, срок хранения принимается равным семи дням;

«регистрационный номер набора» — архивный номер первой катушки набора, расположенного на нескольких катушках (от 1 до 6 символов);

«порядковый номер тома» для многотомного набора (от 1 до 4 двоичных цифр). При переключении на следующую ленту этот номер автоматически увеличивается. Если параметр опущен для выводного набора данных, его значение принимается равным 0001. Если параметр опущен для вводного набора, контроль номера тома не производится;

«порядковый номер набора» задается для многофайлового тома (от 1 до 4 двоичных цифр). В случае выводных наборов он автоматически увеличивается при выполнении процедур OPEN/CLOSE. Если параметр опущен для выводного набора, его значение принимается равным 0001. Если он опущен для вводного, то контроль порядкового номера не производится;

«номер поколения» определяет дополнительную информацию к идентификатору набора данных (от 1 до 4 двоичных цифр). Если параметр опущен для выводного набора, его значение принимается равным 0001. Если опущен для вводного, контроль не производится;

LBLTYP — разервировать память для информации о метках. Оператор определяет объем основной памяти, которая должна быть отведена РЕДАКТОРОМ в области проблемной программы для обработки меток файла:

```
//—LBLTYP {NSD (nn)}
          {TAPE   }
```

где NSD (*nn*) задается, если должна быть зарезервирована одна или несколько областей меток для индексно-последовательного или регионального набора данных. Число *nn* задает максимальное количество участков, используемых для одного такого набора. Это число равно количеству операто-

ров EXTENT для набора. Для наборов INDEXED оно может не совпадать с числом, заданным в режиме EXTENTNUMBER;

TAPE задается в том случае, если обрабатываются один или несколько ленточных наборов данных с метками. При обработке индексно-последовательных или региональных наборов одновременно с ленточными наборами с метками нужно задавать только параметр NSD (*nn*). Область меток необходима только тогда, когда в фазе обрабатываются ленточные наборы данных с метками, индексно-последовательные и региональные наборы на дисках. Если такие наборы в программе не используются, оператор LBLTYP не нужен. Этот оператор должен непосредственно предшествовать оператору //—EXEC LNKEDT.

— *МТС — выполнить служебные операции на магнитной ленте.*

$$\text{//—МТС код команды } \begin{Bmatrix} \text{X'}\textit{ои} \\ \text{SYSxxx} \end{Bmatrix} [,nn],$$

где код команды определяет вид служебной операции, выполняемой на МЛ;

BSF — шаг на группу зон назад, лента устанавливается в положение для чтения ленточной марки, предшествующей пропущенному набору;

BSR — шаг на зону назад;

ERG — очистка зоны (очищается промежуток при движении ленты вперед);

ESE — шаг на группу зон вперед, лента устанавливается после ленточной марки, следующей за пропущенным набором;

ESR — шаг на зону вперед;

RUN — перемотка и разгрузка;

REW — перемотка;

WTM — записать ленточную марку;

X'*сии*' — адрес физического устройства в шестнадцатиричном коде;

SYSxxx — символическое имя логического устройства которому назначена МЛ;

nn — десятичное число, указывающее, сколько раз должно быть повторено выполнение служебной операции на МЛ Если параметр опущен, операция выполняется один раз.

ПРИЛОЖЕНИЕ 6

Коды и причины прерываний во время выполнения программы

Таблица 1

Ошибки организации

Код ошибки	Причина прерывания (появления ошибок)
21	Для DSA недостаточно области памяти
22	Метка, заданная в операторе GOTO, является недействительной или ошибочной
23	Процедура с режимом MAIN вызывается программой на ПЛ/1 (вызов главной процедуры)
24	Процедуре, которая требует в качестве параметра переменную с плавающей точкой длинного формата, передается значение с плавающей точкой короткого формата
25	В позиции, которая соответствует знаку T, I или P спецификации шаблона, находится недопустимый для данной позиции знак

Таблица 2

Ошибки при обращении к встроенным функциям

Код ошибок	Причина прерывания
30	Аргумент X встроенной функции SQRT(X) отрицателен
31	Абсолютное значение аргумента X встроенных функции SIN(X) или COS(X) при K=π или SIND(X) или COSD(X) при K=180° больше или равно (2 ** 18)*K
32	Абсолютное значение аргумента X встроенной функции TAN(X) при K=π или TAND(X) при K=180° больше или равно (2 ** 18)*K
33	Абсолютное значение встроенных функций TAN или TAND стремится к бесконечности
34	Аргументы (X, Y) встроенной функции ATAN(X, Y) равны нулю
35	Аргумент X встроенных функций SINH и COSH больше 174.6
36	Аргумент X встроенной функции EXP (X) больше 174.6
37	Аргумент X встроенной функции ATANH больше 1
38	Аргумент X встроенных функций LOG, LOG2, LOG10 меньше или равен нулю
39	В выражении (X ** Y) X=0, а Y≤0
3A	В выражении (X ** Y) X и Y равны нулю
40	Аргумент X встроенной функции SQRT отрицателен
41	Абсолютное значение аргумента X встроенной функции SIN или COS при K=π или SIND или COSD при K=180° больше или равно (2 ** 50)*K
42	Абсолютное значение аргумента X встроенной функции TAN при K=π или TAND при K=180° больше или равно (2 ** 50)*K
43	Абсолютное значение встроенной функции TAN или TAND стремится к бесконечности
44	Аргументы X, Y встроенной функции ATAN(X,Y) равны нулю
45	Аргумент X встроенной функции SINH или COSH больше 174.6
46	Аргумент X встроенной функции EXP больше 174.6
47	Аргумент X встроенной функции ATANH больше 1
48	Аргумент X встроенной функции LOG, LOG2 или LOG10 меньше или равен нулю
49	В выражении (X ** Y) X=0, а Y≤0
50	Аргумент Y функции MOD равен нулю (в случае двоичного аргумента)
51	Аргумент Y функции MOD равен нулю (в случае десятичного аргумента)
52	Для аргументов X, Y функции MOD выполняется следующее: а) X/Y меньше $5.4 * 10^{-79}$; б) X/Y больше $7.2 * 10^{75}$; в) Y=0 (в случае короткого аргумента с плавающей точкой)
53	Та же причина, что и при коде 52 (в случае длинного аргумента с плавающей точкой)

Продолжение табл. 2

Код ошибки	Причина прерывания
54	MOD(X,Y)$\geqslant$ABS(Y)*) (в случае короткого аргумента с плавающей точкой)
55	MOD(X,Y)$\geqslant$ABS(Y)*) (в случае длинного аргумента с плавающей точкой)

*) Функция MOD для аргументов с плавающей точкой вычисляется следующим образом: $a=X/Y$; $b=Y*a$; MOD (X, Y)$=X-b$; Если экспонента X так велика, что X+Y имеет такое же значение, что и X, то MOD (X, Y)$=0$. В этом случае выдается сообщение 54 и 55.

Таблица 3

Ошибки ввода — вывода

Код ошибки	Причина прерывания
61	Неразрешенная комбинация элементов списка данных и элементов списка форматов
62	Попытка читать или писать за пределами указанной строки в операторе GET или PUT с режимом STRING
63	Элементы формата PAGE, SKIP, LINE или COLUMN задаются для файла, не имеющего атрибут PRINT, или попытка пропустить больше трех строк при помощи SKIP
64	Попытка выполнить неправильный оператор GET или PUT для файла STREAM
65	Попытка выполнить неправильный оператор READ, WRITE, REWRITE или LOCATE для файла CONSECUTIVE BUFFERED
66	Попытка выполнить неправильный оператор READ, WRITE, REWRITE для файла CONSECUTIVE UNBUFFERED
67	Попытка выполнить неправильный оператор READ, WRITE или REWRITE для файла REGIONAL
69	Указан режим PAGESIZE для файла, не имеющего атрибут PRINT
6A	Попытка выполнить неправильный оператор READ, WRITE или REWRITE для файла INDEXED DIRECT
6B	Попытка выполнить неправильный оператор READ, WRITE или REWRITE для файла INDEXED SEQUENTIAL
6C	Попытка прочитать слишком длинный элемент данных с помощью оператора GET LIST
6D	В одно и то же время необходимо обработать ситуации TRANSMIT и (или) RECORD для более чем трех файлов

Продолжение табл. 3

Код ошибки	Причина прерывания
6E	Файл, для которого ситуация RECORD или TRANSMIT помечена в таблице ввода—вывода, отсутствует в стеке ошибочных файлов. Это сообщение появляется также, когда отсутствует оператор LBLTYR, вследствие чего происходит перекрытие таблицы ввода — вывода областью меток, при этом в таблице ввода—вывода может установиться соответствующий бит
6F	Попытка выполнить неправильный оператор GET LIST или PUT LIST для файла STREAM
70	Во время записи по ключу, который указан в режиме KEY оператора READ, произошла ошибка
71	Область индекса цилиндра недостаточна, чтобы разместить все записи, необходимые для того, чтобы каждый цилиндр, указанный для основной области данных, индексировался
72	Область главного индекса недостаточна, чтобы разместить все записи, необходимые для того, чтобы каждая дорожка области индекса цилиндра индексировалась
7B	Попытка читать или писать за пределами указанной строки операторами GET LIST или PUT LIST с режимом STRING
80	Запись, которая должна быть прочитана с помощью оператора READ KEY, не найдена в файле
81	Нет больше свободного места в области переполнения для записей, которые должны добавляться к файлу INDEXED DIRECT с помощью оператора WRITE KEYFROM
82	Основная область данных заполнена во время создания или пополнения файла INDEXED SEQUENTIAL с помощью оператора WRITE KEYFROM
83	Запись, которая оператором WRITE KEYFROM должна записываться в файл INDEXED SEQUENTIAL или DIRECT, имеет такой же ключ, как и запись, уже имеющаяся в файле
84	Ключ записи, которая записывается в файл INDEXED SEQUENTIAL, не возрастает по сравнению с ключом предыдущей записи
87	Эта ошибка относится к файлу STREAM INPUT, который обрабатывается оператором GET LIST и возникает в следующих случаях: разделитель элементов данных не пробел и не запятая знак B отсутствует во внешнем представлении строки битов внешнее представление элемента данных несовместимо с внутренним объявлением этого элемента, например, внешнее представление — строка символов, а внутреннее объявление — строка битов, внешнее — строковое данное, а внутреннее — арифметическое с фиксированной или плавающей точкой

Таблица емкостей дорожек магнитных дисков

Для определения количества записей на дорожке (при одинаковой длине записей) используется формула

$$N = 1 + \frac{3660 - B}{A},$$

где B — количество байтов в последней записи; A — количество байтов в записи

Они вычисляются по формулам:

$$B = 55 - C + (KL + DL),$$
$$A = 82 - C + 1{,}049\ (KL + DL),$$

где C = 0, если запись с ключом; KL — количество байтов в области ключей; DL — количество байтов в области данных; C = 20, если запись без ключа

Максимальная длина записи (KL + DL)		Количество записей на дорожке (N)	Максимальная длина записи (KL + DL)		Количество записей на дорожке (N)
без ключа	с ключом		без ключа	с ключом	
3625	3605	1	161	142	16
1738	1719	2	148	129	17
1130	1110	3	136	117	18
829	810	4	126	107	19
650	630	5	117	97	20
530	511	6	107	88	21
446	426	7	100	81	22
382	363	8	93	74	23
333	314	9	87	68	24
293	274	10	81	62	25
261	242	11	76	57	26
234	215	12	71	52	27
212	192	13	66	47	28
192	173	14	62	43	29
175	156	15	58	39	30

Пример. Найти количество записей с ключом на дорожке при длине записи EL = 10 байтов и DL = 80 байтов

$$KL + DL = 10 + 80 = 90.$$

Находим по таблице во 2-й колонке ближайшее значение KL + DL = 97. Следовательно, на дорожку можно записать 20 записей.

Список литературы

1. **Джеймейн К.** Программирование на IBM/360. М., «Мир», 1973.
2. **Универсальный** язык программирования. Под ред. В. М. Курочкина. М,. «Мир», 1968.
3. **Операционная** система ДОС/ЕС. ПЛ/1. Пособие по языку. Е10.132.070.Д1 и Е10.132.070.Д2.
4. **Операционная** система ДОС/ЕС.ПЛ/1. Руководство для программиста. Е10.132.071.Д1.
5. **ЕС ЭВМ.** Операционная система ДОС/ЕС. Общие положения. Е10.132.013.Д1.
6. **Schoch H.** Programmierung in PL/I. Leipzig, 1972.

Предметный указатель

Оглавление