БИБЛИОТЕЧКА ПРОГРАММИСТА

Г. Л. СЕМАШКО
А. И. САЛТЫКОВ

Программирование на языке паскаль

БИБЛИОТЕЧКА ПРОГРАММИСТА

Г.Л.СЕМАШКО
А.И.САЛТЫКОВ

ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ ПАСКАЛЬ

Издание второе,
переработанное и дополненное

Под редакцией
В.П.ШИРИКОВА

Москва «Наука»
Главная редакция
физико-математической литературы
1993

ББК 22.18
С30
УДК 519.6

Семашко Г.Л., Салтыков А.И. Программирование на языке Паскаль.—М.: Наука. Гл.ред.физ.-мат.лит., 1993.—208 с.— (Библиотечка программиста.)—ISBN 5-02-015060-6.

Дается описание широко распространенного языка Паскаль. Излагается в основном стандартный Паскаль. Учитываются особенности работы на компьютерах разных типов, более подробные сведения и указания приводятся для машин типа ЕС ЭВМ и БЭСМ-6 и персональных компьютеров типа IBM.

Предлагаемая книга не требует от читателя специальной подготовки и имеет целью дать практические навыки, достаточные для самостоятельного составления несложных программ и запуска их на компьютерах.

Для инженеров, научных работников, аспирантов, начинающих программировать на компьютерах, а также для студентов, приступающих к изучению языка Паскаль.

Ил. 79. Библиогр. 33 назв.

Научное издание

СЕМАШКО Галина Львовна, САЛТЫКОВ Альберт Иванович

ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ ПАСКАЛЬ

Серия «Библиотечка программиста», вып.66

Заведующий редакцией *Е.Ю.Ходан*
Редактор *Л.Г.Полякова*
Художественный редактор *Г.М.Коровина*
Технический редактор *Л.В.Лихачева*. Корректор *Т.С.Вайсберг*

ИБ № 41668

Сдано в набор 18.02.92. Подписано к печати 18.08.92. Формат 84 × 108/32. Бумага книжно-журнальная. Гарнитура литературная. Компьютерный набор. Печать офсетная. Усл.печ.л. 10,92. Усл.кр.-отт. 11,13. Уч.-изд.л. 10,13. Тираж 16540 экз. Заказ №487. С-081.

Издательско-производственное и книготорговое объединение «Наука»
Главная редакция физико-математической литературы
117071 Москва В-71, Ленинский проспект, 15

Отпечатано в Четвертой типографии ВО «Наука»
630077 Новосибирск-77, ул. Станиславского, 25

С $\frac{1404000000—081}{053(02)—93}$ КБ-18-9-92

ISBN 5-02-015060-6

ПРЕДИСЛОВИЕ КО ВТОРОМУ ИЗДАНИЮ

За три года, прошедшие со времени первого издания книги, получили широкое распространение персональные компьютеры. Они используются для самых разных целей, но имеют ряд общих черт. Во-первых, большая часть из них совместима с IBM PC, а во-вторых, большинство пользователей предпочитают работать в системе Турбо-Паскаль. Однако в настоящее время практически нет *простых* описаний этой системы, которые были бы адресованы массовому пользователю.

В связи с этим авторы посчитали необходимым при переиздании книги, ориентированной на широкий круг читателей, включить материал о системе Турбо-Паскаль. Эта информация необходима десяткам тысяч людей, начинающих работать на новой для себя технике либо с новой версией языка Паскаль, значительно отличающейся как от стандарта, так и от версии 3.0.

Включенный материал содержит необходимый минимум знаний по Турбо-Паскалю версии 5.0, не требующий от читателя особой квалификации. Прежде всего дается список файлов, необходимых для запуска системы на персональном компьютере. Затем подробно описывается сеанс работы, что позволит многим читателям самостоятельно начать освоение системы. Приводятся команды редактора, готового выполнить все пожелания пользователя: исправить текст, перенести часть текста с одного места на другое, найти и заменить какой-либо фрагмент по всему тексту и т. д. Описываются средства отладки программ, позволяющие, например, следить за значениями конкретных переменных, изменять "на ходу" их значения. Приводятся языковые отличия от стандарта. В частности, дается подробное описание работы с типами STRING и FILE.

Особую привлекательность Турбо-Паскаля составляет богатство графических средств, о которых читатель найдет в книге исчерпывающую информацию.

Большую ценность не только для начинающих, но и для профессионалов представляет приведенный полный список процедур и функций Турбо-Паскаля 5.0 с необходимыми пояснениями.

Дополнив книгу предлагаемым материалом, авторы сделали ее актуальной и для пользователей персонального компьютера.

Г. Л. Семашко
А. И. Салтыков

ПРЕДИСЛОВИЕ К ПЕРВОМУ ИЗДАНИЮ

Язык программирования Паскаль, созданный Н. Виртом на рубеже 60—70-х годов, получил в настоящее время широкое распространение во всем мире. Трансляторы с Паскаля имеются на отечественных ЭВМ многих типов, в том числе на БЭСМ-6 и ЕС ЭВМ.

Важной особенностью Паскаля, отличающей его от языков программирования, созданных ранее (Фортрана, Алгола и др.), является последовательное проведение в жизнь идей структурного программирования [12]. Другой существенной особенностью Паскаля является концепция структуры данных как одного из фундаментальных понятий, лежащих, наряду с понятием алгоритма, в основе программирования [9].

К настоящему времени на русском языке издано немало книг по Паскалю, в основном переводных [12, 16, 17, 19, 27, 33]. Среди многочисленных книг, изданных на английском языке, отметим [12], отличающуюся полнотой и точностью изложения материала.

Эти книги ориентированы в основном на достаточно опытного читателя. В них рассматриваются, как правило, вопросы, не связанные с реализацией языка Паскаль на ЭВМ конкретных типов. Между тем из опыта известно, что каждая конкретная реализация языка программирования содержит массу особенностей, представляющих трудности в первую очередь для начинающих пользователей ЭВМ.

При написании данной книги мы ставили себе задачу помочь пользователям машин типа БЭСМ-6 и ЕС ЭВМ, начинающим освоение языка Паскаль. С этой целью материал книги разбит на две главы. Первая глава содержит наиболее простые и часто используемые конструкции языка Паскаль, доступные пониманию *начинающего* читателя. Вторая глава содержит более сложный материал, адресованный достаточно *опытному* читателю.

Мы стремились к тому, чтобы читатель как можно раньше начал самостоятельно составлять программы. Поэтому во многих случаях материал излагается сначала на элементарном уровне (и не в полном объеме). И лишь после того, как читатель «встанет на ноги» и приобретет начальный опыт, ему сообщаются более полные сведения по тому или иному вопросу.

Такая методика изложения материала, а также разбиение книги на две главы («элементарную» и более сложную) уже применялись нами ранее. Наш опыт преподавания основ программирования в школах Дубны и в Дубненском филиале НИИЯФ МГУ убедил нас в правильности такого способа изложения материала.

В процессе работы над книгой нам пришлось экспериментально проверять «восприятие» трансляторами на БЭСМ-6 и ЕС ЭВМ тех или иных конструкций языка Паскаль. Иногда обнаруживались расхождения между описаниями и фактическим положением вещей.

Считаем необходимым обратить внимание читателей на следующее обстоятельство.

Обычно в книгах тексты программ на Паскале набираются строчными буквами с выделением ключевых слов полужирным

шрифтом. Здесь все тексты на языке Паскаль набраны прописными буквами, более привычными для пользователей ЭВМ.

Мы понимаем, что для достаточно полного и точного описания особенностей реализации языка Паскаль на ЭВМ конкретного типа требуется несколько лет предварительной работы с соответствующим транслятором.

Данную книгу можно рассматривать как один из первых опытов описания языка Паскаль для машин типа БЭСМ-6 и ЕС ЭВМ.

Появлению этой книги во многом способствовала Екатерина Ивановна Стечкина, которая много лет возглавляла редакцию, выпускавшую книги по информатике и вычислительной математике.

Выражаем искреннюю благодарность О. В. Благонравовой, Г. А. Косянину и Ю. К. Крюкову за помощь в подготовке примеров и задач, А. А. Корнейчуку и А. В. Гусеву за ценные советы и рекомендации, А. Н. Графовой за помощь в подготовке рукописи к изданию.

Мы будем признательны читателям, которые пришлют свои замечания и предложения.

Дубна, 1988 *г.* *Авторы*

ВВЕДЕНИЕ

Язык Паскаль был создан в конце 60-х годов Н. Виртом как специальный язык для обучения студентов, но вскоре получил распространение среди программистов. Сейчас Паскаль широко используется и для написания прикладных программ, и как язык системного программирования.

В частности, программное обеспечение многих мини- и микрокомпьютеров написано на языке Паскаль. Трансляторы с этого языка имеются на наиболее распространенных типах ЭВМ во всем мире.

Перечислим основные причины популярности языка Паскаль.

1. Простота языка позволяет быстро его освоить и создавать алгоритмически сложные программы.

2. Развитые средства представления структур данных обеспечивают удобство работы как с числовой, так и с символьной и битовой информацией.

3. Наличие специальных методик создания трансляторов с Паскаля упростило их разработку и способствовало широкому распространению языка.

4. Оптимизирующие свойства трансляторов с Паскаля позволяют создавать эффективные программы. Это послужило одной из причин использования Паскаля в качестве языка системного программирования.

5. В языке Паскаль реализованы идеи структурного программирования [9, 12], что делает программу наглядной и дает хорошие возможности для разработки и отладки.

В данной книге содержится информация для начинающих пользователей, собирающихся работать на языке Паскаль.

Описываемые возможности Паскаля реализованы на машинах типа БЭСМ-6 и ЕС ЭВМ за исключением случаев, оговоренных особо. Дополнительные возможности, предоставляемые трансляторами на конкретных ЭВМ, в книге не описаны. С ними можно ознакомиться в книгах [7, 14, 17]. Приведены только те режимы трансляции, которые имеются на ЭВМ перечисленных типов.

Учитывая, что многие читатели владеют языком Фортран, мы будем в ряде случаев обращать их внимание на конкретные сходства и различия языков Паскаль и Фортран.

ГЛАВА I

ПАСКАЛЬ ДЛЯ НАЧИНАЮЩИХ

1. Задания для ЭВМ с программой на языке Паскаль

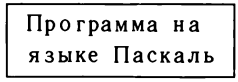
Работа ЭВМ осуществляется под управлением специального комплекса программ, называемого *операционной системой* (*ОС*). Машины каждого типа могут быть оснащены несколькими операционными системами. Например, для машин серии ЕС ЭВМ разработаны операционные системы ДОС ЕС и ОС ЕС, для ЭВМ БЭСМ-6 существуют системы Дубна, Диспак и др.

Чтобы программа могла быть выполнена какой-либо ЭВМ, необходимо составить *задание* для этой ЭВМ. В него, помимо программы, входят так называемые *управляющие карты*. Программа на языке Паскаль может быть выполнена на любой ЭВМ, имеющей транслятор с этого языка, но управляющие карты должны быть свои для ЭВМ каждого типа.

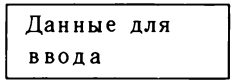
Управляющие карты содержат обязательную информацию о задании. Например, машине надо «знать» фамилию программиста, шифр (пароль), под которым выполняется задание, и некоторые другие сведения. Для каждой операционной системы имеется свой набор управляющих карт. Рассмотрим несколько примеров.

1.1. Задание для машин типа ЕС ЭВМ (вариант ОС ЕС). Управляющие карты (или карты управления заданием) для машин типа ЕС ЭВМ начинаются с символа /, располагаемого в первой колонке перфокарты (с первой позиции строки).

```
//XXXX JOB YYYY,'ФАМИЛИЯ',MSGLEVEL=(2,0)
// EXEC PASCLG
//PASC.SYSIN DD *
```




```
//GO.SYSIN DD *
```




```
//
```

Здесь через XXXX и YYYY обозначены соответственно имя задания и шифр пользователя.

П р и м е р. Пусть пользователь Петров, имеющий на одной из машин типа ЕС ЭВМ шифр PET12, собирается выполнить на этой ЭВМ задание под названием TASK1 (имя задания). Тогда первая карта задания будет такой:

```
//TASK1 JOB PET12,PETROV,MSGLEVEL=(2,0)
```

Мы не будем утомлять читателя подробным описанием всех карт управления заданием, поясним только вторую карту

```
// EXEC PASCLG
```

Здесь EXEC означает «выполнение» (EXECUTION), PAS — программа написана на языке Паскаль, C — требуется трансляция (COMPILATION), L — работа редактора связей (LINKAGE EDITOR), G — выход на счет (GO).

Если выход на счет не требуется, то эта карта выглядит так:

```
// EXEC PASCL
```

На некоторых ЕС ЭВМ принят другой набор карт управления заданием. Приведем пример:

```
//XXXX JOB YYYY,'ФАМИЛИЯ',MSGLEVEL=(2,0)
// EXEC PASCLG
//C.SYSIN DD *
```


```
//G.SYSIN DD *
```

```
Данные для
ввода
```

//

(Подробнее о картах управления заданием см. [6].)

1.2. Задание для ЭВМ БЭСМ-6 в системе Дубна (пакет задачи). На ЭВМ БЭСМ-6 управляющие карты начинаются с символа * (в первой колонке).

```
*NAME XXXX
*PASS:YYYY
*TIME:00.05
*LIBRARY:11
*CALL ALLMEMORY
*PASCAL
```

```
Программа на
языке Паскаль
```

```
*EXECUTE
```

```
Данные для
ввода
```

```
*END FILE
```

Здесь через XXXX и YYYY обозначены соответственно фамилия пользователя и его шифр. Если пакет задачи создан на перфокартах, то следует добавить еще одну перфокарту — так называемый «диспетчерский конец». Эта карта содержит все пробивки в 1-й и 41-й колонках, и притом только в этих колонках.

П р и м е р. Пусть задание первого примера должно быть выполнено на ЭВМ БЭСМ-6 и для его выполнения требуется не более двух минут времени. Первые три карты задания будут такими:

```
*NAME PETROV
*PASS:PET12
*TIME:00.02
```

Более подробно управляющие карты мы не описываем.

1.3. Задание в системе Диспак. Пример пакета имеет вид:

```
ШИФР 5 1 5 3 1 0 3 С 2
Е Е В 1 А 3
ВРЕМ 0 5 3 0 0 0
*NAME СЕМАШКО
*FICMEMORY
*PASCAL
*LIBRARY:11
```

Программа на языке Паскаль

```
*EXECUTE
```

Данные для ввода

```
«диспетчерский конец»
ЕКОНЕЦ
```

2. Словарь языка Паскаль

Язык Паскаль оперирует со следующим набором символов:

а) все латинские (и русские буквы на БЭСМ-6);

б) все арабские цифры;

в) ограничители и специальные символы:

+	:=	:	<	(	..
−	.	'	<=	)	{
*	,	=	>	[	}
/	;	< >	>=	]	↑

г) ключевые слова:

AND	END	NIL	SET
ARRAY	FILE	NOT	THEN
BEGIN	FOR	OF	TO
CASE	FUNCTION	OR	TYPE
CONST	GOTO	PACKED	UNTIL
DIV	IF	PROCEDURE	VAR
DO	IN	PROGRAM	WHILE
DOWNTO	LABEL	RECORD	WITH
ELSE	MOD	REPEAT	

В программе нельзя использовать идентификаторы, совпадающие по написанию с приведенными выше ключевыми словами.

Вместо символов { и } , отсутствующих на клавиатуре устройств подготовки данных, набираются соответственно пары символов (* и *).

Конструкция (* ТЕКСТ *) воспринимается как *комментарий* и может быть помещена в любом месте программы. На БЭСМ-6 нельзя ставить комментарий после END (последнего оператора программы).

П р и м е р 1. Пусть в каком-либо участке программы вычисляется корень уравнения. Тогда перед этим участком полезно поместить следующий комментарий:

(* ВЫЧИСЛЕНИЕ КОРНЯ УРАВНЕНИЯ *)

Пробелы, комментарии, концы строк являются *разделителями*. Между любыми именами, числами, ключевыми словами должен стоять по крайней мере один разделитель, а может их быть и сколько угодно. Но нельзя отделять один символ от другого внутри имени, числа либо ключевого слова.

П р и м е р 2. Ключевое слово GOTO нельзя записать как GO TO (в отличие от Фортрана, где это допускается).

Можно:	Нельзя:
GOTO 5005;	GOTO5005;
GOTO 5005;	GO TO 5005;
GOTO 5005;	GOTO 5 005;

На ЕС ЭВМ используется следующее представление символов:

Стандартное*)	[ ]	{ }	AND	OR	NOT	< >
Паскаль ЕС	(. .)	(* *)	&	!	¬	¬=
*) Используется в стандарте языка Паскаль.						

3. Данные

Программа, написанная на языке Паскаль (равно как и программа, написанная на любом из других языков программирования), предназначена для обработки *данных*. Эти данные могут быть различной природы (числа, тексты, последовательности двоичных разрядов, или битов, и т. п.). Одни данные являются *исходными* (или, как говорят, задаются на входе), другие являются *результатами*, полученными из исходных данных в процессе выполнения программы (про такие данные обычно говорят, что они получаются на выходе).

В зависимости от способа их хранения и обработки в ЭВМ данные можно разбить и на две другие группы: *константы* и *переменные*.

Константы — это те данные, значения которых не изменяются в процессе работы программы. Значения переменных, в отличие от констант, могут изменяться во время выполнения программы. Константы «узнаются» машиной по форме их записи, а переменные — по именам (или идентификаторам).

В языке Паскаль используются константы трех видов: *числовые*, *булевские* и *символьные*. Первые предназначены для представления числовых данных (целых и вещественных). Булевские константы используются для представления данных, имеющих смысл логических высказываний (да — нет, истина — ложь). Символьные константы представляют данные, являющиеся последовательностями символов (тексты).

3.1. Константы. Числовые константы могут быть целыми и вещественными.

3.1.1. Целые константы (INTEGER). Целая десятичная константа представляет собой последовательность десятичных цифр, которой может предшествовать знак –.

Пример 1. 158; –15; 237845

Целые константы в Паскале не должны превосходить по абсолютной величине

на ЭВМ БЭСМ-6 $2^{40} - 1 \approx 10^{12}$,
на ЕС ЭВМ $2^{31} - 1 \approx 2 \cdot 10^{9}$.

На БЭСМ-6 допустимы и целые восьмеричные константы, которые представляют собой последовательности восьмеричных цифр, оканчивающиеся справа буквой B.

Пример 2. 135B; 2B

Можно использовать и восьмеричные константы, оканчивающиеся вместо B буквами C либо T. Буква C означает, что последовательность восьмеричных цифр расположена в машинном слове *справа*, а слева дополняется нулями; буква T — последовательность восьмеричных цифр расположена в слове *слева*, а справа дополняется нулями.

Пример 3.

0025T есть константа 0025 0000 0000 0000,
146C есть константа 0000 0000 0000 0146.

3.1.2. Вещественные константы (REAL). Представление вещественных констант, как и на Фортране, имеет две формы: в виде десятичной дроби, где вместо запятой используется точка (например, число 3.2), и в виде числа, содержащего указание на степень десяти (например, число 2.5E9).

Число, в записи которого использована степень десяти (например, $2{,}5 \cdot 10^9$), изображается на Паскале так: знак умножения опускается, вместо основания 10 пишется буква E, следом за E — показатель степени (т. е. так же, как и на Фортране).

Пример 4.

2.5E9 соответствует числу $2{,}5 \cdot 10^9$
0.13E–10 соответствует числу $0{,}13 \cdot 10^{-10}$

Замечание. Константа, записанная в виде десятичной дроби, в отличие от констант Фортрана, обязательно должна содержать как целую часть, так и дробную, т. е. нельзя записать 2. и .5, а следует обозначать 2.0 и 0.5.

Вещественные константы не должны по абсолютной величине превосходить

на БЭСМ-6 10^{19},
на ЕС ЭВМ 10^{76}.

Целое число может быть записано в виде вещественной константы с нулевой дробной частью (например, число 3.0).

Если вещественная константа по модулю меньше некоторого определенного числа, то машиной она воспринимается как нуль («машинный нуль»).

Для ЭВМ каждого типа эта наименьшая вещественная константа своя. Если обозначить ее через MINR, то

для БЭСМ-6 $MINR = 10^{-19}$,
для ЕС ЭВМ $MINR = 10^{-78}$.

Пример 5. Для БЭСМ-6 результат вычисления выражения $(10^{-10})^2$ есть машинный нуль, а для ЕС ЭВМ — число 10^{-20}.

Таким образом, ЭВМ каждого типа оперирует с конечным набором чисел из определенного диапазона.

Для БЭСМ-6 диапазон изменения вещественных чисел по модулю — от 10^{-19} до 10^{19}, для ЕС ЭВМ — от 10^{-78} до 10^{76}.

3.1.3. Булевские константы (BOOLEAN). Имеются две булевские константы: TRUE и FALSE.

3.1.4. Символьные константы (CHAR). Символ, заключенный в апострофы, есть символьная константа.

Пример 6. 'A', '1', '='

3.1.5. Константы-строки. Последовательность символов, заключенная в апострофы, есть строка.

Пример 7. 'DUBNA'; '12345'; 'A*B'

Длиной строки K называется число символов в ней.

3.1.6. Константы типа ALFA*). Строку символов длиной в одно машинное слово называют константой типа ALFA.

Длина K этой константы зависит от типа ЭВМ:

для БЭСМ-6 K = 6,
для ЕС ЭВМ K = 8.

Пример 8. Константы типа ALFA

на БЭСМ-6 'DUBNA '; '141980'; 'MOCKBA';
на ЕС ЭВМ 'DUBNA '; 'IVANENKO'

---

*) Расширение стандарта языка Паскаль для ЭВМ БЭСМ-6 и ЕС.

Если среди символов константы-строки имеется апостроф, то он изображается двумя апострофами.

Пример 9. Константу A'B'C'D следует набрать как 'A''B''C''D'.

4. Идентификаторы

Идентификатор Паскаля — это последовательность букв или цифр, начинающаяся с буквы. Значащими являются первые восемь символов. Заметим, что в Паскале, в отличие от Фортрана, пробелы в идентификаторах не допускаются.

Пример. A; B12, I5D13; STACKCOMP

В последнем идентификаторе девять символов. Последний, девятый символ не учитывается транслятором, так как значащими являются первые восемь символов. Но для мнемоники программисту иногда бывает удобно использовать более длинные идентификаторы.

Вместо слова «идентификатор» часто употребляют синонимы «имя», «наименование», «название».

В программе нельзя использовать идентификаторы, совпадающие по написанию с приведенными в п. 2 ключевыми словами.

5. О типах переменных

Каждая переменная (например A, B, C), используемая в программе, должна быть описана следующим образом:

A:TYPE1; C,B:TYPE2; ...

Здесь A, B, C — идентификаторы переменных, TYPE1, TYPE2 — типы переменных.

Пример. J,K:INTEGER; A,B,C:REAL; L:BOOLEAN;

В языке Паскаль имеется следующий набор типов переменных:

ПРОСТЫЕ ТИПЫ
- СКАЛЯРНЫЕ
- НЕСТАНДАРТНЫЕ (ПЕРЕЧИСЛЕНИЕ)
- СТАНДАРТНЫЕ
  - ЦЕЛЫЙ (INTEGER)

ВЕЩЕСТВЕННЫЙ (REAL)
БУЛЕВСКИЙ (BOOLEAN)
СИМВОЛЬНЫЙ (CHAR)
СТРОКОВЫЙ (STRING)
ОГРАНИЧЕННЫЕ (SUBRANGE)
СЛОЖНЫЕ ТИПЫ
МАССИВ (ARRAY)
МНОЖЕСТВО (SET)
ФАЙЛ (FILE)
ЗАПИСЬ (RECORD)
ССЫЛКИ (УКАЗАТЕЛЬ, POINTER)

Этот же список более наглядно изображен в виде схемы. Набор типов языка Паскаль представлен на рис. 1.

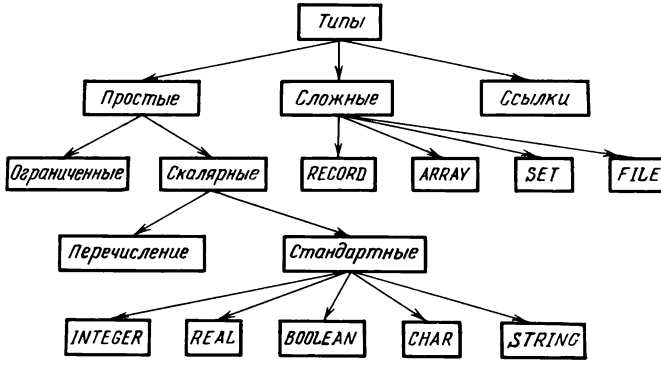


Рис. 1

Тип переменной определяет множество значений этой переменной, набор операций, которые к ней могут быть применены, а также тип результата выполнения этих операций.

Остановимся подробнее на типах, приведенных на схеме.

6. Скалярные типы

Каждый скалярный тип определяет соответствующее ему *упорядоченное* множество значений.

6.1. Тип целый (INTEGER). Переменные типа INTEGER могут принимать только *целые* значения. Такие переменные описываются следующим образом:

A,B,C : INTEGER;

Здесь A, B, C, ... — имена переменных, INTEGER — тип переменных. Транслятор, встретив такое описание переменных A, B, C, ..., запоминает, что эти переменные могут принимать только целые значения и формирует соответственно этому команды программы.

Если используются операнды *целого* типа, то следующие операции дают результат *целого* типа:

* (умножение),
DIV (деление без округления — целая часть частного),
MOD (остаток от деления A MOD B = A — ((A DIV B)*B)),
\+ (сложение),
– (вычитание).

Функции

ABS(X) — абсолютная величина X,
SQR(X) — квадрат X.

Следующие функции дают *целый* результат и для X *вещественного*:

TRUNC(X) — (отбрасывание десятичных знаков после точки);
ROUND(X) — (округление до целого).

Операция (OP) над операндами целого типа выполняется правильно только при условии, что результат и каждый операнд по модулю не превосходят некоторой константы MAXINT:

ABS (A OP B) ≤ MAXINT,
ABS (A) ≤ MAXINT, ABS(B) ≤ MAXINT.

Для БЭСМ-6 $MAXINT = 2^{40} - 1 \approx 10^{12}$,
для ЕС ЭВМ $MAXINT = 2^{31} - 1 \approx 2 \cdot 10^{9}$.

Поясним на примерах работу приведенных операций и функций.

Пример 1. Пусть A = 14; B = 4. Тогда A DIV B дает 3; A MOD B дает 2 (остаток от деления); SQR(B) дает 16.

П р и м е р 2. Пусть X = 8.915. Тогда TRUNC(X) дает 8; ROUND(X) дает 9.

П р и м е р 3. Пусть надо на ЕС ЭВМ вычислить значение выражения $5 \cdot 10^5 \cdot 3 \cdot 10^5 \cdot 8 \cdot 10^{-5}$. Если запрограммировать так: 500000*300000*80E–5, то первое умножение не выполнится, так как результат $15 \cdot 10^{10}$ превышает MAXINT. Надо изменить порядок сомножителей, чтобы ЭВМ вычислила окончательный результат:

500000*80E–5*300000

П р и м е р 4. Выражение 5*3 дает результат типа «целый», а 5*3.0 — типа «вещественный», так как один из сомножителей вещественный.

6.2. Тип вещественный (REAL). Величины X этого типа могут принимать только вещественные значения.

Если хотя бы один из операндов вещественный, следующие операции дают вещественный результат: +, –, *, /.

Операция деления / дает *вещественный* результат и в случае *двух целых* операндов (в отличие от Фортрана).

П р и м е р. 6/2 дает 3.0

Следует обратить внимание, что стандартная функция ABS(X) — (модуль X) — от целого аргумента дает целый результат, а от вещественного — вещественный, как и SQR(X) — квадрат X; но функции

SIN (X) — синус X (X в радианах),
COS (X) — косинус X (X в радианах),
LN (X) — натуральный логарифм X,
EXP (X) — экспонента X,
SQRT (X) — корень квадратный из X,
ARCTAN (X) — арктангенс X

дают вещественный результат как для вещественного, так и для целого аргумента.

Нельзя использовать переменные и константы типа REAL:

а) в функциях PRED(X), SUCC(X), ORD(X);

б) в качестве индексов массивов;

в) в операторах передачи управления в качестве меток;

г) в определении базы типа SET (см. п. 20).

6.3. Тип булевский (BOOLEAN). Переменная булевского типа принимает значения TRUE или FALSE. Эти величины упорядочены следующим образом:

FALSE<TRUE

Операции AND, OR, NOT (применяемые к булевским операндам) дают булевские значения.

Операция AND (логическое умножение, пересечение, операция «И»). Выражение A AND B дает значение TRUE (истина) только в том случае, если и A, и B имеют значения TRUE. Во всех остальных случаях значение выражения A AND B — FALSE (ложь):

⟨TRUE⟩ AND ⟨TRUE⟩ = ⟨TRUE⟩,
⟨TRUE⟩ AND ⟨FALSE⟩ = ⟨FALSE⟩,
⟨FALSE⟩ AND ⟨FALSE⟩ = ⟨FALSE⟩.

Операция OR (логическое сложение, объединение, операция «ИЛИ»). Выражение A OR B дает значение FALSE в том и только в том случае, если и A, и B имеют значения FALSE. Во всех остальных случаях результат — TRUE:

⟨TRUE⟩ OR ⟨TRUE⟩ = ⟨TRUE⟩,
⟨TRUE⟩ OR ⟨FALSE⟩ = ⟨TRUE⟩,
⟨FALSE⟩ OR ⟨FALSE⟩ = ⟨FALSE⟩.

Операция NOT (отрицание, операция «НЕ»). Выражение NOT A имеет значение, противоположное значению A:

NOT⟨TRUE⟩ = ⟨FALSE⟩,
NOT⟨FALSE⟩ = ⟨TRUE⟩.

Стандартными булевскими функциями являются ODD(X), EOLN(X), EOF(X):

ODD(X) = TRUE, если X нечетный (X целый);

EOLN(X) = TRUE, если встретился конец строки текстового файла X;

EOF(X) = TRUE, если встретился конец файла X.

В остальных случаях эти функции принимают значение FALSE.

6.4. Тип символьный (CHAR). Переменная типа CHAR может принимать значения из определенной упорядоченной последовательности символов, разрешенной транслятором с Паскаля на данной ЭВМ.

Две стандартные функции позволяют поставить в соответствие данную последовательность символов множеству целых неотрицательных чисел (порядковым номерам символов последовательности).

Эти функции называются функциями преобразования:

ORD(C) выдает номер символа C (нумерация с нуля),
CHR(I) выдает I–й символ последовательности.

П р и м е р. ORD(H) выдает номер символа H в последовательности всех символов, используемых транслятором; CHR(15) выдает 15-й символ этой последовательности, например букву O.

6.5. Тип ALFA*). Переменная этого типа представляет собой машинное слово, содержащее символьную информацию: слово БЭСМ-6 содержит 6 символов, слово ЕС ЭВМ содержит 8 символов.

Переменная типа ALFA должна занимать *одно полное машинное слово,* т. е. содержать *ровно столько* символов, сколько их содержится в машинном слове.

К переменным этого типа можно применять операции отношения: равно (=), не равно (< >), меньше (<), больше (>), меньше или равно (< =), больше или равно (> =).

П р и м е р. Пусть A — переменная типа ALFA, содержащая слово 'IVANOV', B содержит слово 'PETROV', а F — слово 'ROGOV ' (на БЭСМ-6). Тогда

выражение A = B принимает значение FALSE,
выражение A < B (буква I предшествует P) — TRUE,
выражение B >= F (буква P предшествует R) — FALSE.

6.6. Тип «перечисление». В программу можно ввести и переменные какого-либо типа, не совпадающего ни с одним из

---

*) В стандарте языка Паскаль этот тип отсутствует.

стандартных. Такой тип задается перечислением значений, которые может принимать переменная.

Общий вид описания нестандартного типа:

TYPE NM=(WORD1, WORD2, ..., WORDN);

Здесь NM — идентификатор типа (произвольный идентификатор), WORD1, WORD2, ... — конкретные значения, которые может принимать переменная типа NM. Эти значения считаются упорядоченными, т. е. описание типа одновременно вводит упорядочение

WORD1<WORD2<...<WORDN

Пример 1. Пусть тип COLOR описан как

TYPE COLOR=(RED, YELLOW, GREEN, BLUE);

Здесь определено, что RED<YELLOW<GREEN<BLUE. Переменная типа COLOR может принимать одно из перечисленных значений.

Ко всем переменным скалярного типа, кроме REAL, применимы следующие стандартные функции: SUCC(X), PRED(X), ORD(X).

Функция SUCC(X). По элементу X определяется та упорядоченная последовательность, которой принадлежит X, и выдается элемент, следующий за X в этой последовательности.

Пример 2. Пусть задана последовательность букв в алфавитном порядке. Тогда SUCC(A) есть B; SUCC(L) есть M и т. д. В примере 1 SUCC(RED) есть YELLOW.

Функция PRED(X). По элементу X определяется последовательность, которой принадлежит X, и выдается предыдущий элемент этой последовательности.

Пример 3. PRED(F) есть E; PRED(Z) есть Y и т. д.

Функция ORD(X). Выдается номер элемента X из соответствующей последовательности.

Пример 4. Если заданная последовательность есть латинский алфавит, то ORD('A') есть 0; ORD('C') есть 2. (Нумерация начинается с нуля.)

Ко всем переменным одного и того же скалярного типа применимы операции отношения =, < >, <=, >=, <, >.

7. Ограниченные типы (SUBRANGE)

Для переменной скалярного типа можно указать некоторое подмножество значений, которые может принимать данная переменная.

Общий вид:

A:MIN..MAX;

Здесь A — переменная, MIN — левая граница, MAX — правая граница подмножества (диапазона). Границы диапазона разделяются двумя точками.

Тип MIN и MAX задает множество, определяющее основной тип переменной A (базовый тип). О переменной, описанной таким образом, говорят, что она имеет тип «ограниченный».

П р и м е р 1. Пусть переменная K может принимать значения из множества 1 ÷ 20. Тогда ей приписывают ограниченный тип

(SUBRANGE):K:1..20;

Основным типом переменной K является тип INTEGER, так как границами диапазона являются целые константы 1 и 20.

Если переменная B может принимать одно из значений RED, YELLOW, GREEN, то эту переменную можно описать так:

B:RED..GREEN;

Основным типом B является тип COLOR, описанный выше в примере 1. Граница MIN всегда должна быть меньше MAX.

П р и м е р 2. Пусть I — переменная, принимающая значения года рождения сотрудника какого-либо учреждения. Очевидно, имеет смысл ограничить диапазон значений I подмножеством по крайней мере 1900 ÷ 1980, т. е. описать так: I:1900..1980; переменная I будет иметь тип ограниченный, а не целый.

8. Структура программы

Программа состоит из *заголовка* и *блока*. Проиллюстрируем структуру программы на следующем примере:

```
PROGRAM MA (INPUT, OUTPUT, F1, F2);
LABEL 15,120;
```

```
CONST INMAX=81;
      PI=3.14;
      OUTMAX=60;
      EOL='!';
TYPE FAMILY=(FATHER,MOTHER,CHILD);
    LLINE=1..INMAX;
        AR=ARRAY[1..16] OF CHAR;
VAR LINE:ARRAY[LLINE] OF CHAR;
      L:LLINE;
      I,K,POS:INTEGER;
      SP:AR; F:FAMILY;
      C,D:REAL;
PROCEDURE RLINE;
VAR M:LLINE; B:AR; Z:FAMILY;
    BEGIN
      M:=1;
      WHILE NOT EOLN (INPUT) DO;
       BEGIN READ (LINE[M]); M:=M+1
       END;
       READLN;
       LINE[M]:=EOL;POS:=1;
END; (*  КОНЕЦ ПРОЦЕДУРЫ  *)
BEGIN (*  НАЧАЛО РАБОТЫ ПРОГРАММЫ  *)
15:LINE[1]:=EOL;
120:RLINE;
. . .
END.
```

8.1. Заголовок подпрограммы. В заголовке указывается имя программы и список параметров.

Общий вид:

```
PROGRAM N (INPUT,OUTPUT,X,Y,...);
```

Здесь N — имя программы; INPUT — файл ввода (указывается, если в программе есть ввод данных); OUTPUT — файл вывода (указывается всегда); X, Y — внешние файлы, используемые в программе (см. п. 21.1).

В приведенном выше примере заголовком является строка

```
PROGRAM MA (INPUT,OUTPUT,F1,F2);
```

8.2. Блок. Блок программы состоит из шести разделов, следующих в строго определенном порядке:

1) раздел меток (LABEL),
2) раздел констант (CONST),
3) раздел типов (TYPE),
4) раздел переменных (VAR),
5) раздел процедур и функций,
6) раздел действий (операторов).

Раздел действий должен присутствовать *всегда*, остальные разделы могут отсутствовать.

Каждый из первых четырех разделов начинается с соответствующего ключевого слова (LABEL, CONST, TYPE, VAR), которое записывается один раз в начале раздела и отделяется от последующей информации только пробелом (либо концом строки, либо комментарием).

В приведенном выше примере в блок входят строки от LABEL 15,120; до END.

8.2.1. Раздел меток (LABEL). Любой выполняемый оператор может быть снабжен меткой — целой положительной константой, содержащей не более четырех цифр. Все метки, встречающиеся в программе, должны быть описаны в разделе LABEL.

Общий вид:

LABEL L1, L2, L3, ...;

Здесь L1, L2, L3, ... — метки.

П р и м е р 1. LABEL 5,10,100;

Метка отделяется от оператора двоеточием.

П р и м е р 2. Пусть оператор A:=B; имеет метку 20. Тогда этот оператор выглядит так:

20: A:=B;

В приведенной выше программе в разделе LABEL описаны две метки, 15 и 120, используемые в программе.

8.2.2. Раздел констант (CONST). Если в программе используются константы, имеющие достаточно громоздкую запись (например, число π с восемью знаками), либо сменные константы (например, для задания варианта программы), то такие константы обычно обозначаются какими-либо именами и описы-

ваются в разделе CONST, а в программе используются только имена констант. Это делает программу более наглядной и удобной при отладке и внесении изменений.

Общий вид:

CONST A1=C1; A2=C2; ...

Здесь A1 — имя константы, C1 — значение константы.

Пример 3. CONST PI=3.14; C=2.7531;

В разделе CONST приведенной выше программы вводятся четыре константы, обозначаемые соответственно именами INMAX, PI, OUTMAX и EOL.

8.2.3. Раздел типов (TYPE). Если в программе вводится тип, отличный от стандартного, то этот тип описывается в разделе TYPE:

TYPE T1 = ⟨вид типа⟩;
T2 = ⟨вид типа⟩;
. . .

где T1 и T2 — идентификаторы вводимых типов.

Пример 4. TYPE COLOR=(RED,YELLOW,GREEN,BLUE);

Здесь описан тип COLOR, задаваемый перечислением значений.

В приведенной выше программе вводятся типы FAMILY, LLINE и AR.

8.2.4. Раздел переменных (VAR). Пусть в программе встречаются переменные V11, V12,...; все они должны быть описаны следующим образом:

VAR V11,V12,...:TYPE1;
V21,V22,...:TYPE2;...

Здесь V11, V12, ... — имена переменных; TYPE1 — тип переменных V11, V12, ...; TYPE2 — тип переменных V21, V22,...

Пример 5. VAR K,I,J:INTEGER; A,B:REAL;

Каждая переменная должна быть описана *до* ее использования в программе и отнесена к одному и только одному типу. Названия разделов (CONST, TYPE, VAR,...) указываются только *один раз*.

Пример 6. VAR A:REAL; B:REAL;

Таким образом, в разделе VAR вводится имя каждой переменной и указывается, к какому типу эта переменная принадлежит. Тип переменной можно задать двумя способами: указать имя типа (например, REAL, COLOR и т. д.) либо описать сам тип, например,

ARRAY[1..16] OF CHAR;

Рассмотрим приведенную выше программу (п. 8). В разделе VAR описаны переменные с именами:

LINE, L, I, K, SP, C, D, F

Для переменных L, SP, I, K, C, D, F указаны имена соответствующих типов (LLINE, AR, INTEGER, REAL, FAMILY). Часть из этих имен — стандартные (INTEGER, REAL), а типы LLINE, FAMILY и AR не являются стандартными. Эти типы должны быть описаны в разделе TYPE.

Тип переменной LINE никаким именем не назван и описан одновременно с описанием переменной.

Точно так же можно было бы поступить и для переменных L, SP, F:

L:1..INMAX;
SP:ARRAY[1..16] OF CHAR;
F:(FATHER, MOTHER, CHILD);

В этом случае в разделе TYPE эти типы не описываются. Но переменные M, B и Z (процедуры RLINE) имеют такие же типы, поэтому повторять громоздкие описания нерационально. Есть еще более существенная причина для описания типов LLINE, AR и FAMILY в разделе TYPE (см. п. 14.4), т. е. таких типов, к которым относятся как переменные PROGRAM, так и переменные процедуры.

О разделе процедур и функций речь пойдет в пп. 13 и 14.

8.2.5. Раздел действий (операторов). Эта часть программы начинается с ключевого слова BEGIN и заканчивается словом END, после которого должна стоять точка (END.). Раздел действий есть выполняемая часть программы, состоящая из операторов.

9. Операторы

Под операторами в языке Паскаль подразумевают (в отличие от Фортрана) только описание действий. Операторы отделяются друг от друга точкой с запятой. Если оператор стоит перед END, UNTIL или ELSE, то в этом случае точка с запятой не ставится.

9.1. Оператор присваивания. Общий вид:

V:=A;

Здесь V — переменная, A — выражение, := — операция присваивания. Выражение A может содержать константы, переменные, названия функций, знаки операций и скобки.

П р и м е р. F:=3*C+2*SIN(X);

Вид выражения однозначно определяет правила его вычисления: действия выполняются слева направо с соблюдением следующего старшинства (в порядке убывания):

1) NOT;
2) *, /, DIV, MOD, AND;
3) +, −, OR;
4) =, < >, <, >, <=, >=, IN.

Любое выражение в скобках вычисляется раньше, чем выполняется операция, предшествующая скобкам.

Присваивание допускается для переменных всех типов, за исключением типа файл. В операторе V := A; переменная V и выражение A должны иметь один и тот же тип, а для типа SUBRANGE — одно и то же подмножество значений.

З а м е ч а н и е 1. Разрешается присваивать переменной типа REAL выражение типа INTEGER.

З а м е ч а н и е 2. В отличие от Фортрана нельзя присваивать переменной типа INTEGER выражение типа REAL.

9.2. Вывод информации. Общий вид оператора:

WRITELN(P_1, P_2, ..., $P_п$);

Здесь WRITELN — имя процедуры вывода; P_1, P_2, ..., $P_п$ — список выражений, значения которых выводятся.

Кроме значений выражений, можно выводить и произвольный набор символов. Для этого набор символов заключают в апострофы:

'<набор символов>'

Пример 1. WRITELN ('P=',P);

Этот оператор выполняется так: сначала выводятся символы, заключенные в апострофы: P=. Затем выводится значение переменной P, например 13.5. На экране (на листинге) в результате работы оператора появится P=13.5...

Пример 2. Вычислить длину окружности радиуса 5,782:

```
PROGRAM T10(OUTPUT);
CONST R=5.782;
VAR L:REAL;
 BEGIN
  L:=2*3.1416*R;
  WRITELN(' L=',L)
 END.
```

либо

```
PROGRAM T11(OUTPUT);
VAR R:REAL;
 BEGIN R:=5.782;
  WRITELN(' L=',2*3.1416*R)
 END.
```

Для оператора WRITELN не требуется оператор типа фортранного FORMAT, так как тип выражения (2*3.1416*R) либо тип переменной (L) определяют некоторую спецификацию, выбранную по умолчанию для переменных данного типа (REAL).

Первый символ строки на печать не выводится, а служит для управления устройством печати: если это 1, то устройство начнет печатать с начала новой страницы, если «пробел» — со следующей строки, если «+» — без перехода к новой строке, т. е. с наложением строк.

Если оператор вывода на печать составлен без учета роли первого символа в строке, то могут быть непредвиденные режимы печати: пропуск страниц, строк, наложение строк.

З а м е ч а н и е. Фортранная спецификация nX отсутствует в языке Паскаль.

9.3. Примеры заданий для ЭВМ БЭСМ-6 и ЕС ЭВМ. Пусть программист Петров, имеющий шифр PET12, собирается выполнить программу примера 2 на ЭВМ БЭСМ-6 (ОС Дубна). Тогда ему надо составить следующее задание:

```
*NAME PETROV
*PASS:PET12
*TIME:00.05
*LIBRARY:11
*CALL ALLMEMORY
*PASCAL
  PROGRAM T13 (OUTPUT);
  VAR L:REAL;
   BEGIN
    L:=2*3.1416*5.782;
    WRITELN(' L=',L)
   END.
*EXECUTE
*END FILE
```

«Диспетчерский конец» (для колоды перфокарт)

Если же Петров назовет задание TEST1 и будет выполнять программу на ЕС ЭВМ, то задание будет выглядеть так:

```
//TEST1 JOB PET12, PETROV, MSGLEVEL=(2,0)
// EXEC PASCLG
//PASC.SYSIN DD *
PROGRAM T13(OUTPUT);
VAR L:REAL;
 BEGIN
  L:=2*3.1416*5.782;
  WRITELN(' L=',L)
 END.
//GO.SYSIN DD *
//
```

9.4. Оператор безусловного перехода GOTO. Общий вид:

```
GOTO N;
```

метка N, на которую передается управление, должна быть описана в разделе LABEL.

П р и м е р.

```
PROGRAM T (OUTPUT);
LABEL 7;
VAR A,B:REAL;
 BEGIN
   . . .
   GOTO 7
   . . .
 7:A:=B*3;
   . . .
 END.
```

9.5. Составной оператор. Если при некотором условии надо выполнить определенную последовательность операторов, то их объединяют в один составной оператор.

Составной оператор начинается ключевым словом BEGIN и заканчивается словом END. Между этими словами помещаются составляющие операторы, которые выполняются в порядке их следования. После END ставится точка с запятой, а после BEGIN — только пробелы (либо комментарий).

П р и м е р.

```
BEGIN
  I:=2;
  K:=I/5
END;
```

Слова BEGIN и END играют роль операторных скобок. Тело самой программы также имеет вид составного оператора. После последнего END программы ставится точка (END.). Нельзя извне составного оператора передавать управление внутрь его.

9.6. Оператор условного перехода. Этот оператор имеет две разновидности: IF и CASE.

9.6.1. Оператор IF. Булевские (логические) выражения могут принимать одно из двух значений: TRUE (истина) либо FALSE (ложь).

Простейшими логическими выражениями являются выражения отношения:

A1 OP A2

Здесь A1 и A2 — выражения, а OP — операция отношения. Операции отношения в языке Паскаль обозначаются так:

= — равно;
> — больше;
< — меньше;
>= — больше или равно;
<= — меньше или равно;
< > — не равно.

Пример 1. 3 < 5; 18 >= 2; 5 <= 6; A = B;

Общий вид оператора IF:

IF A THEN ST;

Здесь A — булевское выражение, ST — оператор (простой либо составной).

Если A — «истина», выполняется оператор ST. Если A — «ложь», то управление сразу передается следующему за ST оператору.

Пример 2. IF A < >0 THEN B:=X/A;

Если A ≠ 0, то выполняется оператор B:=X/A; если A = 0, то этот оператор пропускается и управление передается дальше, к следующему оператору.

Оператор IF может иметь и такой вид:

IF A THEN ST1 ELSE ST2;

Здесь A — булевское выражение, ST1, ST2 — операторы.

Если A — «истина», выполняется оператор ST1; если A — «ложь», выполняется оператор ST2, затем в обоих случаях управление передается к следующему оператору.

Замечание 1. Перед ELSE нельзя ставить точку с запятой.

Пример 3. IF A < >0 THEN B:=1/A ELSE B:=0;

Если A ≠ 0, то переменной B присваивается значение 1/A; если A = 0, то значение 0.

З а м е ч а н и е 2. Синтаксическая неоднозначность оператора

```
IF A1 THEN IF A2 THEN ST1 ELSE ST2;
```

трактуется так:

```
IF A1 THEN
  BEGIN IF A2 THEN ST1 ELSE ST2 END;
```

П р и м е р 4.

```
IF A < > 0 THEN IF B < > 0
THEN C:=A/B ELSE C:=0;
K:=-1;
```

Если A ≠ 0, то для B ≠ 0 C = A/B, а для B = 0 C = 0, затем выполняется оператор K := –1. Если A = 0, то управление сразу передается на K := –1.

9.6.2. Оператор CASE. Общий вид:

```
CASE N OF
  M1,...,MN:ST1;
  K1,...,KN:ST2;
  . . .
END;
A:=B;
```

Здесь N — переключатель (селектор), MI, KI — метки (I = 1, 2, ...), которые отличаются по смыслу от меток, описываемых в разделе LABEL. Переключатель и метки должны быть одного и того же скалярного типа, кроме REAL.

Оператор CASE передает управление тому оператору STI, с одной из меток которого совпало значение переключателя N, а затем — на следующий за END оператор.

П р и м е р 5.

```
CASE I OF
    2:X:=0;
    3:X:=X*X;
  100:X:=SIN(X)
END;
    A:=B;
```

Если I = 3, то выполняется оператор X := X * X; а затем управление передается на оператор A := B;

З а м е ч а н и е. Метки оператора CASE не описываются в разделе LABEL и на них нельзя переходить оператором GOTO.

9.7. Операторы цикла. В языке Паскаль имеется три вида операторов цикла:

WHILE, REPEAT и FOR

9.7.1. Оператор цикла WHILE. Общий вид:

WHILE A DO ST

Здесь A — булевское (логическое) выражение, ST — оператор (простой либо составной).

Выражение A вычисляется перед каждым выполнением оператора ST. Если A — «истина», то ST выполняется и управление передается на вычисление A; если A — «ложь», то ST не выполняется и происходит выход из цикла. Если первоначальное значение A — FALSE, то ST не будет выполнен ни разу.

П р и м е р 1.

```
WHILE X < >0 DO
 BEGIN C:=C+1/X;
  X:=X-1
 END;
```

В этом примере вычисляется выражение X < > 0. Если оно — «истина» (X ≠ 0), будут выполняться операторы

C := C + 1/X; X := X - 1;

и управление опять будет передано на вычисление выражения X < > 0.

Как только условие X ≠ 0 не выполнится, управление сразу передастся оператору, следующему за END;

З а д а ч а 1.

Вычислить сумму S = 1 + 1/2 + 1/3 + ... + 1/50:

```
PROGRAM N1(OUTPUT);
VAR S:REAL; N:INTEGER;
BEGIN
```

```
    S:=0; N:=1;
  WHILE N<=50 DO
    BEGIN S:=S+1/N;
     N:=N+1
    END;
  WRITELN(' S=',S)
  END.
```

Результат: S = 4.499...

9.7.2. Оператор цикла REPEAT. Общий вид:

```
REPEAT STS UNTIL A;
```

Здесь STS — группа выполняемых операторов, A — булевское выражение.

Работает оператор так: выполняются операторы STS, вычисляется выражение A; если оно — «ложь», то вновь выполняются операторы STS, если A — «истина», цикл заканчивается. Если A — «истина» с самого начала, то операторы STS выполняются один раз. Если A никогда не принимает значение «истина», то группа операторов STS выполняется бесконечное число раз, происходит «зацикливание».

П р и м е р 2. Рассмотрим такой цикл:

```
REPEAT C:=C+1/X; X:=X-1
UNTIL X = 0;
```

Сначала выполнятся операторы C := C + 1/X; X := X – 1, затем проверяется условие X = 0. Если X ≠ 0 (т.е. «ложь»), то повторяется выполнение указанных операторов; если X = 0 («истина»), то управление передается на оператор, следующий за строкой UNTIL X = 0;

З а д а ч а 2. Решить задачу 1 с использованием оператора REPEAT:

```
PROGRAM N2(OUTPUT);
VAR S:REAL; N:INTEGER;
 BEGIN
  S:=0; N:=1;
  REPEAT S:=S+1/N; N:=N+1;
  UNTIL N > 50;
```

```
  WRITELN(' S=',S)
 END.
```

Результат S=4.499...

9.7.3. Оператор цикла FOR. Общий вид:

```
FOR I:=N1 TO N2 DO ST;
```

Здесь I — переменная цикла, N1 — начальное значение переменной цикла, N2 — конечное значение, ST — оператор (простой либо составной).

Переменные I, N1 и N2 должны быть одного и того же скалярного типа, но не REAL. I принимает последовательные значения данного типа от N1 до N2. Если N1 и N2 — целые числа, а I — целая переменная, то шаг по I всегда равен единице.

П р и м е р 3. FOR I:=1 TO 20 DO A:=A+1;

Для I = 1, 2, 3, ..., 20 будет выполняться оператор

```
A := A + 1;
```

Если N1 и N2 символьного типа — например, имеют значения A и Z соответственно, то переменная I принимает последовательные значения в порядке алфавита: A, B, C, ..., Z.

Если N1 и N2 типа COLOR (тип COLOR=(RED,YELLOW,GREEN, BLUE)), например RED и GREEN соответственно, то переменная I принимает значения RED, YELLOW, GREEN.

Цикл по убывающим значениям параметра I от N2 до N1 имеет вид:

```
FOR I:=N2 DOWNTO N1 DO ST;
```

В этом случае параметр I принимает последовательные убывающие значения данного типа от N2 до N1.

П р и м е р 4. FOR I:=20 DOWNTO 1 DO A:=A+1;
Переменная I изменяется от 20 до 1 с шагом −1.

З а д а ч а 3.

Вычислить сумму S = 1 + 1/2 + 1/3 + ... + 1/50:

```
PROGRAM N3(OUTPUT);
VAR I:=INTEGER; S:=REAL;
 BEGIN S:=0;
```

```
  FOR I:=1 TO 50 DO
    S:=S+1/I;
  WRITELN (' S=',S)
END.
```

Оператор S := S + 1/I; выполняется 50 раз соответственно для I = 1, 2, 3, ..., 50.

Результат S = 4.499...

З а м е ч а н и е 1. Внутри цикла нельзя изменять ни начальное, ни конечное значение переменной цикла, а также само значение переменной цикла.

З а м е ч а н и е 2. Если в цикле по возрастающим значениям переменной начальное значение больше конечного, то цикл не выполняется ни разу. (Аналогично для цикла с «DOWNTO», если начальное значение меньше конечного.)

З а м е ч а н и е 3. После завершения цикла значение переменной цикла «портится» (становится неопределенным).

П р и м е р 5.

```
FOR I:=1 TO 50 DO
  BEGIN
    S:=S+1/I;
    K:=I
  END;
L:=I;
```

В этом фрагменте K будет принимать последовательные значения 1, 2, 3, ..., 50. После завершения всего цикла выполнится оператор L := I;

Будет ошибкой предполагать, что L содержит число 50 — переменная I «испортилась» после окончания цикла. Если необходимо тем не менее запомнить последнее значение переменной I, то следует выполнить оператор L := K;

10. Процедура ввода

Общий вид оператора:

READ (V1, V2, ..., VN);

Здесь V1, V2, ..., VN — идентификаторы переменных.

Значения переменных вводятся с клавиатуры и должны соответствовать типам переменных. Переменные V1, V2, ..., VN могут быть одного из четырех типов: INTEGER, CHAR (либо SUBRANGE этих типов), REAL и STRING.

Пример 1.

```
3 PROGRAM N3(INPUT,OUTPUT);
  VAR A, B, C : REAL; I:INTEGER;
   BEGIN
    READ (A,B,C,I);
    WRITELN (' A=',A,' B=',B,' C=',C,' I=',I)
   END.
```

Для ввода чисел 1.5, 2.15, –1.1, 25 можно набрать:

```
1.5 2.15 –1.1 25
```

Если вводится последовательность символов, то пробел воспринимается как символ. В этом случае и конец строки (EOL) трактуется как символ «конец строки», а соответствующая переменная получает значение «пробел».

Пример 2. Пусть

```
R:REAL;  I:INTEGER;  C1,C2,C3:CHAR;
```

Переменным R, C1, C2, C3, I надо присвоить соответственно значения 1.5, 'A', 'B', 'C', 25. На картах эти значения можно расположить одним из способов:

а)
```
 1.5ABC25
```
б)
```
+1.5E+0ABC
   25
```

Но нельзя после 1.5 поместить пробел, так как он воспримется как значение символьной константы.

Оператор READ (R,C1,C2,C3,I); введет нужные данные.

Конец входного файла и конец строки (карты) входного файла можно определить с помощью функций EOF и EOLN соответственно: функция EOF принимает значение TRUE только в случае исчерпания всех входных данных (конец файла); если исчерпались данные на одной карте (строке входного файла), то значение TRUE принимает другая функция — EOLN.

Если данные вводятся не с перфокарт или с терминала, а с какого-либо иного файла ввода F (магнитной ленты, диска), то используется оператор

READ (F,V1,V2, ..., VN);

Вводимые данные могут быть разделены одним или несколькими пробелами, но нельзя отделять пробелом знак числа либо одни цифры числа от других.

11. Процедура вывода

Процедура WRITE (P1,P2,...,PN) дописывает в выходной файл OUTPUT значения выражений P1, P2, ..., PN.

Процедура WRITELN (P1,P2,...,PN) дописывает в OUTPUT значения P1, P2, P3, ..., PN, заносит признак конца строки EOLN и выдает эту строку.

Пример. Выдача значений A, B может быть осуществлена по-разному:

а) WRITE(A);
WRITE(B);
WRITELN;

б) WRITE(A);
WRITELN(B);

в) WRITELN(A, B);

11.1. Формат вывода данных. Транслятор отводит по умолчанию определенное число позиций для величин каждого стандартного типа.

Тип	CHAR	ALFA	BOOLEAN	INTEGER	REAL	
					M	N
Число позиций на БЭСМ-6	1	6	8	10	14	4
Число позиций на ЕС ЭВМ	1	8	5	12	24	16

Пример 1.

```
PROGRAM M (OUTPUT);
VAR I,J:INTEGER;
 BEGIN
  I:=2;
  J:=15;
```

```
  WRITELN (' I=',I' J=',J)
END.
```

Результат работы программы на БЭСМ-6 будет выглядеть так:

I=_____2_J=_____15

на ЕС ЭВМ:

I=_____2_J=_____15

Программист имеет возможность задать ширину поля (число позиций) M для выводимой величины P:

WRITE (P1:M1, P2:M2, ..., PN:MN);

Если MK избыточна, то поле слева дополняется пробелами; если MK недостаточна для размещения PK, то транслятор сам увеличивает ширину поля так, чтобы уместилось PK, а для БЭСМ-6 слева оставляет еще один пробел (кроме величин типа INTEGER, где пробел не предусмотрен).

Для вещественных значений можно задавать поля M и N, где M — общее число позиций, отводимых под все число PK, N — число позиций под его дробную часть.

П р и м е р 2. WRITE (P:10:2);

Здесь под P отводится 10 позиций, а 2 из них — под дробную часть.

П р и м е р 3. Оператор

WRITELN (3.14159:0:2, 1=1, 3.14:7, 56, 'PAPA':5);

содержит пять выражений:

P1:3,14159
P2:1=1
P3:3.14
P4:56
P5:'PAPA'

Они при выводе изобразятся следующим образом:

а) на БЭСМ-6

3.14E+00 TRUE 3.1400E+00 56 PAPA

б) на ЕС ЭВМ этот оператор является ошибочным: нельзя указывать нулевую ширину поля и необходимо, чтобы было M > N.

12. Сложные типы переменных

Базируясь на простых типах, можно строить и более сложные типы переменных. Переменные сложных типов состоят из отдельных компонент. Тип компонент называют базовым типом данного сложного типа. Таких сложных типов в языке Паскаль четыре: массивы, записи, файлы и множества.

12.1. Массивы (ARRAY). Массив — структура, состоящая из фиксированного числа компонент одного типа.

Общий вид описания массива (в разделе VAR):

A:ARRAY [TYPE1,TYPE2,..., TYPEL] OF TYPEC;

Здесь A — имя массива; TYPE1, TYPE2,..., TYPEL — типы индексов; TYPEC — тип компонент (базовый тип).

Количество индексов (L) определяет размерность массива. Индексы могут быть любых скалярных типов, кроме REAL и INTEGER (не путать с SUBRANGE). Индексы разделяются запятыми и заключаются в квадратные скобки.

П р и м е р 1. Пусть в памяти ЭВМ расположена таблица чисел, представляющая собой двумерный массив M:

15	10	2	30
7	19	55	11
22	18	6	3

Каждое число в таблице имеет тип INTEGER. Это — тип компонент (TYPEC).

Первый индекс — номер строки таблицы — может в данном примере быть от 1 до 3; второй индекс — номер столбца — может меняться от 1 до 4.

Таким образом, описание этого массива выглядит так:

M:ARRAY[1..3, 1..4] OF INTEGER;

Задав конкретные значения индексов, можно выбрать определенную компоненту массива. Например, оператор:

N := M [1, 2];

зашлет в N значение, стоящее в первой строке, втором столбце, т. е. 10.

12.1.1. Упакованные массивы. Если перед названием типа стоит ключевое слово PACKED, то при трансляции генерируется программа, плотно упаковывающая данные в ячейки памяти. Такая программа экономит память, но не время счета, так как для работы с элементами такого массива, как правило, требуется предварительная распаковка.

К упакованному массиву Z можно применить стандартные процедуры PACK и UNPACK.

Пусть массив B имеет тип ARRAY [M..N] OF TYPEC; а массив Z — тип PACKED ARRAY [U..V] OF TYPEC; где N–M ≥ V–U; Тогда PACK(B,I,Z) означает «упаковку» элементов, начиная с первой компоненты массива B, в массив Z. UNPACK(Z,B,I) означает «распаковку» массива Z в массив B, начиная с I-го элемента массива B. Строка из N символов текстовой информации есть не что иное, как упакованный массив:

PACKED ARRAY [1..N] OF CHAR;

Тип ALFA — частный случай упакованного массива:

PACKED ARRAY [1..8] OF CHAR; на ЕС ЭВМ,
PACKED ARRAY [1..6] OF CHAR; на БЭСМ-6.

13. Процедуры

При решении разных задач часто возникает необходимость проводить вычисления по одним и тем же алгоритмам, например, вычислять корень уравнения $f(x) = 0$. В языке Паскаль предусмотрена возможность объединения любой последовательности операторов в самостоятельную подпрограмму, называемую процедурой. Процедуры, используемые во многих задачах, помещены в библиотеку процедур и находятся в памяти ЭВМ (оперативной либо внешней).

Те алгоритмы, которые программистом оформляются как процедуры в его собственной программе, должны начинаться с заголовка и кончаться оператором END;

Процедура Паскаля — аналог фортранной подпрограммы.

Общий вид заголовка:

```
PROCEDURE N (P1:T1; P2:T2; VAR P3:T3, ...);
```

Здесь N — имя процедуры, P1, P2, P3, ... — формальные параметры, T1, T2, T3, ... — их типы.

Процедура имеет ту же структуру, что и главная программа (PROGRAM): разделы LABEL, CONST, TYPE, VAR и выполняемую часть (от BEGIN до END;).

Процедура помещается в главной программе после раздела VAR и перед BEGIN программы.

Пример 1.

```
PROGRAM MA (INPUT, OUTPUT);
VAR A:INTEGER;
    B:REAL; C:CHAR;
PROCEDURE N (P1:REAL; P2:CHAR);
 BEGIN (* НАЧАЛО РАБОТЫ ПРОЦЕДУРЫ *)
. . .
 END; (* КОНЕЦ ПРОЦЕДУРЫ *)
BEGIN (* НАЧАЛО РАБОТЫ PROGRAM *)
. . .
END.
```

Формальные параметры — это наименования переменных, через которые передается информация из программы в процедуру либо из процедуры в программу.

Пусть, например, процедура SQ осуществляет решение квадратного уравнения $ax^2 + bx + c = 0$. Тогда она должна иметь пять формальных параметров: для значений коэффициентов a, b, c и для результатов: x_1 и x_2.

Для того чтобы запустить процедуру в работу, необходимо к ней обратиться (ее вызвать). *Вызов* процедуры N производится оператором вида

```
N(P1,P2,P3, ...);
```

Здесь N — имя процедуры, P1, P2, P3, ... — *фактические параметры*.

При вызове процедуры машина производит следующие действия: устанавливает взаимно однозначное соответствие между фактическими и формальными параметрами, затем управление

передает процедуре. После того как процедура проработает, управление передается *вызывающей программе* на оператор, следующий за вызовом процедуры.

Соответствие между фактическими и формальными параметрами должно быть следующим:

а) число фактических параметров должно быть равно числу формальных параметров;

б) соответствующие фактические и формальные параметры должны совпадать по порядку следования и по типу.

Соответствующие параметры не обязательно должны быть одинаково обозначены.

П р и м е р 2. Вызвать процедуру SQ можно так:

```
SQ(P, Q, R, Y, Z);
```

Здесь P, Q, R — коэффициенты квадратного уравнения, а Y и Z — корни этого уравнения. Если вызвать SQ оператором SQ(X1,X2,A,B,C); то машина воспримет X1, X2, A как коэффициенты уравнения, а корни зашлет в переменные B и C.

П р и м е р 3. Составим процедуру SQ решения квадратного уравнения $ax^2 + bx + c = 0$ в предположении, что дискриминант неотрицателен:

```
PROCEDURE SQ(A,B,C:REAL; VAR X1, X2:REAL);
VAR D:REAL;
 BEGIN
  D:=B * B-4 * A * C;
  X1:=(-B+SQRT(D))/(2 * A);
  X2:=(-B-SQRT(D))/(2 * A)
 END;
```

С помощью этой процедуры решим квадратное уравнение $5,7y^2 - 1,2y - 8,3 = 0$:

```
PROGRAM S (OUTPUT);
VAR Y1, Y2:REAL;
 PROCEDURE SQ (A,B,C:REAL; VAR X1, X2:REAL);
 VAR D:REAL;
   BEGIN
     D:=B * B-4 * A * C;
     X1:=(-B+SQRT(D))/(2 * A);
```

```
      S2:=(-B-SQRT(D))/(2 * A)
    END;
BEGIN (* НАЧАЛО РАБОТЫ ПРОГРАММЫ *)
 SQ(5.7, -1.2, -8.3, Y1, Y2);
 WRITELN (' Y1=', Y1,' Y2=', Y2)
END.
```

Результат X1 = 0.49, X2 = −5.2.

Как видно из примера 3, процедура помещается после декларативных операторов программы. Первым выполняется оператор обращения к процедуре

```
SQ(5.7, -1.2, -8.3, Y1, Y2);
```

Здесь первые три фактические параметра соответствуют формальным A, B, C, а последние два фактических параметра Y1 и Y2 соответствуют формальным X1 и X2. После того как процедура «запустится», в ячейки A, B, C попадут числа 5.7, −1.2, −8.3 и начнут выполняться операторы D := ..., X1 := ..., X2 := ...

После окончания работы процедуры управление возвратится к оператору WRITELN, который отпечатает результат. Параметры процедур могут быть четырех видов: *параметры-значения, параметры-переменные, параметры-процедуры, параметры-функции.*

13.1. Параметры-значения. Если в качестве формального параметра указана переменная, то такой параметр и есть параметр-значение. Примерами таких параметров служат параметры A, B и C в процедуре SQ:

```
PROCEDURE SQ (A,B,C:REAL; VAR X1, X2:REAL);
```

В этом случае фактическим параметром, соответствующим A либо B, либо C, может быть любое выражение соответственного типа, в частности, константа.

Например, обратиться к SQ можно так:

```
SQ((25./3+2)*2, -1.5, (8.2-3.1)/3, X1, X2);
```

Для параметров-значений машина при вызове процедур производит следующие действия: выделяет место в памяти для каждого формального параметра, вычисляет значение фактиче-

ского параметра и засылает его в ячейку, соответствующую формальному параметру.

Если фактический параметр есть имя переменной, например R, то значение этой переменной пересылается в соответствующий формальный параметр, например A. На этом всякая связь между A и R обрывается.

Если даже фактический и формальный параметры одинаково обозначены, в памяти ЭВМ эти параметры занимают разные ячейки. Это полезно знать, чтобы не допустить распространенной среди начинающих программистов ошибки — пытаться передать информацию из процедуры в вызывающую программу через параметр-значение.

Пример 1.

```
PROGRAM T2 (OUTPUT);
VAR I:INTEGER; A:REAL;
 PROCEDURE P(I:INTEGER);
 BEGIN
 I:=I*2
 END;
BEGIN (* НАЧАЛО T2 *)
 I:=2;
 P(I);
 WRITELN ('  I=',I)
END.
```

В T2 происходит засылка числа 2 в ячейку, отведенную для переменной I, затем идет обращение к процедуре P с фактическим параметром I = 2. При этом значение 2 пересылается в другую ячейку, отведенную для формального параметра I. В этой ячейке после выполнения оператора присваивания I := I * 2 появляется число 4. Но после возврата из процедуры на оператор WRITELN программа T2 «знает» только одну переменную I, которая по-прежнему содержит число 2. Поэтому программа напечатает I = 2.

Если формальный параметр есть параметр-значение, то соответствующим фактическим параметром должно быть выражение того же типа, что и формальный параметр.

13.2. Параметры-переменные. Если перед именем формального параметра стоит ключевое слово VAR, то такой параметр есть параметр-переменная. Примерами таких параметров служат X1 и X2 в заголовке процедуры

```
PROCEDURE SQ (A,B,C:REAL; VAR X1, X2:REAL);
```

Фактический параметр, соответствующий параметру-переменной, может быть *только переменной* (не константой и не выражением).

При вызове процедур (функций) параметры-переменные обрабатываются так: для формального параметра используется именно та ячейка, которая содержит соответствующий фактический параметр.

П р и м е р 1. При вызове процедуры SQ оператором SQ (P, Q, R, Y, Z) для переменных X1 и X2 используются непосредственно те ячейки, которые отведены для Y и Z. Поэтому оператор присваивания X1 := (–B+SQRT(D))/(2*A); засылает полученное значение в ячейку Y.

Под формальные и фактические параметры-значения транслятор отводит разные области памяти. Поэтому результат выполнения процедуры может быть передан только через параметр-переменную.

П р и м е р 2.

```
PROGRAM L1351 (OUTPUT);
VAR A,B:INTEGER;
 PROCEDURE H (X:INTEGER; VAR Y:INTEGER);
 BEGIN
  X:=X+1; Y:=Y+1;
  WRITELN (X,Y)
 END;
BEGIN A:=0; B:=0;
 H(A,B);
 WRITELN (A,B)
END.
```

Результаты, выдаваемые процедурой H, 1 и 1; программа печатает 0 и 1.

Разберем пример 2: фактический параметр A соответствует формальному параметру-значению X, а фактический параметр B — формальному параметру-переменной Y. Под параметры A и X отведены две *разные* ячейки памяти, а под B и Y — *одна и та же* ячейка.

При обращении к H (A,B) из ячейки A пересылается значение 0 в ячейку X, а в ячейку Y засылается адрес ячейки B, содержащей 0, так как в процедуре H параметр X — это параметр-значение, а Y — параметр-переменная.

При выполнении оператора X := X + 1 в ячейку X прибавляется 1 и в ячейке X окажется 1, а в ячейке A — по-прежнему 0.

Выполнение оператора Y := Y + 1 имеет следующий смысл: взять число из ячейки, адрес которой находится в Y (т. е. из ячейки B), прибавить 1 и заслать в ту же ячейку (т. е. в B). Поэтому в результате выполнения оператора Y := Y + 1 значение ячейки B станет 1. Оператор печати из процедуры WRITELN (X, Y) выдаст содержимое ячеек X и Y, т. е. 1 и 1. Оператор печати WRITELN (A, B) в программе напечатает содержимое A, которое осталось равным 0, и содержимое ячейки B, которое равно 1.

Процедуры в Паскале допускают *рекурсию*, т. е. процедура может вызвать сама себя.

Если в процедуре P есть обращение к процедуре Q, описанной ниже, то перед описанием P процедура Q декларируется как FORWARD: после заголовка процедуры Q ставится двоеточие, а затем ключевое слово FORWARD*).

Пример 3.

```
PROCEDURE Q(X:T1):FORWARD;
PROCEDURE P(X:Y);
 BEGIN
```

---

*) Обращаем внимание, что в этом случае параметры процедуры описываются только в операторе с FORWARD. В заголовке самой процедуры параметры опускаются. Подробнее о рекурсии сказано в п. 14.3.

```
  Q(A)
 END;
PROCEDURE Q;
 BEGIN
  P(B)
 END;
```

14. Функции

Функция в Паскале является аналогом фортранной подпрограммы-функции и состоит из заголовка и блока.

Общий вид заголовка:

```
FUNCTION F(P1:T1; ...PN:TN):TYPEF;
```

Здесь F — имя функции; P1, ..., PN — формальные параметры; T1, ..., TN — их типы; TYPEF — тип результата.

Самостоятельный алгоритм можно оформить как функцию, если в результате получается единственное значение.

К функциям обращение выглядит проще, чем к процедурам. Для вызова функции достаточно указать ее имя (с фактическими параметрами) в любом выражении.

Пусть, например, функция MAX (X,Y:REAL):REAL выдает значение большего из параметров X, Y. Тогда оператор

```
R := MAX(T,P*Q)*B/2;
```

найдет большее из значений T и P*Q, затем выполнит дальнейшие действия и зашлет результат в R.

После работы функции результат присваивается имени функции, поэтому в блоке функции обязательно должен присутствовать оператор присваивания вида:

⟨имя функции⟩ := ⟨результат⟩;

П р и м е р. Вычисление большего из двух данных чисел можно оформить таким образом:

```
FUNCTION MAX (X,Y:REAL):REAL;
 BEGIN
  IF X>Y THEN MAX:=X ELSE MAX:=Y
 END;
```

Процедуры, функции, программы часто называют *модулями*.

14.1. Побочные эффекты. Если в теле процедуры (функции) перевычисляется некоторая *нелокальная* переменная, т. е. такая переменная, которая описана в других модулях, содержащих данный, то могут наблюдаться непредвиденные последствия.

П р и м е р 1. Пусть функция F(X) имеет такой вид:

```
FUNCTION F(X:REAL):REAL;
 BEGIN
  V:=V*X;
  F:=SQRT(V)+X
 END;
```

т. е. в процессе работы функция F изменяет некоторую нелокальную величину V. Рассмотрим теперь два выражения, которые вычисляются в программе, содержащей F(X):

F(X)+V и V+F(X).

Эти выражения дадут *разные* результаты, так как в первом случае к F(X) прибавится уже *измененное* значение V (в процессе работы F), а во втором случае к *первоначальному* значению V добавляется F(X).

Вторая опасность заключается в неправильном использовании параметров-переменных в качестве формальных параметров.

П р и м е р 2. Найти пятый член последовательности

$a_{n+1} = 3a_n - 2,$

$a_1 = 1.$

Опасно оформлять функцию в виде

```
FUNCTION F (VAR A,N:INTEGER):INTEGER;
VAR I:INTEGER;
 BEGIN
  FOR I:=1 TO N DO A:=3*A-2;
  F:=A
 END;
```

Так, если обратиться к этой функции оператором B:=F(1,5), будет «испорчена» константа 1, поскольку в ячейку памяти (первый фактический параметр), содержавшую ранее единицу,

функция F поместит текущий член последовательности, и при дальнейшей работе программы вместо 1 будет использоваться значение a_5. Такие ошибки бывает трудно найти, поэтому полезно придерживаться следующего правила: в функциях не использовать параметры-переменные.

14.2. Параметры-процедуры. Параметры-функции. Такие параметры в списке формальных параметров предваряются ключевыми словами PROCEDURE и FUNCTION соответственно.

Пример 1.

```
PROCEDURE P (PROCEDURE A);
```

Здесь процедура P имеет один параметр-процедуру A.

Пример 2.

```
PROCEDURE Q (FUNCTION S:REAL; B:REAL);
```

Процедура Q имеет два параметра: параметр-функцию S и параметр-значение B.

Замечание 1. На ЕС ЭВМ имеются *отличия от стандарта* в описании параметров-функций (процедур). Транслятор требует перечислить все параметры функции, являющейся параметром.

Пример 3.

```
PROCEDURE Q (FUNCTION F(I:INTEGER):REAL);
```

Здесь формальный параметр F — функция от одного целого аргумента, результат F — вещественный.

Если вызывается процедура (функция), имеющая параметр-процедуру (функцию), то соответствующий фактический параметр должен совпадать по типу результата с формальной процедурой (функцией). Программисту необходимо внимательно следить за совпадением типов результатов, так как в случае нарушения этого правила *никакой диагностики не выдается*, а программа работает неверно. Поясним на примере.

Обратиться к процедуре Q(FUNCTION F(I:INTEGER):REAL) можно так: Q(SINUS); функция SINUS(K) есть SIN(K). Если K имеет тип INTEGER, тогда SINUS(K) — типа REAL. Это совпадает с типами I и F в заголовке Q. Нельзя, однако, обратиться к Q с функцией ABS(K), а именно: Q(ABS); в этом

случае тип формального параметра F — REAL, а тип фактического ABS(K) — INTEGER, т. е. формальный и фактический параметры не совпадают по типу.

З а д а ч а. Составить процедуру выдачи таблицы произвольной вещественной функции. Процедура должна иметь следующие формальные параметры: вещественную функцию, нижнюю границу аргумента, верхнюю границу аргумента, шаг по аргументу:

```
PROCEDURE TAB (FUNCTION F:REAL; LOW, UP, STEP:REAL);
VAR X:REAL;
    J:INTEGER;
BEGIN
 X:=LOW;
 FOR J:=0 TO TRUNC ((UP-LOW)/STEP) DO
  BEGIN
   WRITELN (X:10,F(X):10);
   X:=X+STEP
  END
END;
```

(см. замечание 2).

Выражение TRUNC ((UP–LOW)/STEP) дает число точек, в которых вычисляется функция F (при счете от 0).

Если к процедуре TAB обратиться оператором

```
TAB (SIN, 0.0, 6.4, 0.33);
```

то будет напечатана таблица функции sin x для x от 0 до 6.4 с шагом 0.33. Алгоритмы, употребляемые наиболее часто различными пользователями, оформляются в виде процедур и функций, помещаются в память машины и составляют библиотеку стандартных программ (модулей).

З а м е ч а н и е 2. Многие трансляторы, в том числе БЭСМ-6 и ЕС ЭВМ, *не допускают* использования стандартных функций в качестве фактических параметров. Для таких трансляторов оператор

```
TAB (SIN, 0.0, 6.4, 0.33);
```

является ошибочным, так как SIN(X) — стандартная функция.

Это ограничение можно легко обойти, введя новую функцию, эквивалентную стандартной.

П р и м е р 4. Введем функцию SINUS(X) таким образом:

```
FUNCTION SINUS (X:REAL):REAL;
BEGIN
 SINUS:=SIN(X)
END;
```

Тогда составить таблицу sin x можно, обратившись к процедуре TAB следующим оператором:

```
TAB(SINUS, 0.0, 6.4, 0.33);
```

При использовании параметров-процедур и параметров-функций надо иметь в виду возможные осложнения.

1. Ошибки, допускаемые программистом в процедурах, имеющих параметры-процедуры и параметры-функции, иногда бывает трудно найти, что ведет к длительной отладке таких процедур.

2. Если число и тип параметров формального параметра-функции не совпадает с числом либо типом параметров соответствующего фактического параметра-функции, то такая программа не может быть правильно выполнена, а многие версии трансляторов с Паскаля не выдают в этом случае никакой диагностики.

3. Правила языка Паскаль требуют, чтобы фактические параметры-процедуры (функции) содержали только параметры-значения. Это накладывает серьезные ограничения на использование параметров-процедур и параметров-функций.

14.3. Рекурсии. В языке Паскаль процедуры и функции могут вызывать сами себя, т. е. обладать свойством рекурсивности.

П р и м е р 1. Выдать на печать в обратном порядке цифры целого положительного числа N. Составим процедуру REVERSE:

```
PROCEDURE REVERSE (N:INTEGER);
BEGIN
 WRITE (N MOD 10);
 IF (N DIV 10)< > 0
```

```
  THEN REVERSE (N DIV 10)
END;
```

Рассмотрим работу этой процедуры для числа N = 153. Оператор WRITE (N MOD 10) выдает в файл OUTPUT остаток от деления числа 153 на 10, т. е. последнюю цифру 3. Оператор

```
IF (N DIV 10) < > 0
THEN REVERSE (N DIV 10)
```

проверяет целую часть частного 153/10 = 15 на нуль. Если целая часть не нуль, то с этим значением (15) идет вновь обращение к процедуре REVERSE. Оператор WRITE (N MOD 10) дописывает в файл OUTPUT остаток от деления числа 15 на 10, т. е. 5; затем со значением 15/10 = 1 идет обращение к REVERSE. После вывода цифры 1 оператором WRITE (N MOD 10) проба N DIV 10 на нуль передает управление на конец процедуры. В файле OUTPUT будет записано число 351. Его можно выдать оператором WRITELN.

Таким образом, *однократное* обращение извне к процедуре REVERSE вызвало *трехкратное* ее срабатывание. Условие полного окончания работы рекурсивной процедуры должно находиться *в самой процедуре* (иначе произойдет зацикливание).

Рекурсивные процедуры и функции (модули) имеют одну из двух форм: *прямую рекурсию* и *косвенную рекурсию*. В первом случае модуль содержит оператор вызова *этого же модуля*, как в приведенной выше процедуре REVERSE. Во втором случае один модуль вызывает другой модуль, который либо сам, либо посредством других модулей вызывает исходный модуль.

П р и м е р 2. Если A, B — имена модулей, то схема вызова может быть такой: A → B → A.

В случае косвенной рекурсии возникает проблема: как и где описать вызываемый модуль. По правилам языка Паскаль каждый вызываемый модуль должен быть описан *до его вызова*. Но если модуль A вызывает B, а B вызывает A, то получается замкнутый круг. Для подобных ситуаций принято следующее правило: один из рекурсивных модулей, вызывающих друг друга, описывается *предварительно* следующим образом:

```
PROCEDURE P (<список форм. параметров>); FORWARD;
```

Здесь P — имя процедуры, FORWARD — ключевое слово. Это описание указывает транслятору, что текст процедуры P помещен ниже.

Подобным же образом описывается функция: к оператору FUNCTION добавляется слово FORWARD. Список формальных параметров и тип результата (для FUNCTION) включается только в это предварительное описание и опускается в заголовке соответствующего модуля.

П р и м е р 3. Пусть функция B при работе вызывает функцию A, которая, в свою очередь, вызывает функцию B. Тогда эти модули можно описать так:

```
FUNCTION B(X:INTEGER):REAL; FORWARD;
FUNCTION A(Y:INTEGER):REAL;
 BEGIN
 . . .
  A:=B(I) + 3.5
 END;
FUNCTION B;
 BEGIN
 . . .
  B=A(D)–1.8
 END;
```

Заголовок (FUNCTION B) перед текстом функции B содержит только имя этой функции (список формальных параметров и тип функции не указываются).

14.4. Локальные и глобальные переменные. Напомним, что каждый модуль (процедура, функция, программа) состоит из заголовка (PROCEDURE..., FUNCTION..., PROGRAM...) и блока.

П р и м е р 1.

```
PROCEDURE P(A:REAL; VAR X:REAL); (* ЗАГОЛОВОК *)
VAR K:INTEGER; (* НАЧАЛО БЛОКА *)
 BEGIN
 . . .
 END; (* КОНЕЦ БЛОКА *)
```

Вложенные процедуры. Если блок какой-либо процедуры P1 содержит внутри процедуру P2, то говорят, что P2 *вложена* в P1.

Пример 2.

```
PROCEDURE P1 (X:REAL; VAR Y:REAL);
VAR C:INTEGER;
 PROCEDURE P2 (VAR Z:REAL);
 . . .
 END;
BEGIN
. . .
END;
```

Любые идентификаторы, введенные внутри какого-либо блока (процедуры, функции) для описания переменных, констант, типов, процедур, называются *локальными* для данного блока. Такой блок вместе с вложенными в него модулями называют *областью действия* этих локальных переменных, констант, типов и процедур.

Пример 3.

```
PROCEDURE T1;
VAR Y1, Y2:REAL;
 PROCEDURE SQ1;
 VAR A,B,C,D:REAL;
 BEGIN
 (* ПЕРЕМЕННЫЕ A,B,C,D ЯВЛЯЮТСЯ ЛОКАЛЬНЫМИ ДЛЯ
 SQ1, ОБЛАСТЬ ИХ ДЕЙСТВИЯ — ПРОЦЕДУРА SQ1 *)
 . . .
 END;
BEGIN
(* ПЕРЕМЕННЫЕ Y1, Y2 — НЕЛОКАЛЬНЫЕ ДЛЯ SQ1,
ОБЛАСТЬ ИХ ДЕЙСТВИЯ — T1 И SQ1 *)
. . .
END;
```

Константы, переменные, типы, описанные в блоке PROGRAM, называют *глобальными*. Казалось бы, проще иметь дело вообще только с глобальными переменными, описав их все в PROGRAM. Но использование локальных переменных позволяет системе лучше оптимизировать программы, делает их более наглядными и уменьшает вероятность появления ошибок.

При написании программ, имеющих вложенные модули, необходимо придерживаться следующих правил.

1. Описывать идентификаторы в том блоке, где они используются, если это возможно.

2. Если один и тот же объект (переменная, тип, константа) используется в двух и более блоках, то описать этот объект надо в самом внешнем из них, содержащем все остальные блоки, использующие данный объект.

3. Если переменная, используемая в процедуре, должна сохранить свое значение до следующего вызова этой процедуры, то такую переменную надо описать во внешнем блоке, содержащем данную процедуру.

Локализация переменных дает программисту большую свободу в выборе идентификаторов. Так, если две процедуры A и B полностью отделены друг от друга (т. е. не вложены одна в другую), то идентификаторы в них могут быть выбраны совершенно произвольно, в частности, могут повторяться. В этом случае совпадающим идентификаторам соответствуют разные области памяти, совершенно друг с другом не связанные.

Пример 4.

```
PROGRAM T2 (OUTPUT); VAR K:INTEGER;
 PROCEDURE A;
 VAR X,Z:REAL;
 BEGIN (* НАЧАЛО A *)
 (* ЧЕРЕЗ X, Z ОБОЗНАЧЕНЫ ДВЕ ВЕЛИЧИНЫ —
 ЛОКАЛЬНЫЕ ПЕРЕМЕННЫЕ ДЛЯ A;
 K — ГЛОБАЛЬНАЯ ПЕРЕМЕННАЯ ДЛЯ A *)
 . . .
 END; (* КОНЕЦ A *)
 PROCEDURE B;
 VAR X,Y:INTEGER;
 BEGIN (* НАЧАЛО B *)
 (* ЧЕРЕЗ X, Y ОБОЗНАЧЕНЫ ДВЕ ДРУГИЕ ВЕЛИЧИНЫ —
 ЛОКАЛЬНЫЕ ПЕРЕМЕННЫЕ ДЛЯ B; K — ГЛОБАЛЬНАЯ
 ПЕРЕМЕННАЯ ДЛЯ B *),
 . . .
 END; (* КОНЕЦ B *)
BEGIN (* НАЧАЛО РАБОТЫ PROGRAM T2 *)
(* K — ЕДИНСТВЕННАЯ ПЕРЕМЕННАЯ, КОТОРУЮ МОЖНО
ИСПОЛЬЗОВАТЬ В T2 *)
```

```
. . .
END.
```

Если один и тот же идентификатор описан в блоке B и второй раз описан во вложенном в B блоке C, то надо помнить, что эти два одинаковых идентификатора соответствуют *разным* ячейкам памяти.

Пример 5.

```
PROGRAM T3 (OUTPUT);
VAR I:INTEGER; A:REAL;
 PROCEDURE P(VAR D:REAL);
 VAR I:INTEGER;
 BEGIN (* НАЧАЛО P *)
  I:=3;
  D:=I+10*D
 END; (* КОНЕЦ P *)
BEGIN (* НАЧАЛО T3 *)
 A:=2.0;
 I:=15;
 P(A);
 WRITELN (' I=',I,' A=',A)
END.
```

Глобальным переменным I и A отводятся две ячейки памяти. Первыми выполняются операторы A := 2.0 и I := 15. Затем вызывается процедура P(A). В процессе работы P отводится ячейка для локальной переменной I и туда засылается число 3. После окончания работы процедуры P эта ячейка I программой «забывается». После возврата на оператор WRITELN программа «знает» только одну ячейку I — глобальную, т. е. ту, которая содержит число 15. Поэтому программа T3 выдаст на печать I = 15, A = 23.0, так как A = 3 + + 10 * 2.

Если локальная и глобальная переменные принадлежат к одному и тому же сложному типу, то этот тип надо описать в разделе TYPE, а сами переменные описывать через этот общий тип.

Пример 6.

```
PROGRAM T1 (OUTPUT);
```

```
TYPE AB=ARRAY [1..3] OF REAL;
VAR A:AB;
 PROCEDURE Q;
 VAR B:AB;
 . . .
 END.
```

В этом примере переменные A и B описаны через общий тип AB. Если же локальная и глобальная переменные описаны одинаково, но не через общий тип, то программа может «не понять», что эти переменные принадлежат одному типу.

Пример 7.

```
PROGRAM T2 (OUTPUT);
VAR A:ARRAY [1..3] OF REAL;
 PROCEDURE Q;
 VAR B:ARRAY [1..3] OF REAL;
 . . .
END.
```

В этом примере переменные A и B — одинаковые массивы, т. е. типы этих переменных одинаковы, но программа тем не менее «не считает», что A и B принадлежат одному типу. Это происходит из-за того, что описание массивов дано в разных блоках.

15. Стандартные процедуры и функции

Наиболее часто употребляемые библиотечные процедуры и функции называют стандартными. Для работы с ними не надо ни заказывать библиотеку, ни описывать их предварительно в программе.

Часть приводимых ниже процедур (от RESET до DISPOSE) будут описаны в гл. II, поэтому при первом чтении их можно опустить.

1) ORD(C) — выдает порядковый номер символа C в упорядоченной последовательности символов, задаваемой транслятором. Нумерация начинается с нуля.

2) CHR(I) — выдает символ с порядковым номером I. Функции ORD(C) и CHR(I) взаимообратны, т. е. CHR(ORD(C)) = C и ORD(CHR(I)) = I. В силу упорядоченности последовательно-

сти символов, ORD(C1) < ORD(C2), если C1 < C2 и, вообще, если R — логическая операция отношения (=, < >, >, <, >=, <=), то из ORD(C1) R ORD(C2) следует C1 R C2, и обратно.

3) PRED(C) — выдает значение, предшествующее C в упорядоченной последовательности значений.

4) SUCC(C) — выдает значение, следующее за C. Результат этих двух функций будет неопределенным, если соответствующих значений не существует; в этом случае не все трансляторы выдают диагностику.

5) ODD(X) — выдает TRUE, если X нечетно (X целое).

6) элементарные математические функции:

ABS(X) — модуль X,

SQR(X) — квадрат числа X

(обратить внимание: нельзя на БЭСМ-6 использовать выражение X**2). Эти две функции выдают результат того же типа, что и аргумент.

З а м е ч а н и е. В этом, в общем удобном для программиста свойстве, кроется и источник «тяжелых» ошибок.

Например, одна из этих функций является фактическим параметром. Смена типа аргумента приведет к смене типа фактического параметра. Но тип соответствующего формального параметра строго фиксирован в описании соответствующего модуля, т. е. смена начальных данных в такой программе может привести к ошибке, которой при других данных не было.

TRUNC(X) — целая часть X, получающаяся при *отбрасывании* знаков после десятичной точки. Результат целый, X — вещественный.

П р и м е р 1.

TRUNC (3.9) дает 3,

TRUNC (–3.9) дает –3.

ROUND (X) — вещественный X *округляется* до целого, выдаваемого как результат.

П р и м е р 2.

ROUND(3.9) дает 4,

ROUND(–3.9) дает –4.

Аргумент следующих функций может быть как вещественным, так и целым; результат — всегда вещественный:

LN(X) — натуральный логарифм от X,

EXP(X) — *e* в степени X,

SQRT(X) — квадратный корень из X,

SIN(X) — синус X,

COS(X) — косинус X,

ARCTAN(X) — арктангенс X.

7) PACK(A, K, Z) — осуществляет упаковку *одномерного* массива A, начиная с K-й компоненты, в массив Z с 1-й компоненты. (Массив Z типа PACKED.)

8) UNPACK (Z, A, K) — осуществляет распаковку *одномерного* массива Z, начиная с 1-й компоненты, в массив A с K-й компоненты. (Массив Z типа PACKED.)

9) RESET(F) — устанавливает указатель файла на первую компоненту файла F и считывает ее в окно*) F↑; функции EOF(F) присваивается значение FALSE, если файл не пуст; иначе — значение F↑ не определено и функция EOF(F) имеет значение TRUE.

10) REWRITE(F) — очищает файл (т. е. F становится пустым файлом), устанавливает указатель файла на первую компоненту, присваивает функции EOF(F) значение TRUE. К функции REWRITE(F) необходимо обратиться *перед записью* в первую компоненту файла F.

11) GET(F) — в случае, когда EOF(F)=FALSE, продвигает указатель файла к следующей компоненте и присваивает значение этой компоненты окну F↑; если эта компонента — «конец файла», то значение F↑ становится неопределенным, а EOF(F) принимает значение TRUE.

12) PUT(F) — заносит значение F↑ в ту компоненту файла, куда установлен указатель, если значение EOF(F) есть TRUE к моменту выполнения PUT(F). После выполнения PUT(F) значение F↑ становится неопределенным.

Пример 3.

```
VAR DATA:FILE OF INTEGER;
```

---

*) Подробнее о файлах см. п. 21.

```
  A:INTEGER;
BEGIN
. . .
  A:=SQR(DATA↑);
    GET (DATA);
    . . .
```

Здесь оператор A := SQR(DATA↑); присваивает переменной A квадрат текущей компоненты файла DATA; оператор GET (DATA) продвигает указатель к следующей компоненте файла и читает ее в переменную DATA↑.

13) WRITELN(F) — записывает символ «конец строки» в текущую компоненту текстового файла F.

14) READLN(F) — пропускает остаток текущей строки, устанавливает указатель файла на начало следующей строки текстового файла F.

15) EOLN(X) — принимает значение TRUE, если указатель файла установлен на символ конца строки (FALSE в противном случае) и засылает в X пробел.

16) EOF(X) — принимает значение TRUE, если X соответствует концу файла (FALSE — в противном случае).

17) NEW(P) — отводит место в памяти для новой динамической переменной и в P запоминает ее адрес (см. п. 22.1).

18) NEW(P,P1) — отводит место в памяти для записи (RECORD) с вариантом P1.

19) DISPOSE(P) — стирает динамическую переменную, на которую указывает переменная P.

20) DISPOSE(P,P1) — стирает динамическую переменную, созданную процедурой NEW(P,P1) и на которую указывает переменная P (см. п. 22.3).

16. О кодировке символов

Для ЭВМ БЭСМ-6 все символы, кроме ↑, соответствуют символам клавиатуры дисплея и перфоратора ЕС-9080 (кодировка КПК-12). Символ ↑ соответствует символу @ (коммерческое А). Если карты пробиты в кодировке КПК-12, то для БЭСМ-6 с операционной системой Дубна в качестве первой карты надо положить карту с пробивками 7/9 IBM (7/9 означает в режиме МР пробивку 7 и 9 в первой позиции).

На ЕС ЭВМ используется такое представление символов:

Стандартное	[	]	{	}	AND	OR	NOT	< >
Паскаль ЕС	(.	.)	(*	*)	&	!	¬	¬=

17. Дополнительные сведения о языке Паскаль для ЕС ЭВМ

17.1. Как читать листинг задачи. Каждая страница листинга начинается с информации о версии транслятора, дате и времени запуска программы.

Затем выдается информация о режиме трансляции. Например, если задан режим B+, то следующая строка будет INITIAL OPTIONS:B+. Далее идет программа на Паскале (см. пример).

Первая вертикальная колонка чисел слева — номера соответствующих строк программы.

Следующая вертикальная колонка четырехзначных шестнадцатеричных чисел содержит относительные адреса команд и переменных.

Для переменных в разделе VAR этот адрес указывает относительное смещение переменной от начала транслированной процедуры. В теле процедуры (и программы) адрес есть относительный адрес от начала процедуры. Относительные адреса используются в ссылках раздела ERROR MESSAGE — списка диагностик транслятора.

Следующая вертикальная колонка состоит из двух символов индикаторов вложений циклов и составных операторов. Когда в программе встречается первый оператор BEGIN, то вместо тире слева выдается нуль. Когда в какой-либо строке появляется END, соответствующий этому BEGIN, то в правом разряде двузначного «числа» появляется нуль.

Если внутри первой конструкции появляется вторая (BEGIN, CASE, REPEAT), то в левом разряде выдается 1, а для соответствующего END(UNTIL) в правом разряде появляется 1. Правильно составленная программа должна начинаться нулем в левом разряде индикатора вложения у первого оператора BEGIN программы и оканчиваться нулем в правом разряде у последнего оператора END. Если в программе есть раздел

процедур и функций, то на листинг выводится буква — индикатор вложения модулей (процедур и функций). Эта буква выводится слева от заголовка модуля и его BEGIN и END. Для уровня вложения 2 принята буква A, для уровня вложения 3 — буква B и т. д. Этот индикатор делает наглядной структуру процедур и позволяет сразу увидеть ошибку в случае пропуска END в конце модуля.

Пример.

```
1 0680  -  -    PROGRAM L10T (INPUT,OUTPUT);
2 06A8  -  -    VAR CH:CHAR;
3 0000  0  -    BEGIN
4 0062  -  -      WHILE NOT EOF DO;
5 008A  1  -       BEGIN WRITE (' ');
6 0002  -  -      WHILE NOT EOLN DO;
7 00EE  2  2       BEGIN  READ(CH);WRITE(CH) END;
8 016C  -  1      WRITELN;READLN END
9 0184  -  0    END.
```

17.2. Ошибки, обнаруживаемые при трансляции. Рассмотрим пример:

```
1 0680  -  -    PROGRAM L6T1(INPUT,OUTPUT),
                ***ERROR***                !_14,18
2 06A8  -  -    VAR A,B,R:REAL; I:INTEGER;
3 0000  0  -      BEGIN
. . .                . . .
11 0196 -  0      END.
```

Ошибки, обнаруженные при трансляции, отмечаются в тексте программы символом «!», выдаваемым под ошибкой, и номером ошибки.

Ниже текста программы печатается расшифровка всех найденных ошибок (ERROR MESSAGE). Так, в приведенном примере будет напечатано:

14: Пропущена ; (возможно в предыдущей строке).

18: Ошибка в декларативной части.

17.3. Коды завершения трансляции.

0 — трансляция закончена, ошибок нет;

4 — выдана предупредительная диагностика, фатальных ошибок нет, программа может быть выполнена;

8 — были фатальные ошибки, программа не выполняется;

12 — ошибка системы;

16 — ошибка транслятора.

Ошибки 12 и 16 могут быть следствием ошибок в управляющих картах (например, мало заказано памяти, не заказан соответствующий транслятор и т. д.).

Если управляющие карты правильные, то при кодах 12 и 16 необходимо обратиться к консультанту. Подробнее о кодах завершения здесь мы говорить не будем.

17.4. Внутреннее представление данных.

Тип	Число байт	Величина
INTEGER	4	32 бита
BOOLEAN	4	1 = TRUE; 0 = FALSE
CHAR	4	EBCDIC — код в младшем (правом) байте
REAL	8	Число с плавающей запятой
SET	8	Элементы, представляемые побитно, начиная слева
SCALAR	4	Порядковый номер величины в типе, начиная с нуля

В упакованных структурах скалярные типы занимают чаще всего один байт, если порядковый номер упакованных элементов лежит в диапазоне от 0 до 255.

Пример.

X:INTEGER (занимает 4 байта),

X:PACKED ARRAY [1..4] OF CHAR (занимает 4 байта),

B:ARRAY [1..4] OF CHAR (занимает 16 байт),

C:REAL (занимает 8 байт).

18. Диагностика ошибок, обнаруженных при трансляции

В случае обнаружения ошибки в тексте программы транслятор выдает следующее сообщение (диагностику).

На ЭВМ БЭСМ-6 подробная диагностика появляется прямо в тексте программы перед ошибочным оператором, ошибка отмечается символом 0.

На ЕС ЭВМ в тексте программы под ошибочным оператором появляется только номер допущенной ошибки и символ !, отмечающий место ошибки в программе. Тексты подробных сообщений об ошибках выдаются единым массивом ниже текста всей программы.

После того как транслятор обнаружил ошибку, он пытается возобновить анализ программы, пропустив часть текста до ожидаемого символа. Часто удается успешно продолжить трансляцию; иногда это может привести к *наведенным* ошибкам.

Например, пусть при описании переменной I допущена следующая ошибка:

```
I,INTEGER;
```

т. е. вместо двоеточия стоит запятая.

Транслятор выдаст сообщение об ошибке, а затем будет «ругать» каждый оператор, использующий переменную I. Эти ошибки наведенные, они являются следствием того, что тип переменной I описан в ошибочном операторе. Ошибки, допущенные в программе, бывают двух видов: *фатальные* и *нефатальные*. Если ошибка нефатальная, транслятор выдает предупредительную диагностику, но программа *выходит на счет*. Если допущена фатальная ошибка, программа *на счет не выходит*.

18.1. Сообщения об ошибках. Приведем список всех сообщений об ошибках, которые может обнаружить транслятор.

1: Ошибка в простом типе.
2: Пропущен идентификатор.
3: Пропущен оператор.
4: Пропущена ')'.
5: Пропущено ':'.
6: Недопустимый символ (возможно, пропущен символ ';').
7: Ошибка в параметрах.
8: Пропущено OF.
9: Пропущена '('.
10: Ошибка в типе.
11: Пропущена '[' (или '(.').
12: Пропущена ']' (или '.)').

13: Пропущен END.
14: Пропущена ';' (возможно, строкой выше).
15: Должно быть INTEGER.
16: Пропущено '='.
17: Пропущен BEGIN.
18: Ошибка в разделе описаний.
19: Ошибка в списке полей.
20: Пропущена запятая.
21: Пропущена точка.
21*: Пропущена переменная*).
50: Ошибка в константе.
51: Пропущен знак ':='.
52: Пропущено THEN.
53: Пропущено UNTIL.
54: Пропущено DO.
55: Пропущено TO или DOWNTO.
56: Пропущено IF.
57: Пропущено слово FILE.
58: Ошибка в множителе (ошибочное выражение).
59: Ошибка в переменной.
60: Пропущено IN.
101: Дважды описанный идентификатор.
102: Нижняя граница больше верхней.
103: Идентификатор не принадлежит соответствующему классу.
104: Неописанный идентификатор.
105: Здесь знак не допускается.
106: Пропущено число.
107: Несовместимые ограниченные типы.
108: Здесь FILE не допускается.
109: Здесь тип не может быть REAL.
110: Тип переключателя должен быть скалярным или ограниченным.
111: Тип несовместим с типом переключателя.
112: Тип индекса не может быть REAL.
113: Индекс должен иметь скалярный тип либо ограниченный.

---

*) Номера ошибок со звездочкой приводятся для ЕС ЭВМ.

114: Базовым типом не может быть REAL.
115: Базовым типом должен быть скалярный либо ограниченный.
116: Ошибка в типе параметров стандартной процедуры.
117: Недопустимая ссылка на еще не описанное понятие.
118: Неописанный тип используется при описании переменной.
118*: Распаковку/упаковку применять нельзя; проверьте элементы массива.
119: Повторное описание списка параметров не допускается.
120: Тип результата функции может быть скалярным, ограниченным либо POINTER.
121: Параметр-значение не может быть файлом.
122: Функция уже декларирована (FORWARD); не допускается повторное описание типа результата функции.
123: Пропущен тип результата в описании функции.
124: Формат F допустим только для REAL.
125: Ошибка в типе параметра стандартной функции.
126: Число параметров иное, чем в описании функции (процедуры).
127: Недопустимые фактические параметры.
128: Тип результата параметра-функции не соответствует описанию.
129: Несовместимость типов операндов.
130: Тип выражения — не SET.
131: Допускается проверка только на равенство.
132: Не допускается строгое включение.
133: Не допускается сравнение файлов.
134: Недопустимый тип операнда.
135: Операнд должен иметь тип BOOLEAN.
136: Тип элемента множества скалярный или ограниченный.
137: Несовместимые типы элементов множества.
138: Тип переменной — не массив.
139: Тип индекса не соответствует описанию.
140: Тип переменной — не RECORD.
141: Тип переменной должен быть FILE либо POINTER.
142: Недопустимые типы фактических параметров.
143: Недопустимый тип переменной цикла.

144: Недопустимый тип выражения.
144*: Тип переключателя скалярный или ограниченный.
145: Несоответствие типов.
145*: Несовместимость с типом управляющей переменной.
146: Нельзя использовать файлы в операторе присваивания.
147: Тип метки не соответствует типу выражения переключателя.
148: Границы диапазона должны иметь скалярный тип.
149: Индекс может иметь тип ограниченный, но не INTEGER.
150: Нельзя присваивать значения стандартным функциям.
151: Нельзя присваивать значение формальному параметру-функции.
152: В данной записи нет такого поля.
153: Ошибка в типе параметра READ.
154: Фактический параметр должен быть переменной.
154*: Пропущена переменная.
155: Переменная цикла не может быть формальным параметром.
156: Одинаковые метки в CASE.
157: Слишком много вложенных операторов CASE.
158: Пропущено описание соответствующего варианта.
159: Типы REAL и строка недопустимы для переключателя.
160: Пропущено описание FORWARD.
161: Повторно описано FORWARD.
162: Размер памяти, занимаемый параметром, должен быть фиксированным.
162*: Внешние модули не могут быть описаны как FORWARD.
163: Пропущен вариант в описании.
164: Не допускается подстановка стандартной процедуры (функции).
165: Метки не должны повторяться.
166: Дважды описанная метка.
167: Неописанная метка.
168: Отсутствует метка, описанная в LABEL.
169: Ошибка в базовом типе SET.
170: Параметр должен быть параметром-значением.
170*: Тип переменной должен здесь быть ограниченным, скалярным либо POINTER.

171: Стандартный файл не требует описания.
172: Неописанный внешний файл.
173: Должна быть фортранная процедура или функция.
174: Должна быть паскаль-процедура или функция.
175: В заголовке PROGRAM пропущен файл INPUT.
176: В заголовке PROGRAM пропущен файл OUTPUT.
177: Здесь не допускается присваивание имени функции.
177*: Переменная цикла FOR должна быть локальной.
178: Дважды описанный вариант в RECORD.
178*: За именем файла данных не может следовать «/» в заголовке программы.
179*: Отсутствует оператор присваивания значения функции ее идентификатору.
180: Переменная цикла не может быть формальным параметром.
180*: Слишком длинная исходная строка.
181: Значение переключателя — вне диапазона.
182: Присваивание идентификатору функции должно стоять внутри FUNCTION.
183: Не происходит присваивания идентификатору функции в области действия этой функции.
185: Оператор присваивания идентификатору функции должен находиться в блоке этой функции.
186: Ошибка в заголовке процедуры; несоответствие фактических и формальных параметров по числу или по типу.
186*: Ошибка в заголовке процедуры.
187: Компоненты упакованных переменных не могут быть параметрами-переменными.
188: Идентификатор должен быть описан раньше, чем использован в других описаниях.
189: Ошибка в ширине поля для форматного ввода/вывода.
190: Нельзя изменять оператором присваивания переменную цикла.
191: Переменная цикла не может быть фактическим параметром-переменной.
201: Ошибка в вещественной константе (возможно, в ее записи присутствуют не только цифры, либо отсутствует точка).

202: Константа типа строка не должна выходить за пределы исходной строки.
203: Слишком большая целая константа.
204: Нулевая строка не допускается.
204*: В восьмеричном числе не может быть цифр 8 и 9.
205: Нулевая строка не допускается.
205*: Величина либо диапазон элементов SET ошибочны.
220: Инициализация переменных допускается только в главной программе.
221: Несоответствие типов при инициализации переменных.
222: Несоответствие числа компонент описанию структурной константы.
223: Несоответствие типов компонент описанию структурной константы.
224: Недопустимый формат структурной константы.
225: Отсутствует '*)' (т. е. правая фигурная скобка).
226: Недопустимый тип структурной константы.
227: Недопустима запись с вариантами для структурных констант.
250: Слишком много уровней вложения областей действия идентификаторов.
251: Слишком много вложенных процедур и (или) функций.
252: Слишком много FORWARD.
253: Слишком длинная процедура.
254: Слишком много констант в этой процедуре.
254*: Слишком большие переменные.
255: Слишком много ошибок в этой строке.
256: Слишком много внешних ссылок.
257: Слишком много EXTERNAL.
258: Слишком много локальных файлов.
259: Слишком сложное выражение.
260: Слишком много ENTRY.
261: Слишком много процедур либо переходов к глобальным меткам.
280: Имя EVENT не декларировано.
281: Нельзя использовать оператор POSTLUDE для события EXIT.

282: Дважды описанный оператор POSTLUDE.

292: Нельзя изменять тип константы.

293: Режим U устанавливает номер позиции, с которой начинается комментарий.

300: Деление на нуль.

300*: Инициализация величин не допускается во внешних модулях.

301: Нет поля CASE для данного значения.

302: Значение индекса вышло из диапазона.

303: Присваиваемое значение вышло из диапазона.

304: Значение элемента SET вышло из диапазона.

305: 'NIL' можно засылать только в переменную типа POINTER.

306: Тип POINTER не может указывать на переменную, содержащую поле типа FILE.

310: Внутри комментария символ ';' либо '(*'.

311: На метку, описанную в разделе LABEL, нет ссылки.

320: Структурная константа — расширение стандартного Паскаля.

321: Раздел инициализации переменных — расширение стандартного Паскаля.

322: Оператор FORALL — расширение стандартного Паскаля.

323: Оператор LOOP — расширение стандартного Паскаля.

324: Упрощенная форма оператора CASE — расширение стандартного Паскаля.

325: Задание диапазона меток в операторе CASE — расширение стандартного Паскаля.

326: Стандартный Паскаль допускает здесь только идентификатор типа.

327: Использование оператора ** — расширение стандартного Паскаля.

328: Функция изменения типа — расширение стандартного Паскаля.

329: Спецификация интерактивного файла — расширение стандартного Паскаля.

330: Ввод строки символов — расширение стандартного Паскаля.

331: Автоматическая инициализация переключателя — расширение стандартного Паскаля.

332: Расширение стандартного Паскаля; в стандартном Паскале этот тип надо описать.

333: Расширение стандартного Паскаля; в стандартном Паскале эту процедуру надо описать.

334: Расширение стандартного Паскаля; в стандартном Паскале эту функцию надо описать.

335: Процедура ORD(POINTER) — расширение стандартного Паскаля.

336: 'EXTERN', 'PASCAL', 'FORTRAN' — расширение стандартного Паскаля.

337: Стандартный Паскаль требует описания внешнего файла в главной программе.

338: Метка должна содержать не более четырех цифр в стандартном Паскале.

339: Стандартный Паскаль не допускает использование символа $ в идентификаторе.

380: Нельзя передавать процедуру или функцию во внешний модуль.

381: Ошибка в типе результата внешней функции.

382: Нельзя отменять режим (*$E+ *).

383: Можно транслировать только две процедуры одного уровня вложения после установления режима E.

384: Фактический параметр не может быть фортранным модулем.

385: Нельзя устанавливать режим E после трансляции внутренней процедуры.

389: Ограничение, накладываемое реализацией системы.

389*: Неожиданное появление оператора END.

399: Массивы переменной размерности не реализованы в системе.

399*: Не реализовано в системе.

400: Ошибка транслятора. Обратитесь к консультанту.

ГЛАВА II

ДЛЯ ТЕХ, КТО РЕШИЛСЯ ИДТИ ДАЛЬШЕ

19. Записи (RECORD)

Запись — структура, состоящая из фиксированного числа компонент, называемых полями. В одном поле данные имеют один и тот же тип, а в разных полях могут иметь разные типы. Общий вид описания типа RECORD:

```
TYPE T=RECORD
 ID11,ID12,...ID1N:TYPE1;
 ID21,ID22,...ID2L:TYPE2;
 . . .
IDK1,IDK2,...IDKM:TYPEK
  END;
```

Здесь IDIJ — идентификаторы полей; TYPEI — типы полей; T — имя типа.

Пример 1. Данные комплексного вида можно описать переменной типа RECORD:

```
TYPE COMPLEX=RECORD
 RE,IM:REAL
 END;
VAR C:COMPLEX;
```

Здесь COMPLEX — имя типа, а C — имя переменной. Переменная C состоит из двух полей: RE и IM, имеющих один и тот же тип (REAL). Эти поля переменной C обозначаются как C.RE и C.IM.

Пример 2. Даты каких-либо событий можно описать следующим образом:

```
TYPE DATE=RECORD
 MONTH:1..12;
```

```
  DAY:1..31;
  YEAR:INTEGER
    END;
VAR D:DATE;
```

В этом примере описан тип DATE и переменная D, принадлежащая этому типу. Переменная D описана как запись, состоящая из трех полей: MONTH, DAY и YEAR. Каждое поле содержит соответственно данные: целое число в пределах от 1 до 12 (номер месяца), целое число от 1 до 31 (число), целое число (год). Поле DAY переменной D записывается как D.DAY

Например, чтобы заслать в D дату 12.01.1985, надо выполнить следующие операторы:

```
D.MONTH:=1;
D.DAY:=12;
D.YEAR:=1985;
```

П р и м е р 3. Вычислить сумму S двух комплексных чисел X = 2 + 7i и Y = 6 + 3i (т. е. X, Y, S:COMPLEX; см. пример 1). Фрагмент программы выглядит так:

```
X.RE:=2.0; X.IM:=7.0;
Y.RE:=6.0; Y.IM:=3.0;
S.RE:=X.RE + Y.RE;
S.IM:=X.IM + Y.IM;
```

Записи, как и массивы, могут быть упакованными.

П р и м е р 4.

```
X:PACKED RECORD
  A:1..5;
  B:CHAR
    END;
```

Запись может быть компонентой других структур. Например, введем тип FAMILY (семья: отец, мать, 1-й ребенок 2-й ребенок):

```
TYPE FAMILY=(FATHER,MOTHER,CHILD1,CHILD2);
VAR BIRTHDAY:ARRAY [FAMILY] OF DATE;
```

где DATE — описанная выше запись.

Переменная BIRTHDAY есть массив, состоящий из записей — дат рождения членов семьи: отца, матери, 1-го ребенка, 2-го ребенка. Каждая дата рождения имеет тип DATE, который должен быть описан в программе.

Для занесения даты рождения, например, матери (MOTHER) достаточно выполнить операторы:

```
BIRTHDAY[MOTHER].MONTH:=5;
BIRTHDAY[MOTHER].DAY:=1;
BIRTHDAY[MOTHER].YEAR:=1950;
```

Занесется дата рождения матери — 1.5.1950.

19.1. Оператор WITH. Этот оператор используется для удобства работы с переменными типа RECORD (запись). Общий вид:

```
WITH A DO ST;
```

Здесь A — имя переменной типа RECORD, ST — оператор.

В операторе ST при ссылках на компоненты записи имя A можно опускать.

П р и м е р. Для занесения даты рождения в предыдущем примере достаточно выполнить операторы:

```
WITH BIRTHDAY[MOTHER] DO
BEGIN
    MONTH:=5;
    DAY:=1;
    YEAR:=1950
END;
```

19.2. Запись с вариантами. Общий вид:

```
TYPE V=RECORD
 A:TYPE1
 B:TYPE2
 . . .
  CASE N:TYPEN OF
   C1:(T11:TYPE11; T12:TYPE12;...);
   C2:(T21:TYPE21; T22:TYPE22,...);
     . . .
   CK:(TK1:TYPEK1; TK2:TYPEK2;...)
```

```
END;
VAR Z:V;
```

Здесь Z — переменная типа V; N — переменная, называемая *переключателем*; TYPEN — тип переменной N.

Этому же типу должны принадлежать метки C1,C2, ..., CK. Каждой метке соответствует набор *полей* T11,T12, ... Эти поля являются *компонентами варианта.*

Переменную N называют также тагом (тэгом), ярлыком, признаком, дискриминантом.

Если какой-либо метке CL вообще не соответствуют поля, то пишут CL:();

З а м е ч а н и е 1. Любая запись (RECORD) может иметь только одну вариантную часть (CASE).

З а м е ч а н и е 2. Вариантная часть должна помещаться после постоянной части.

З а м е ч а н и е 3. Среди идентификаторов полей не должно быть одинаковых. Обращение к компоненте Z.Tij записи Z происходит так:

1) Присваивается соответствующее значение (C_i) переключателю N. В зависимости от значения N переменная Z, помимо полей A, B, ..., содержит те поля, которые соответствуют той метке C_i, с какой совпадает значение N.

2) Выполняется операция с компонентой Z.Tij.

П р и м е р 1 (записи с вариантами). Пусть необходимо собрать следующие сведения о сотрудниках института: фамилию, дату рождения и, если есть семья, то фамилию и дату рождения супруги (супруга).

Эта информация может быть описана, например, записью PERSON.

Пусть переменная типа KIND может иметь одно из значений («женат», «холост»).

```
KIND=(MARRIED,SINGLE);
 PERSON=RECORD
  NAME:ALTA;
  DATEBIRTH:DATE;
  CASE YESNO:KIND OF
   MARRIED:(NAME1:ALFA;DATE1:DATE);
```

```
    SINGLE:()
  END;
```

Здесь NAME — строка символов (например, 'ROGOV '); DATEBIRTH — запись, описанная выше, содержит дату рождения (например, 15.02.62); YESNO — переключатель типа KIND, который может принимать одно из двух значений: MARRIED либо SINGLE; NAME1 — строка символов, содержащая фамилию супруги (супруга) (например, 'ROGOVA'); DATE1 — запись, содержащая дату рождения супруги (супруга); SINGLE — пустое поле.

Если ROGOV женат, то присутствует поле MARRIED, если холост — поле SINGLE, а поле MARRIED отсутствует. Паскаль допускает вложение вариантов в типе RECORD.

П р и м е р 2. Пусть необходимо, помимо информации предыдущего примера, иметь о сотрудниках следующие сведения: если сотрудник холост, но состоял в браке раньше, то указать, когда разведен.

Опишем тип KIND как (женат, холост, разведен, нет):

```
KIND=(MARRIED,SINGLE,DEVORCED,NO);
```

Тогда

```
PERSON=RECORD
  NAME:ALFA;
  DATEBIRTH:DATE;
  CASE YESNO:KIND OF
    MARRIED:(NAME1:ALFA; DATE1:DATE);
    SINGLE: (CASE YN:KIND OF
      DEVORCED:(DATEDV:DATE));
      NO:()
  END;
```

Здесь для варианта SINGLE имеется вложенная запись с вариантами DEVORCED (разведен) и NO.

Если сотрудник состоит в браке, то в записях информации отсутствует поле SINGLE; если разведен, то отсутствует MARRIED; если в браке не состоял, то запись содержит лишь поле NAME, DATEBIRTH и пустое поле NO.

З а м е ч а н и е 4. Перед засылкой информации в запись программист должен присвоить переключателю соответст-

вующее значение. В противном случае информация (например, MARRIED) в поле заслана не будет, и система никакой диагностики не выдаст.

Пример 3 (засылки информации о сотруднике РОГОВЕ, родившемся 1.12.32, женатом на РОГОВОЙ, родившейся 15.3.30 (ЭВМ БЭСМ-6)):

```
PROGRAM L8T3(INPUT,OUTPUT);
TYPE KIND=(MARRIED,SINGLE);
   DATE=RECORD
    DAY:1..31;
    MONTH:1..12:
    YEAR:INTEGER
    END;
   PERSON=RECORD
    NAME:ALFA;
    DATEBIRTH:DATE;
    CASE YESNO:KIND OF
     MARRIED:(NAME1:ALFA; DATE1:DATE);
     SINGLE:(NAMESING:ALFA)
    END;
VAR P:PERSON;
BEGIN
   WITH P DO
    BEGIN
     YESNO:=MARRIED;
     NAME:='ROGOV ';
     WITH DATEBIRTH DO
      BEGIN
       DAY:=1;
       MONTH:=12;
       YEAR=1932
      END;
     CASE YESNO OF
      MARRIED:BEGIN
         NAME1:='ROGOVA';
         WITH DATE1 DO
          BEGIN
```

```
            DAY:=15;
            MONTH:=3;
            YEAR:=1930
           END
         END;(* MARRIED *)
       SINGLE:NAMESING:='SINGLE'
      END (* CASE YESNO OF *)
    END (* WITH P DO *)
  WITH P DO
   WRITE (' ', NAME);
  WITH P.DATEBIRTH DO
   WRITELN (' ',DAY,'/',MONTH,'/',YEAR);
  WITH P DO
   WRITE (' ',NAME1);
  WITH P.DATE1 DO
   WRITELN (' ',DAY,'/',MONTH,'/',YEAR)
END.
```

Более сложная программа, иллюстрирующая работу с переменными типа «запись», приведена в Приложении 1.

20. Множества (SET)

Рассмотрим несколько примеров.

П р и м е р 1. Пусть имеется множество из трех монет разного достоинства: 1 к., 5 к., 10 к. Из этих монет можно составить следующие подмножества (их число равно $2^3 = 8$):

1) {1 к.};
2) {5 к.};
3) {10 к.};
4) {1 к., 5 к.};
5) {1 к., 10 к.};
6) {5 к., 10 к.};
7) {1 к., 5 к., 10 к.};
8) пустое множество ϕ.

Эти подмножества и будут принадлежать некоторому множеству, тип которого назовем SUM; сами элементы (монеты), из которых составляется подмножество, пусть принадлежат некоторому базовому типу, который назовем MONET.

Опишем типы данных этого примера:

```
TYPE MONET=(KOP1,KOP5,KOP10);
 SUM=SET OF MONET
```

П р и м е р 2. Рассмотрим в качестве элементов базового типа сигналы от четырех абонентов (AB1, AB2, AB3, AB4), поступающие на телефонную станцию. Обозначим базовый тип через ABONENT:

```
TYPE ABONENT=(AB1,AB2,AB3,AB4);
```

тогда комбинации сигналов можно описать переменной типа «множество». Назовем этот тип SIGN:

```
SIGN=SET OF ABONENT;
```

Тип SIGN описывает такие комбинации:

1) AB1; 2) AB2; 3) AB3; 4) AB4; 5) AB1 и AB2; 6) AB1 и AB3; 7) AB1 и AB4; 8) AB1 и AB3; 9) AB2 и AB4; 10) AB3 и AB4; 11) AB1, AB2 и AB3; 12) AB1, AB2 и AB4; 13) AB1, AB3 и AB4; 14) AB2, AB3 и AB4; 15) AB1, AB2, AB3 и AB4; 16) отсутствие сигналов.

В общем виде тип «множество» описывается так:

```
TYPE A = SET OF TC;
```

Здесь A — идентификатор типа (произвольный); TC — тип компонент множества, называемый базовым типом.

Значение переменной типа «множество» изображается путем перечисления конкретных компонент, разделенных запятыми и заключенных в квадратные скобки.

П р и м е р 3. Пусть базовый тип INT и тип A заданы так:

```
TYPE INT=1..3;
  A=SET OF INT;
```

Переменная типа A в этом случае может принимать следующие значения: [1], [3], [2], [1,3], [1,2], [3,2], [1,3,2], [ ], где [ ] означает пустое множество. Например, если переменная B имеет тип A, то можно присвоить ей одно из значений: B:=[1,3]; B:=[1,3,2]; и т. д.

П р и м е р 4. Если базовый тип описывает набор двоичных бит, то можно получить их комбинацию. Пусть

```
TYPE BIN=(BIT1,BIT2,BIT3);
     BTS=SET OF BIN;
```

Переменная типа BTS может принимать значения: [BIT1], [BIT2], [BIT3], [BIT1,BIT2], [BIT1,BIT3], [BIT2,BIT3], [BIT1,BIT2,BIT3], [ ].

Таким образом, используя переменные типа SET, можно работать с битовой информацией.

Паскаль допускает множества, состоящие из ограниченного числа элементов N ≤ NMAX, где NMAX — машинозависимая константа:

на БЭСМ-6 NMAX = 48,
на ЕС ЭВМ NMAX = 63.

В качестве базового типа можно использовать любой простой тип, кроме REAL. Если в качестве базового типа выбран тип CHAR, то допускается использовать не более NMAX первых символов, имеющихся в распоряжении транслятора. Если задача требует использования множества, состоящего из большего числа элементов, то его можно представить как массив множеств, состоящих из допустимого числа элементов.

Для ограниченных типов от базового типа INTEGER можно использовать в качестве компонент целые числа от нуля до N ≤ NMAX − 1

Пример 5.

```
TYPE N=(0..62);
VAR P:SET OF N;
```

20.1. Данные типа SET. Данные типа SET задаются путем перечисления значений, разделенных запятыми и заключенных в квадратные скобки. Общий вид:

[EXPR1,EXPR2,...EXPRN];

Здесь EXPRI — выражения базового типа. Порядок следования выражений несуществен. Непустой набор может быть также выражением вида:

[EXPR1],
[EXPR1..EXPRK],
[EXPR1,EXPRK..EXPRN].

Данные вида [EXPR1..EXPRK] соответствуют набору всех элементов базового типа от значения EXPR1 до EXPRK.

Пример 1. [3*6–7..15+4] соответствует набору [11..19], т. е.

[11,12,13,14,15,16,17,18,19].

Если окажется, что для [I..J], I > J, то такое множество интерпретируется как пустое, а в случае I = J — как множество, содержащее один элемент — I.

Пример 2. [3*6–5+6] эквивалентно [11].

Пример 3.

```
TYPE COLOR=(RED,YELLOW,GREEN,BLUE);
VAR MIX:SET OF COLOR;
. . .
MIX:= [RED,BLUE];
```

Напомним, что на ЕС ЭВМ символам '[' и ']' соответствуют пары символов '(.' и '.)'.

Пример 4.

```
TYPE N=(1,3,5,7,9);
VAR K:SET OF N;
. . .
 K:=[3..9];
```

В этом случае в K зашлется комбинация [3,5,7,9].

20.2. Операции с переменными типа SET. К переменным типа SET применимы следующие операции:

=, < >, >=, <=, IN, +, –, *

Операции = и < > используются для проверки эквивалентности: два значения переменной типа SET считаются равными, если они состоят из одних и тех же элементов.

Пример 1.

[1,3] = [3,1]	дает TRUE,
[1..3] = [1,2,3]	дает TRUE,
[1] < > [2]	дает TRUE,
[1,2,3] = [1,4,3]	дает FALSE,
[RED, BLUE] = [RED, YELLOW]	дает FALSE.

Операции >= и <= используются для проверки принадлежности одного множества другому: так, если множество A содержится во множестве B, то A <= B дает TRUE.

Пример 2.

[1, 2] <= [1,2,3] дает TRUE.

Пустое множество [ ] содержится во всех множествах, т. е. всегда [ ] <= [B] дает TRUE.

Операция IN используется для установления наличия определенного элемента в величине типа SET. Так, если X есть элемент множества B, то (X IN B) дает TRUE. Общий вид:

X IN A;

Здесь X — величина базового типа, A — величина типа SET.

Пример 3.

RED IN [RED,YELLOW] дает TRUE;
RED IN [BLUE,GREEN] дает FALSE.

Замечание 1. Чтобы проверить, является ли значение N цифрой, удобно использовать операцию IN следующим образом:

IF N [0..9] THEN...

Замечание 2. Результат операции IN может быть неопределенным в некоторых случаях.

Пример 4. Пусть

A: SET OF 1..50;
X: INTEGER;

Если заслать в X число, большее максимального значения 50 (например, X:= 55), то в этом случае результат операции X IN A не всегда FALSE.

К переменным типа SET, относящимся к одному и тому же конкретному типу, применимы операции: + (объединение); * (пересечение); – (дополнение).

Пусть A и B — операнды, имеющие один и тот же конкретный тип. Тогда:

A + B — объединение множеств элементов, входящих в A и B (одинаковые элементы не повторяются);

A * B — пересечение множеств элементов A и B;

A – B — множество элементов, которые есть в A, но отсутствуют в B

Пример 5.

[1,3] + [1,4] дает [1,3,4];
[1,3] * [1,4] дает [1];
[1,3] – [1,4] дает [3].

Операция A := A + X добавляет элемент X к множеству A. Если X уже имелся в A, то множество A не меняется. Операция A := A – X исключает X из A. Если X отсутствовал в A, то множество A не меняется.

21. Файлы (FILE)

Для того чтобы понятие файла было для читателя достаточно ясным, рассмотрим процесс записи на магнитную ленту. Запись производится посредством магнитных головок на тот участок ленты, который находится против головки. Для простоты будем считать, что головка может перемещаться вдоль магнитной ленты (в действительности головки закреплены, а лента движется).

Пример 1. Пусть записывающая головка (изобразим ее стрелкой) установлена против участка A магнитной ленты (рис. 2).

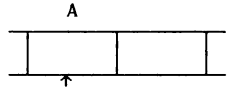


Рис. 2

Если на ленту будет записано некоторое число 153, то головка после записи переместится к следующему участку ленты (B) (рис. 3).

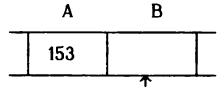


Рис. 3

Пусть на ленту таким образом будет записана последовательность чисел 153, 512, 25, –13 (рис. 4).

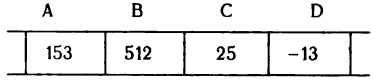


Рис. 4

В этом случае говорят, что на магнитной ленте находится файл (переменная типа FILE) длиной 4, а записывающая головка продвинулась за пределы файла.

Файл представляет собой последовательность компонент одного и того же типа. Число компонент не фиксировано. В каждый момент доступна только одна компонента. Говорят, что на эту компоненту установлен *указатель файла*.

Если выполнилась операция записи в n-ю компоненту файла, то указатель автоматически продвигается к (n + 1)-й компоненте, т. е. для записи становится доступной уже только (n + 1)-я компонента.

Длиной файла называется число записанных компонент. Файл, не содержащий компонент, называется *пустым*, его длина равна нулю. Читать файл можно также только последовательно по одной компоненте. Файлы Паскаля поэтому называют *файлами последовательного доступа*. Начать писать в файл можно только *с самого его начала*, дописывая новые компоненты последовательно одну за другой; для чтения также надо начинать просмотр файла *с самого начала*.

Вследствие такой организации на одном просмотре файла нельзя совмещать и чтение, и запись информации: можно либо только читать, либо только писать.

Для того чтобы определять готовность файла к чтению либо к записи информации, существует стандартная функция EOF(F), где F — имя файла. Если указатель файла продвинулся за конец файла (готовность к записи), то эта функция принимает значение TRUE, во всех остальных случаях эта функция принимает значение FALSE.

Общий вид описания типа FILE:

```
TYPE R = FILE OF TC;
```

Здесь R — идентификатор типа, TC — тип компонент (может быть любым, кроме типа FILE).

Файлы могут быть разных типов: состоять из целых компонент, либо вещественных, либо записей и т. д. Как и другие переменные, каждую переменную-файл надо описать в разделе VAR. Вводя имя переменной файла (имя файла), надо указать,

какого типа файл. Этот тип должен быть обозначен каким-либо именем и описан в разделе TYPE.

Например, файл F вещественных чисел:

```
TYPE N=FILE OF REAL;
(* описание переменной файла *)
VAR F:N;
```

Файл может быть описан и непосредственно при описании переменной, например

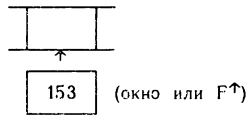
```
VAR F:FILE OF REAL;
```

В первом случае введено имя файла F, а его тип имени не имеет и поэтому в разделе TYPE не описывается.

Если переменная F имеет тип FILE, то транслятор автоматически вводит переменную $F^{\uparrow}$, называемую буферной переменной, окном, через которое можно прочитать или записать одну компоненту. Переменная $F^{\uparrow}$ имеет тот же тип, что и компоненты файла F. Окно устанавливается против той компоненты, где находится указатель файла.

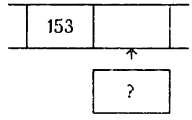
П р и м е р 2. Рассмотрим на схеме запись в файл чисел 153, 512, 25.

1)



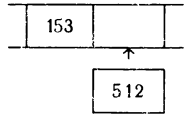
В окно $F^{\uparrow}$ записано число 153.

2)



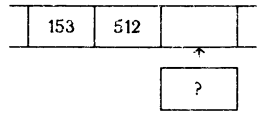
Выполнен приказ на запись в файл. Значение $F^{\uparrow}$ «испорчено», там уже не 153.

3)



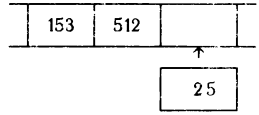
В окно занесено число 512.

4)



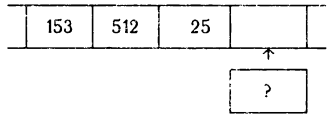
Выполнен приказ на запись в файл.

5)



В окно занесено число 25.

6)



Выполнен приказ на запись в файл.

После записи в файл значение буферной переменной F↑ становится неопределенным («портится»).

Файлы создаются не только на магнитных лентах, но и на магнитных дисках, других внешних устройствах, а также в оперативной памяти. Способы работы с такими файлами с точки зрения программы во всех этих случаях одинаковы: доступна только одна компонента файла посредством окна. Файлы прямого доступа в стандарте языка Паскаль отсутствуют [17].

Работа с файлами (F) проводится посредством следующих стандартных процедур.

1. RESET(F) — подготовка к чтению информации из файла с именем F. Эта процедура осуществляет следующие действия: если файл пустой, то указатель файла устанавливается на 1-ю компоненту и эта компонента считывается в окно (F↑); функции EOF(F) присваивается значение FALSE. Если файл был пустым, то значение F↑ «портится», а EOF(F) присваивается значение TRUE.

2. GET(F) — чтение компоненты файла F. Процедура выполняется только при условии, если EOF(F) имеет значение

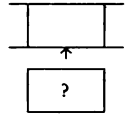
FALSE. Проверку этого условия необходимо сделать в программе перед обращением к GET(F). Процедура продвигает указатель к следующей компоненте файла и считывает эту компоненту в окно (F↑); если указатель выйдет за пределы файла, то функция EOF(F) принимает значение TRUE, а значение F↑ «испортится».

3. REWRITE(F) — подготовка к записи информации в начало файла F. Процедура очищает файл F и устанавливает указатель файла на 1-ю компоненту; функции EOF(F) присваивается значение TRUE.

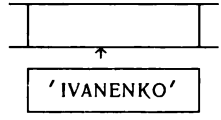
4. PUT(F) — запись компоненты в файл F. Перед обращением к этой процедуре в программе необходимо проверить значение функции EOF(F): файл готов к записи только при значении TRUE. Процедура записывает в файл значение буферной переменной (F↑) и передвигает указатель за пределы файла, готовясь записать следующую компоненту. Значение EOF(F) остается TRUE, а F↑ «портится».

П р и м е р 3. Проиллюстрируем работу процедур PUT и REWRITE для следующей задачи: записать в начало файла GRUPP2 следующие компоненты: 'IVANENKO', 'SHELKOV'. Очевидно, файл содержит компоненты типа ALFA (см. 6.5.1), буферная переменная также типа ALFA. (Файл GRUPP2 содержал ранее другие значения.)

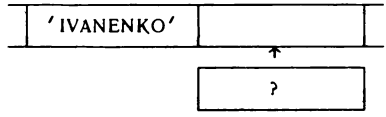
1) Выполним процедуру REWRITE (GRUPP2):



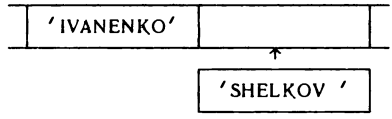
2) Занесем 1-е значение в буферную переменную
GRUPP2↑:='IVANENKO';



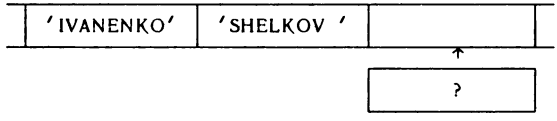
3) Выполним процедуру PUT (GRUPP2):



4) Занесем 2-е значение в буферную переменную:



5) Выполним процедуру PUT(GRUPP2):



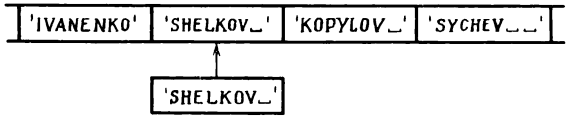
Для упрощения работы с файлами введены процедуры чтения информации из файла (READ) и записи в файл (WRITE), которые освобождают программиста от манипуляций с буферной переменной.

Общий вид:

READ (F, X1, X2, ... , XN);

Здесь F — имя файла, X1, X2, ..., XN — переменные, куда считываются компоненты файла, начиная с той компоненты, какая была занесена в окно файла.

П р и м е р 4. Пусть состояние файла GRUPP2 соответствует схеме



Тогда после выполнения оператора

READ (GRUPP2,X1,X2,X3);

(где X1, X2, X3 должны иметь тип ALFA) картинка будет выглядеть так:

Значения переменных X1, X2, X3 будут следующими:

X1	X2	X3
'SHELKOV '	'KOPYLOV '	'SYCHEV '

Процедура записи в файл имеет вид

WRITE (F, A_1, A_2, ..., A_n);

Здесь F — имя файла, A_1, A_2, ..., A_n — *выражения* того же типа, что и компоненты файла. Процедура записывает значения выражений A_1, ..., A_K по одному в файл F, начиная с того места, куда был установлен указатель файла в момент обращения к процедуре WRITE.

П р и м е р 5. Пусть A1='TEVELEV ', A2='RODIONOV', A3='MALYSHEV', а состояние файла GRUPP2 такое, каким оно было после выполнения оператора READ. Тогда после работы оператора WRITE(GRUPP2,A1,A2,A3); файл GRUPP2 будет таким:

В стандарте языка Паскаль работа с файлами через READ и WRITE допускается только для компонент, имеющих тип CHAR. Но трансляторы на ЕС ЭВМ и БЭСМ-6 не накладывают ограничений: для любых файлов можно использовать эти процедуры.

В качестве примеров рассмотрим три задачи, имеющие практическое значение.

П р и м е р 6. Запись информации в файл F1 (переменная X должна иметь тип компонент файла). Фрагмент программы такой:

```
REWRITE (F1);
REPEAT
(* ПОЛУЧЕНИЕ ЗНАЧЕНИЯ X — ОЧЕРЕДНОЙ
   КОМПОНЕНТЫ ФАЙЛА *)
WRITE(F1,X)
UNTIL <условие окончания>
```

П р и м е р 7. Чтение всей информации из файла F2 (переменная Y должна иметь тип компонент файла F2):

```
RESET(F2)
WHILE NOT EOF(F2) DO
BEGIN
 READ(F2,Y);
 (* ОБРАБОТКА ОЧЕРЕДНОЙ КОМПОНЕНТЫ *)
END;
```

П р и м е р 8. Записать в файл F1 числа 2, 3, 4, 5, а в файл F2 — числа 2!, 3!, 4!, 5!. Затем прочитать из файлов F1 и F2 эти числа и напечатать:

```
//TEST5 JOB XXXXX,SEMASHKO,MSGLEVEL=(2.0)
// EXEC PASCLG
//PASC.SYSIN DD *
PROGRAM FL(INPUT, OUTPUT);
VAR I,J,X,Y:INTEGER;
   F1,F2:FILE OF INTEGER;
BEGIN
(* ЗАПИСЬ В ФАЙЛ *)
    REWRITE(F1); REWRITE(F2);
    X:=1; Y:=1;
    REPEAT
     X:=X+1; Y:=Y*X;
     WRITE(F2,Y);WRITE(F1,X)
    UNTIL X=5;
    (* ЧТЕНИЕ ИЗ ФАЙЛА *)
    RESET(F1); RESET(F2);
```

```
    WHILE NOT EOF(F1) DO
     BEGIN
       READ (F2, Y);
       READ (F1, X);
       WRITELN(' X= ,X, ' Y=',Y)
     END
END.
//GO.P@ LOCAL DD UNIT=SYSDA,
//              DISP=OLD,VOL=SER=RESMVT
//GO.SYSIN DD *
//
```

Результат работы программы:

X = 2; Y = 2;
X = 3; Y = 6;
X = 4; Y = 24;
X = 5; Y = 120.

Пример 9. Копирование файла F2 в файл F1. Компоненты файлов F1 и F2, а также переменная X должны иметь один и тот же тип:

```
RESET(F2);
REWRITE(F1);
WHILE NOT EOF(F2) DO
BEGIN
 READ(F2,X); WRITE(F1,X)
END;
```

21.1. Внешние файлы. Файлы по отношению к программе могут быть внешними и внутренними.

Внутренними файлами являются такие, которые создаются, используются и существуют только во время работы данной программы.

Однако часто возникает необходимость использовать файлы, созданные ранее и хранящиеся на внешних запоминающих устройствах. Например, программы, обрабатывающие данные физических экспериментов, вводят эти данные с файлов на магнитных лентах либо дисках; эти файлы были записаны ранее, возможно, в ходе самого эксперимента. С другой сторо-

ны, обработка данных физического эксперимента, как правило, представляет собой сложную задачу, решение которой ведется в несколько этапов. На каждом этапе получается файл промежуточных результатов, который используется впоследствии другими программами. Такие файлы, которые существуют вне программы, называют *внешними файлами*.

Внешние файлы указываются в списке параметров PROGRAM и соответственно описываются в управляющих картах.

21.2. Текстовые файлы. Особое значение имеют файлы, компонентами которых являются символы. Такие файлы называются *текстовыми*.

Тип текстового файла (TEXT) в каждом трансляторе с Паскаля заранее предопределен как

```
TEXT=FILE OF CHAR;
```

(описывать в программе этот тип не требуется). Текстовый файл состоит из последовательности строк, каждая из которых содержит величины типа CHAR и заканчивается специальным символом «конец строки» (EOL).

Стандартные файлы INPUT и OUTPUT являются текстовыми.

С символом «конец строки» оперируют следующие процедуры:

WRITELN(F) — записывается символ «конец строки» в компоненту файла, на которую установлен указатель файла.

READLN(F) — пропускается оставшаяся часть текущей строки и указатель файла устанавливается на первый символ новой строки. В окно файла F↑ считывается этот символ.

EOL(F) — функция принимает значение TRUE, если указатель установлен на символ «конец строки», и засылает пробел в F↑.

Если F — текстовый файл, а CH — символьная переменная, то можно использовать следующие формы процедур записи и чтения.

Для записи в текстовый файл F значения символьной переменной CH можно воспользоваться процедурой

```
WRITE (F,CH) вместо F↑:=CH; PUT(F);
```

Если V1, V2, V3, ..., VN — символьные переменные, то мож-

но их значения записать в файл F процедурой

```
WRITELN(F,V1,V2,V3,...,VN) вместо
WRITE(F,V1);WRITE(F,V2);...,WRITE(F,VN);
WRITELN(F);
```

При этом следом за значением VN в файл F запишется символ «конец строки».

Аналогично можно воспользоваться оператором

```
READ (F, CH); вместо CH:=F↑; GET(F);
```

для чтения из текстового файла F.

Если надо прочитать N символов и перейти к новой строке файла F, то можно использовать оператор

```
READLN(F,V1,V2,V3,...,VN) вместо
READ(F,V1); READ(F,V2);...,READ(F,VN);
READLN(F);
```

21.3. Стандартные текстовые файлы INPUT и OUTPUT. Стандартный ввод и вывод в языке Паскаль осуществляется с помощью текстовых файлов INPUT и OUTPUT (файлов стандартного типа TEXT), описанных в качестве параметров PROGRAM.

Для упрощения работы с такими файлами предоставлены дополнительные возможности: по умолчанию для переменной CH типа CHAR

```
WRITE(CH) соответствует WRITE(OUTPUT, CH);
READ(CH)  соответствует READ(INPUT, CH);
WRITELN   соответствует WRITELN(OUTPUT);
EOF       соответствует EOF(INPUT);
EOLN      соответствует EOLN(INPUT);
READLN    соответствует READLN(INPUT).
```

К файлам INPUT и OUTPUT нельзя применять процедуры RESET и REWRITE.

Первый символ каждой строки файла OUTPUT управляет устройством печати и на печать не выводится. Если этот символ «пробел», то —переход к следующей строке; «0» — пропуск строки; «1» — переход к началу следующей страницы листинга; «+» — печать без перехода к новой строке (печать с наложением строк).

Пример 1. Печать значения A с новой страницы:

```
WRITE('1',A)
```

Пример 2. Печать содержимого внешнего текстового файла X:

```
PROGRAM L10T2 (OUTPUT,X);
VAR CH:CHAR;
 X:FILE OF CHAR;
BEGIN WRITE(' ');
 RESET (X); WHILE NOT EOF(X) DO
 BEGIN
   WHILE NOT EOLN(X) DO
   BEGIN READ(X,CH); WRITE(CH) END;
   WRITELN; READLN(X)
 END
END.
```

Кроме того, для файлов INPUT и OUTPUT процедуры READ и WRITE позволяют работать с параметрами не только типа CHAR, но и параметрами типа BOOLEAN, REAL и INTEGER. Если первый параметр процедур READ и WRITE — текстовый файл, то информация из него читается или в него записывается; если первый параметр — не текстовый файл, то автоматически информация читается из файла INPUT или записывается в файл OUTPUT соответственно. Трансляторы ЕС ЭВМ и БЭСМ-6 позволяют также вводить и выводить переменные типа ALFA, строки символов.

22. Ссылки (POINTER)

До сих пор мы имели дело с такими переменными, которые описывались в разделе VAR какого-либо блока программ. Транслятор после анализа этого раздела отводит каждой переменной соответствующее число ячеек памяти и закрепляет их за данной переменной на все время работы блока. Такие переменные называют *статическими*. Они не могут быть использованы системой под другие нужды, даже если в процессе дальнейшей работы программы эти переменные больше не понадобятся.

Данные могут быть организованы и другим образом. Их можно хранить в некоторой области памяти, не обозначая именем переменной, а используя ссылку на эту область, — аналогично тому, как иногда «обозначают» зрителя: «человек с 5-го ряда, 3-го места». Как мы увидим ниже, такой вид доступа позволяет динамически захватывать и освобождать память в процессе работы блока программы. Поэтому и сами переменные, которые могут создаваться и ликвидироваться по мере надобности, называют *динамическими*.

Остановимся подробнее на работе с динамическими переменными, которые чаще всего реализуются как *связанные структуры*.

Пример 1. Проиллюстрируем особенности такой связанной структуры на примере очереди на прием к врачу. Каждый пациент запоминает человека, за которым занял очередь. Все пациенты связаны в цепочку согласно очереди, но в пространстве они размещены произвольным образом: вновь подошедший садится на любое свободное место, т. е. соседние элементы очереди могут располагаться в пространстве произвольно.

Подобным образом строится структура связанных данных, которые могут занимать память не подряд, а размещаться там, где есть свободное место. Каждый элемент такой структуры должен «знать», за кем он «стоит», т. е. содержать ссылку на предыдущий элемент цепочки.

Чтобы проиллюстрировать преимущества динамических переменных, продолжим аналогию с очередью пациентов.

Пусть один из пациентов покидает очередь. Этот процесс не требует перемещения в пространстве остальных пациентов: просто стоящий за ушедшим теперь запоминает другого человека. То есть исключение элемента из цепочки данных сводится к изменению одной-единственной ссылки и не требует перемещения остальных элементов, как это имело бы место для массива. На рис. 5 показывается исключение элемента цепочки.

Исключенный элемент теперь можно освободить от участия в работе блока и использовать занимаемый им участок памяти для других целей.

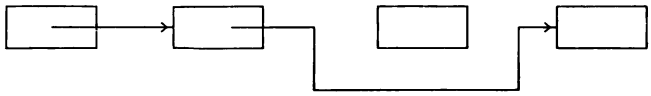


Рис. 5

С помощью ссылок легко вставить новую компоненту в цепочку данных. Для этого достаточно изменить две ссылки (см. рис. 6).

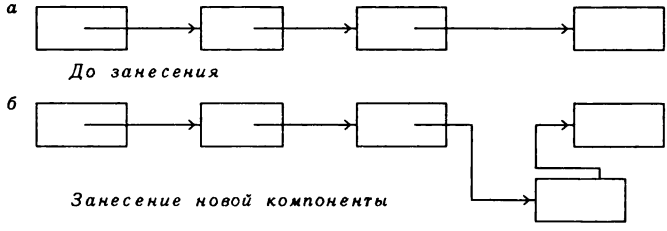


Рис. 6

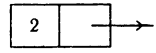
Новая динамическая компонента может быть размещена в любом свободном месте памяти, отведенном под такие переменные. Сама динамическая переменная не обозначается идентификатором. Динамическая переменная — это «невидимка» в программе: идентификатором она не обозначается, транслятор ей место в памяти не отводит. Память под такую переменную резервируется и освобождается динамически в процессе счета (с помощью специальных процедур).

Обращение к динамической переменной происходит посредством *ссылочной переменной*, которая содержит *адрес* соответствующей динамической переменной.

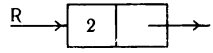
Под ссылочную переменную транслятор отводит место в памяти машины; эта переменная имеет имя и явно упоминается в программе. Ссылочные переменные образуют новый тип данных — «ссылки» (указатели).

Динамические переменные, как правило, имеют тип «запись» (RECORD), так как должны содержать, помимо значения (целого, вещественного и т. п.), ссылку на другую динамическую переменную связанной структуры.

П р и м е р 2. Пусть в памяти машины имеется динамическая переменная, содержащая поле целого значения 2 и поле ссылки (указатель) на другую компоненту связанной структуры (цепочки):



Адрес данной переменной (ссылка) содержится в ссылочной переменной R:



Обозначим тип ссылочной переменной через POINT, а тип динамической переменной — через CT. Тогда этот факт описывается следующим образом:

```
TYPE POINT= ↑CT;
```

Говорят, что «тип POINT указывает (ссылается) на компоненты типа CT» или «тип POINT связан с типом CT».

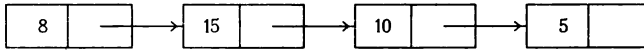
Ссылочную переменную R можно описать двумя способами:

а) TYPE POINT= ↑CT; либо б) VAR R: ↑CT;
VAR R:POINT

Переменная R указывает на компоненты типа CT.

Чтобы связать динамические переменные в цепочку, надо в каждой компоненте иметь ссылку на предыдущую компоненту.

Например, компоненты, содержащие числа 5, 10, 15, 8, должны иметь еще информацию о том, где находится предыдущий элемент, так как это не массив и компоненты размещаются не обязательно подряд.



Опишем тип таких данных, обозначив его CT. Очевидно, этот тип есть «запись» с двумя полями: полем целого значения (I) и полем ссылки (P)

```
TYPE CT=RECORD
 I:INTEGER;
```

```
P:POINT
END;
```

Очевидно, ссылочная переменная, указывающая на такого типа данные, должна иметь тоже тип POINT. Опишем этот тип:

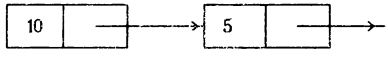
```
TYPE POINT= ↑CT;
```

Как мы видим, возник порочный круг: для описания типа POINT привлекается понятие CT, а при описании типа CT необходимо использовать POINT.

Условились в этом случае *сначала* описывать тип ссылочный переменной, а *затем* уже тип компоненты:

```
TYPE POINT= ↑CT;
 CT=RECORD
   I:INTEGER
 END;
```

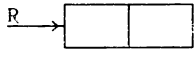
Правила языка Паскаль *только* при *описании ссылок* допускают использование идентификатора (CT) *до* его описания; во всех остальных случаях, прежде чем упомянуть идентификатор, необходимо описать его тип. Рассмотрим схему образования цепочки динамических данных, содержащих числа 5, 10:



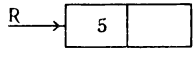
Машине необходимо произвести следующие действия:

1. Найти и зарезервировать место в памяти для компоненты:

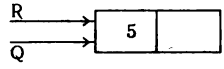
2. Заслать ссылку на эту компоненту (адрес) в ссылочную переменную R:



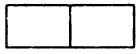
3. Присвоить полю I значение 5:



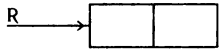
4. Присвоить некоторой ссылочной переменной Q значение R (скопировать):



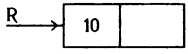
5. Найти и зарезервировать место в памяти для новой компоненты:



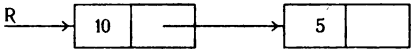
6. Заслать в переменную R адрес этой компоненты:



7. Заслать в поле I значение 10:



8. Заслать в поле P значение Q:



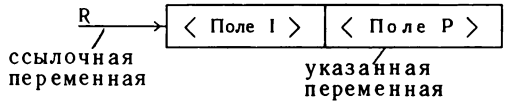
Последовательность подобных действий создает цепочку динамических переменных.

22.1. Процедура NEW. Резервирование места в памяти под динамическую переменную и засылка этого адреса в ссылочную переменную R выполняется при обращении NEW(R). При этом выделяется столько ячеек памяти, сколько требует динамическая переменная, с которой связана R. Эти все данные система получает из раздела описания типов в программе.

Динамические переменные, созданные посредством процедуры NEW(R), называют также *указанными переменными* (указатель R).

П р и м е р. Пусть переменная R имеет тип POINT, описанный выше. Тогда после обращения к процедуре NEW(R) будет создана *указанная* переменная, в которой предусмотрено поле под значение типа INTEGER и поле ссылки. При этом ссылочная переменная R содержит адрес указанной переменной. Через R↑ обозначается сама указанная переменная;

R↑.I — поле целого значения I; R↑.P — поле ссылки P:



22.2. Операции над ссылочными переменными. Значение ссылочной переменной R можно присваивать другой ссылочной переменной того же типа.

П р и м е р 1. Пусть Q,R:↑POINT; тогда оператор Q:=R; зашлет в Q тот же адрес, что хранится в R.

Рассмотрим действия со ссылочными переменными на следующей схеме. Пусть Q и R указывают на различные компоненты динамических переменных типа C:

```
C=RECORD
  I:INTEGER;
  P:POINT
  END;
```

Пусть в памяти машины размещены две цепочки динамических переменных (рис. 7):

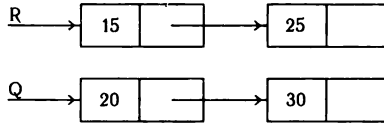
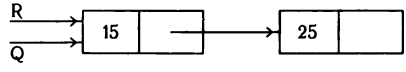


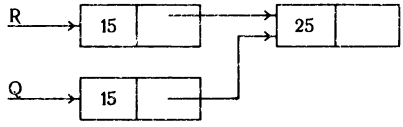
Рис. 7

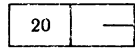
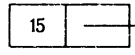
Выполним один из четырех операторов: Q := R; Q↑ := R; Q↑.I := R↑.I; либо Q↑.P := R↑.P;

а) После выполнения оператора Q := R; переменная Q указывает на ту же динамическую переменную, что и R:

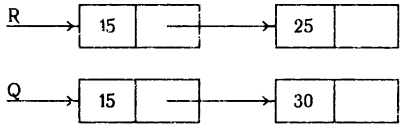


б) После выполнения оператора Q↑ := R↑; (из исходного состояния (рис. 7)) получим



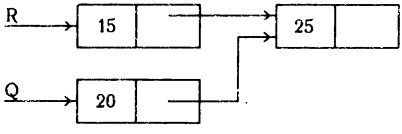
На месте указанной переменной  , указывавшей на 30, заслана переменная  , указывающая на 25.

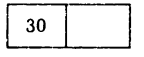
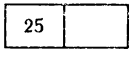
в) После выполнения оператора Q↑.I := R↑.I; из исходного состояния (рис. 7) получим следующее:



На место целого значения 20 заслано значение 15; поле указателя не изменилось.

г) После выполнения оператора Q↑.P := P↑.P; из исходного состояния (рис. 7) получим:



На место ссылки на компоненту  заслана ссылка на компоненту  , поле целого значения не изменилось.

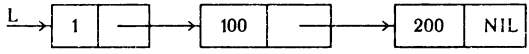
Ссылочные переменные могут указывать на одну и ту же переменную, т. е. быть равными, как R и Q в случае а).

Ссылочные переменные можно сравнивать посредством операций = и < >. Логическое выражение Q = R имеет значение TRUE для случая а) и значение FALSE для случаев б) и в), так как для б) ссылочные переменные Q и R указывают на

разные динамические переменные, имеющие, правда, равные значения.

В качестве аналога нуля для ссылочных переменных принято специальное значение NIL: если переменная имеет значение NIL, то это означает, что она не указывает ни на какую переменную. Значение NIL в поле указателя имеет всегда *первая* компонента цепочки динамических переменных.

Пример 2.



Значение NIL можно заслать оператором присваивания: L := NIL; если L = NIL, то цепочка пуста.

Чтобы определить, что данный элемент является первым в цепочке переменных, достаточно проверить на NIL значение поля указателя этой переменной.

Пример 3. IF L↑.P=NIL THEN...

Замечание. Попытка обратиться к указанной переменной с указателем, имеющим значение NIL, приводит к ошибке. Диагностика в этом случае не всегда выдается.

22.3. Процедура DISPOSE. Динамическая переменная, созданная процедурой NEW, может быть «стерта» только процедурой DISPOSE.

Общий вид:

DISPOSE(R);

Здесь R — ссылочная переменная, указывающая на ту динамическую переменную, которую следует стереть. После стирания динамической переменной R↑ нельзя использовать значение R, такая ошибка может привести к порче памяти и другим серьезным последствиям.

Динамические переменные, не стертые посредством DISPOSE, продолжают занимать место в памяти после окончания работы соответствующего блока программы (становятся «мусором»). Поэтому необходимо все лишние динамические переменные стереть перед окончанием работы блока.

22.4. Стек («магазин»). Начнем с рассмотрения примера. Пусть в трубку с запаянным концом закатывают шарики

(рис. 8). Извлекать их можно только в обратном порядке: тот шарик, что закатился последним, будет извлечен первым.

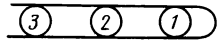


Рис. 8

Подобным образом можно организовать и данные. *Стек* — такая структура динамических данных, которая состоит из переменного числа компонент одинакового типа. Компоненты извлекаются из стека таким образом, что *первой* выбирается та компонента, которая была помещена *последней*. Извлеченная компонента в стеке не сохраняется.

П р и м е р. Рассмотрим последовательные этапы засылки в стек чисел 1, 2, 3 (рис. 9).

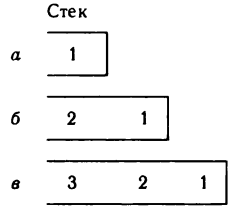


Рис. 9

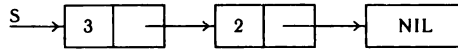
На этапе *б* обращение к процедуре извлечения из стека дает число 2, на этапе *в* — число 3.

Опишем стек, в который можно помещать цепочку динамических переменных:

```
TYPE STACKP=↑STACKCOMP;
STACKCOMP=RECORD
  I:INTEGER;
  P:STACKP
  END;
VAR S:STACKP;
```

Если поместить в этот стек последовательно числа 1, 2, 3, то получится следующий вид:

1.

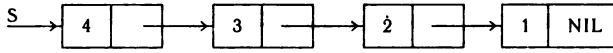


Поместить в такой стек компоненту можно, например, процедурой IS:

```
S:=NIL;
. . .
PROCEDURE IS(K:INTEGER);
VAR INEW:STACKP;
(* ТИП STACK ДОЛЖЕН БЫТЬ ОПИСАН ВЫШЕ,
НАПРИМЕР В PROGRAM *)
BEGIN
(* РЕЗЕРВИРУЕТСЯ ПАМЯТЬ ПОД НОВУЮ КОМПОНЕНТУ,
   А В INEW ЗАСЫЛАЕТСЯ АДРЕС ЭТОЙ КОМПОНЕНТЫ *)
NEW (INEW);
 WITH INEW DO
  BEGIN I:=K; P:=S END;
 S:=INEW
END
```

Если со стеком вида 1 обратиться к процедуре IS для засылки числа 4, то получим стек вида 2:

2.



Процедура извлечения компоненты из такого стека может иметь следующий вид:

```
PROCEDURE OUT(VAR K:INTEGER);
VAR IOLD:STACKP;
BEGIN
 IOLD:=S;
(* АДРЕС ПОСЛЕДНЕЙ КОМПОНЕНТЫ *)
 K:=IOLD↑.I;
(* ИЗВЛЕКАЕТСЯ И ЗАСЫЛАЕТСЯ В S ЗНАЧЕНИЕ
СООТВЕТСТВУЮЩЕГО УКАЗАТЕЛЯ НА 3 *)
 S:=IOLD↑.P;
 DISPOSE (IOLD)
END;
```

После обращения к процедуре OUT стек вернется к виду 1.

Пустым стеком называется стек, не содержащий компонент. Такой стек можно получить, присвоив значение NIL соответствующей ссылочной переменной (в нашем случае: S:=NIL;).

Если к пустому стеку применить несколько раз процедуру IS, а затем *столько же* раз процедуру OUT, то получим снова пустой стек.

З а м е ч а н и е. Нельзя применять процедуру OUT к пустому стеку.

Стеки позволяют гибко и экономно использовать память, так как в каждый момент в стеке могут находиться только те переменные, которые нужны для дальнейшей работы программы. (В то время как под массивы, например, мы часто вынуждены резервировать и держать избыточную память.)

22.5. Очередь. *Очередь* — такая структура данных, при которой *изъятие* компонент происходит *из начала* цепочки, а *запись* — *в конец* цепочки.

В этом случае вводят два указателя: один на начало очереди, другой — на ее конец.

22.6. Дозапись новых компонент.

П р и м е р. Пусть имеется цепочка динамических переменных (рис. 10).

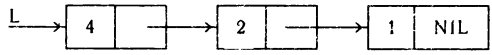
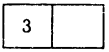
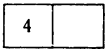
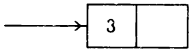


Рис. 10

Переменные имеют описанный выше тип STACKCOMP.

Требуется вставить в цепочку новую компоненту 

после компоненты 

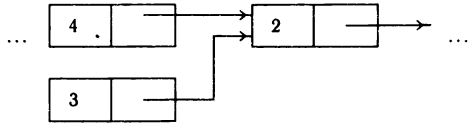
, если известен указатель NEWP 

.

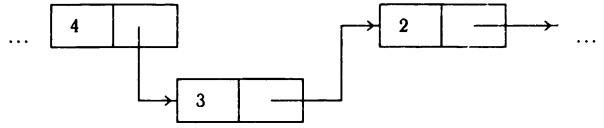
Для записи этой новой компоненты достаточно выполнить операторы:

```
NEWP↑.P:=L↑.P;
L↑.P:=NEWP;
```

Первый оператор засылает в поле указателя новой компоненты | 3 | <NEWP↑.P> | ссылку на компоненту | 2 | | . Эта ссылка находится в поле указателя последней компоненты | 4 | <L↑.P> | , т. е. получается следующий вид:

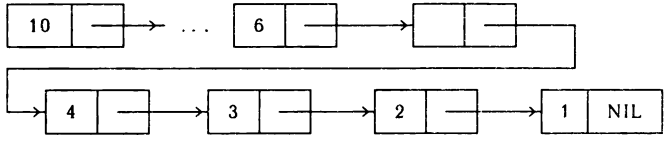


Второй оператор помещает в поле указателя компоненты | 4 | | ссылку на компоненту | 3 | | . Получается следующая картинка:



З а д а ч а. Построить цепочку динамических переменных, содержащих целые числа, а затем между 4-й и 5-й переменными вставить новую динамическую переменную.

Пусть требуется построить такую цепочку:



и вставить элемент | 11 | | между | 4 | | и | 5 | | .

Р е ш е н и е.

```
PROGRAM POINT (INPUT,OUTPUT);
TYPE INTP=↑INTREC;
 INTREC=RECORD
  I:INTEGER;
  P.INTP
```

```
    END;
  VAR IP,IR,INSERTI, INSERT4,INSERT5,WRTP:INTP;
    N,K,L,M:INTEGER;
  BEGIN
    IR:=NIL;N:=10;
    FOR K:=1 TO N DO
    BEGIN READ(L); WRITE (' ',L);
      NEW (IP); IP↑.I:=L; IP↑.P:=IR; IR:=IP;
      IF K=5 THEN
      BEGIN INSERT5:=IP; INSERT4:=IP↑.P END
    END;
  WRITELN;
  READ(L); WRITELN(' L=',L);
  NEW(INSERTI); INSERTI↑.I:=L;
  INSERTI↑.P:=INSERT4;
  INSERT5↑.P:=INSERTI;
  FOR K:=1 TO N + 1 DO
    BEGIN WRTP:=IR; M:=WRTP↑.I; WRITE (' ',M);
      IR:=WRTP↑.P;
      DISPOSE(WRTP)
    END;
  WRITELN
  END.
```

Результат:

```
  1 2 3 4 5 6 7 8 . . . 10
L = 11
    10 9 8 7 6 5 11 4 . . . 1
     9
     8
     7
     6
     5
    11
     4
     3
     2
     1
```

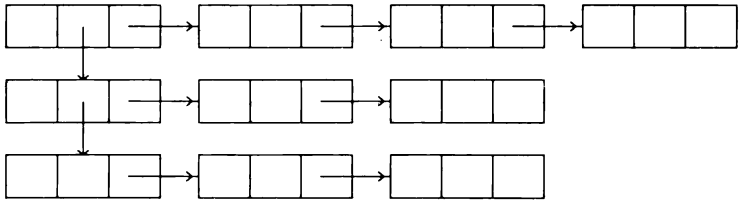
Вводятся числа: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 и 11. Между 4 и 5 вставляется число 11.

22.7. Нелинейные структуры. До сих пор мы рассматривали *линейные* структуры динамических переменных.

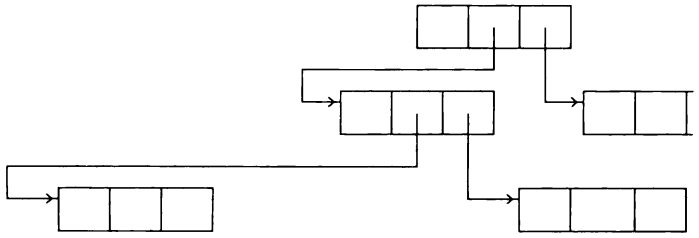
Введение в динамическую переменную двух и более полей указателей создает возможность получать *нелинейные структуры*.

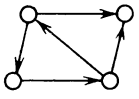
Примеры нелинейных структур показаны ниже.

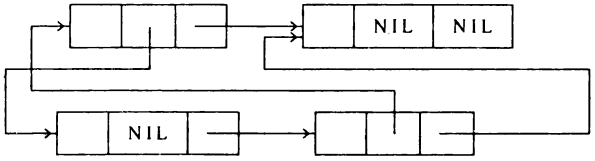
а) Текст:



б) Двоичное дерево можно представить так:



в) Направленный граф:  можно представить:



Остановимся подробнее на этих примерах.

а) Текст — это связанная структура. Ее элементы представляют собой записи с тремя полями: первое поле содержит текстовую информацию (например слово), второе поле либо указывает на первый элемент следующей строки, либо имеет значение NIL, третье указывает на следующий элемент данной строки.

б) Компонента двоичного дерева также имеет три поля: первое поле содержит основную информацию (число, символ, слово и т. д.), а второе и третье поля содержат ссылки на предыдущую и последующую компоненты дерева. Для некоторых элементов дерева одно из этих полей либо оба имеют значение NIL.

в) Посредством ссылок можно выразить все связи в структуре, моделирующей направленный граф: вершинам графа соответствуют динамические переменные, а ребрам — ссылки. *Узлом* называют переменную, содержащую два различных, отличных от NIL, значения указателя. Так, узловые элементы текста содержат ссылки на очередной элемент данной строки и на первый элемент следующей строки.

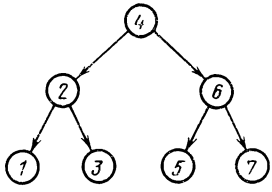


Рис.11

Нелинейные структуры удобны для задач поиска. Список упорядоченных элементов в виде двоичного дерева представлен на рис. 11. Узел 4 называется *корнем* дерева. Вход в дерево происходит только через корень. Для каждого узла различаются левое и правое *поддеревья*. Упорядоченность элементов следующая: элементы левого поддерева меньше узла и меньше элементов правого поддерева.

Время поиска в двоичном дереве сокращается по сравнению с линейной структурой с ~N до ~$\log_2$N, где N — число узлов в дереве. Компоненту двоичного дерева можно представить переменной типа

```
BIREC=RECORD
  I:INTEGER;
```

```
PRED,SUCC:INTP
END;
```

Каждая такая переменная содержит три поля: поле целого значения — поле I, поле указателя на предыдущий (в смысле упорядоченности) элемент — поле PRED и поле указателя на последующий элемент — поле SUCC.

23. Работа с внешними модулями*)

Паскаль позволяет работать с внешними процедурами (функциями), которые существуют вне главной программы (PROGRAM).

Если модуль является стандартным (библиотечным), то никаких описаний его в программе не требуется.

В остальных случаях внешний модуль должен быть описан в PROGRAM следующим образом.

1. Процедура Паскаля:

```
PROCEDURE N(P1:T1;...);EXTERNAL;
```

Здесь N — имя процедуры, P1,... — формальные параметры, T1,... — типы формальных параметров (напоминаем, что параметру, предназначенному для результата, должно предшествовать ключевое слово VAR).

2. Функция Паскаля:

```
FUNCTION F(X:TYPEX;...):TYPEF; EXTERNAL;
```

Здесь F — имя функции, X — формальный параметр, TYPEX — тип этого параметра, TYPEF — тип результата, EXTERNAL(EXTERN) — указание на то, что эта функция существует вне данной программы.

З а м е ч а н и е. На БЭСМ-6 следует писать EXTERNAL, на ЕС ЭВМ — EXTERN.

3. Подпрограмма Фортрана:

```
PROCEDURE SUB (X:TYPEX;...); FORTRAN;
```

Здесь X — параметр, TYPEX — его тип, FORTRAN — указание на то, что использован фортранный модуль.

---

*) Работа с внешними модулями — расширение стандарта языка Паскаль.

4. Подпрограмма-функция Фортрана:

FUNCTION F(X:TYPEX;...):TYPEF; FORTRAN;

Ниже приводится пример, который поможет читателю использовать внешние модули в своих программах.

П р и м е р (ЕС ЭВМ).

```
//TEXT5 JOB C3746, SEMASHKO,MSGLEVEL=(2,0)
// EXEC FORTGC
//FORT. SYSIN DD *
     SUBROUTINE SUB(X)
C ТРАНСЛЯЦИЯ SUB
    X=2.
    RETURN
    END
    FUNCTION F(X)
C ТРАНСЛЯЦИЯ F
    F=4*X
    RETURN
    END
// EXEC PASC
//PASC.SYSIN DD *
PROGRAM FILLER (OUTPUT);
(* ТРАНСЛЯЦИЯ PROC *)
(* $E+ *)
PROCEDURE PROC (VAR X:REAL);
 BEGIN X:=1.0 END;
BEGIN END.
// EXEC PASC
//PASC.SYSIN DD *
PROGRAM FILLE(OUTPUT);
(* ТРАНСЛЯЦИЯ FUNPSCL *)
(* $E+ *)
FUNCTION FUNPSCL(X:REAL):REAL;
 BEGIN FUNPSCL:=3.0 END;
BEGIN END.
// EXEC PASCLG,
//PASC.SYSIN DD *
PROGRAM GETHER (OUTPUT);
```

```
(* ТЕСТ НА ВНЕШНИЕ МОДУЛИ *)
VAR A,B:REAL;
 PROCEDURE SUB(VAR X:REAL); FORTRAN;
 FUNCTION F(X:REAL):REAL; FORTRAN;
 FUNCTION FUNPSCL(X:REAL):REAL; EXTERN;
 PROCEDURE PROC (VAR X:REAL); EXTERN;
BEGIN A:=1.0;
 SUB(B); WRITE(' SUB(B):',B,'F(A)=',F(A));
 A:=FUNPSCL(A); WRITELN(' A=',A);
 PROC(B);WRITELN(' PROC(B):',
 B,'FUNPSCL(A)=',A)
END.
//LKED.SYSLIB DD
//DD DSN=SYS1.FORTLIB,DISP=SHR
//GO.SYSIN DD *
//
```

Результат:

```
SUB(B):2,000...F(A)=4.000...A=3.000...
PROC(B):1.000...FUNPSCL(A)=3.000...
```

В этом примере выполняется программа GETHER, использующая внешние фортранные модули SUB и F, а также модули Паскаля PROC и FUNPSCL, описанные вне программы GETHER.

Внешние процедуры и функции не должны иметь в качестве параметров параметры-функции и сами не могут быть переданы в качестве параметров.

Внешняя процедура может использоваться рекурсивно и передана как параметр только процедуре, декларированной *внутри* нее самой.

23.1. Трансляция внешних модулей. Необходимо соблюдать следующие правила.

1. Внешние Паскаль-процедуры и функции транслируются только в режиме E+ (см. п. 24).

2. Режим E+ должен быть установлен до декларации внешней процедуры.

В этом режиме не должны транслироваться внутренние процедуры и главная программа.

3. Допускается только два уровня вложения внешних процедур. Однако сама внешняя процедура может иметь несколько уровней вложения внутренних (для нее) процедур.

4. Если во внешней процедуре описываются глобальные переменные, константы или типы, то они должны в точности совпадать по описанию с соответствующими переменными, константами, типами в PROGRAM.

5. Если однажды установлен режим E+, то нельзя его устанавливать повторно.

Когда Паскаль-программа вызывает фортранную подпрограмму или функцию, надо учитывать следующее.

1. Тип REAL на ЕС ЭВМ соответствует фортранному DOUBLE PRECISION (или REAL*8).

2. Тип BOOLEAN соответствует фортранному LOGICAL.

3. Массивы Паскаля располагаются по строкам, а массивы Фортрана — по столбцам.

З а м е ч а н и е. При передаче двумерного массива в качестве параметра из Паскаль-процедуры в фортранную (и обратно) этот массив необходимо транспонировать.

24. Режимы трансляции

Режимы трансляции задаются комментариями особого вида (псевдокомментариями):

(*CH R1,R2,... *)

Здесь (* и *) — ограничители, CH — специальный символ, превращающий комментарий в управляющую карту, R1, R2 — задаваемые коды режимов трансляции. На БЭСМ-6 CH — это символ =, на ЕС ЭВМ CH — это символ $ (либо совпадающий с ним по кодировке знак).

Каждый код режима состоит из двух символов: буквы, за которой следует либо знак (+ или –), либо цифра.

Знак + означает включение данного режима, знак – означает отказ от него.

П р и м е р. (*=T+,E+,P– *) либо (*$T+, E+, P– *)

Чаще всего используются следующие режимы:

1. T — этот режим обеспечивает динамические проверки во время счета:

а) всех операций с индексными переменными на принадлежность каждого индекса допустимому диапазону индексов;

б) всех операторов присваивания на принадлежность значений переменных ограниченного типа соответствующему подмножеству:

в) всех делителей в операциях деления (на нуль);

г) всех автоматических преобразований

INTEGER → REAL на ABS(I)<=MAXINT;

д) всех операторов CASE на соответствие переключателя одной из меток CASE.

По умолчанию установлен T+.

Для отлаженных программ рекомендуется использовать T–. Это ускоряет выполнение программы.

2. P позволяет выдавать подробную информацию при «авостах» (аварийных остановах — прекращении счета при ошибке). В режиме P+ выдаются значения локальных переменных, идентификаторы вызванных процедур (функций) и номера строк программы, в которых начинаются соответствующие составные операторы.

По умолчанию установлен P+.

Для отлаженных программ следует указывать режим P–, что экономит память и время выполнения программы.

3. E позволяет так транслировать процедуры и функции, что к ним можно обращаться из других программ как к внешним модулям.

Если данная процедура используется как EXTERNAL, то ее необходимо транслировать только в режиме E+.

По умолчанию установлен E–.

4. U+ — все символы входной строки, начиная с 73-го, считаются комментариями. На БЭСМ-6 режима U нет. Если используется U–, то все символы, начиная со 121-го, считаются комментариями.

5. BN — для БЭСМ-6. Пусть S — нижняя граница размера памяти, выделенной под буфер файлов, S>256*N.

По умолчанию установлено N=1.

B+ для ЕС ЭВМ — зарезервированные Паскалем ключевые слова (AND, ARRAY, ..., WITH) на листинге АЦПУ печатаются жирно.

По умолчанию установлен B–.

6. L управляет информацией об исходной программе, выдаваемой на АЦПУ.

На ЕС ЭВМ:

L+ — подробная выдача,

L– — режим счета, подавление листинга программы.

По умолчанию установлен L+.

На БЭСМ-6:

L0 — выдаются только сообщения об ошибках (*NO LIST);

L1 — выдается таблица загрузки и текст программы;

L2 — дополнительно к информации, выдаваемой по L1, выдаются коды стандартного массива (см. [7]).

По умолчанию установлен L1.

ПРИЛОЖЕНИЕ 1

ПРОГРАММА ИЗМЕНЕНИЯ ДЛИНЫ СТРОК ТЕКСТА

(пример использования записей с вариантами)

З а д а ч а. Составить программу, вводящую текстовую информацию со строками длиной до 80 символов и выдающую этот же текст со строками длиной до 60 символов (без разбиения слов при переносе).

В тексте могут присутствовать пустые строки; абзац начинается с нескольких пробелов. Абзацем будем считать любую непустую строку, имеющую три и более пробелов в начале. Конец всего текста пусть будет отмечен специальным символом EOT, конец строки — символом EOL.

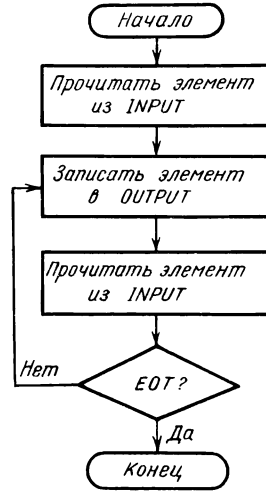


Рис. 12

Тогда любой текст можно представить в виде последовательности элементов: слов, знаков препинания и символов EOL и EOT. Все элементы разделяются пробелами. Простейшая схема программы приводится на рис. 12.

Напомним, что слова не разбиваются для переноса: если слово умещается, оно записывается в данную выходную строку; если не умещается, то слово целиком записывается в следующую строку OUTPUT. Элемент «слово» и элемент «знак» неравнозначны: знак обязательно должен стоять в той же строке, что и слово, за которым он поставлен, а последовательные слова можно писать в разных строках.

Поэтому программа после ввода каждого элемента должна ввести еще один элемент и только после его анализа обрабатывать предыдущий (оставлять на текущей строке либо переносить на следующую).

Усложненная схема программы, имя которой INOUT, приводится на рис. 13.

Пусть чтение элемента выполняется процедурой RNEXT и запись — процедурой WTHIS.

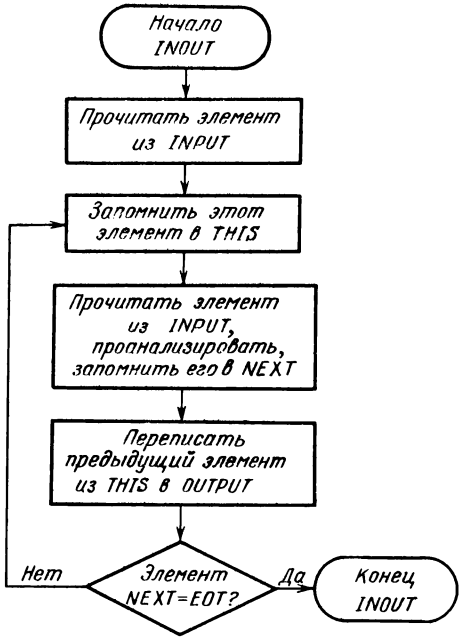


Рис. 13

Если в исходном тексте есть абзац или пустая строка, то будем считать, что в соответствующем месте текста есть некоторый элемент, играющий роль управляющего символа (CONTR).

Тогда весь текст можно представить состоящим из следующих элементов: слов (WORD), знаков препинания (PUNCT), уп-

равляющих символов (CONTR) и признака «конец текста» (EOT).

Будем считать, что слово состоит не более чем из 16 символов.

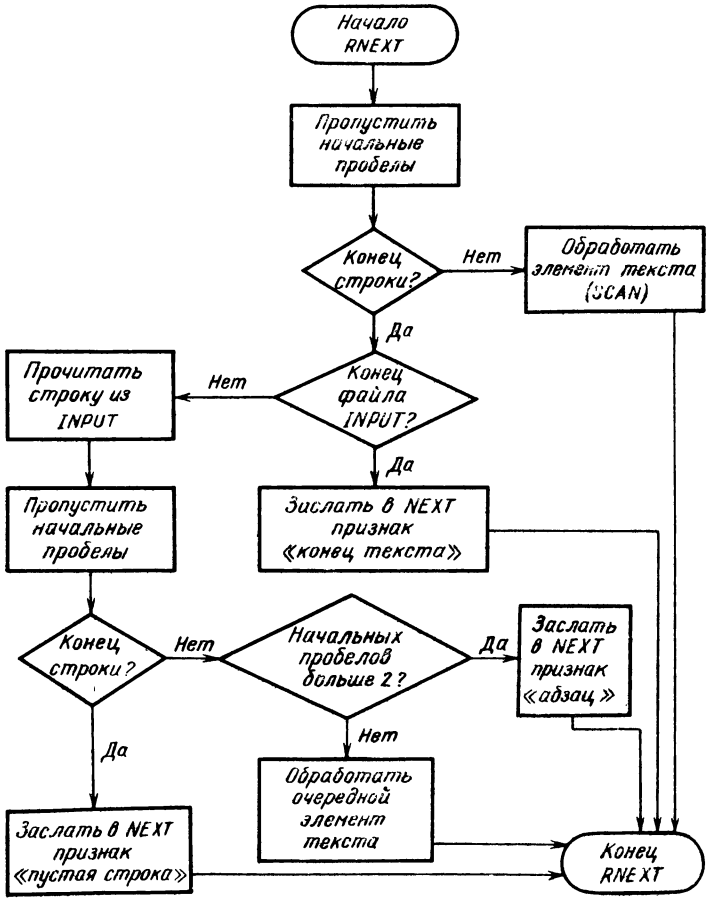


Рис. 14

В этих предположениях произвольный элемент (ITEM) текста описывается как запись с вариантами:

TYPE ITEM=(WORD, PUNCT, CONTR, EOT);

```
TEXTITEM=RECORD
 CASE KIND:ITEM OF
 WORD:(L:1..16;
  SP:ARRAY[1..16] OF CHAR);
 PUNCT:(P: CHAR);
 CONTR:(C:(JUMP,ABZ))
 EOT:( )
 END;
```

Если текущий элемент есть «слово» (WORD), то задается его длина L и массив SP на 16 символов, где размещаются буквы этого слова; если элемент — «знак» (PUNCT), то в P находится соответствующий символ; если элемент — «управляющий символ» (CONTR), то в C находится признак «пустая строка» (JUMP) либо «абзац» (ABZ). Символ «конец текста» (EOT) служит только признаком конца и в выходной текст не попадает.

Введем две переменные THIS и NEXT для хранения элемента текста

```
VAR THIS,NEXT:TEXTITEM;
```

Процедура RNEXT должна прочитать символы, составляющие следующий элемент входного текста, проанализировать и поместить информацию в переменную NEXT, т. е. заполнить соответствующее поле записи типа TEXTITEM для переменной NEXT.

Удобнее вводить из INPUT не по одному элементу текста, а по целой строке. Для этого заведем массив LINE длиной от 1 до INMAX символов по числу символов во входной строке:

```
LINE:ARRAY [1..INMAX] OF CHAR;
```

В нашем конкретном случае входная строка содержит до 80 символов. Для упрощения алгоритма программы будем отмечать конец входной строки каким-либо специальным символом, например !. Этот символ поместим в LINE после последнего отличного от пробела символа введенной строки. Таким образом, длина массива LINE должна быть на 1 больше длины входной строки, т. е. для входной строки в 80 символов INMAX=81.

Элементы строки процедура выбирает из массива LINE. Схема RNEXT приводится на рис. 14.

Задачу «обработать элемент текста» можно оформить в виде процедуры SCAN (рис. 15).

На последнем этапе работы программы потребуется формировать выходные строки текста в файле OUTPUT. Запись элемента текста в выходную строку можно осуществить следующей процедурой WTHIS (рис. 16).

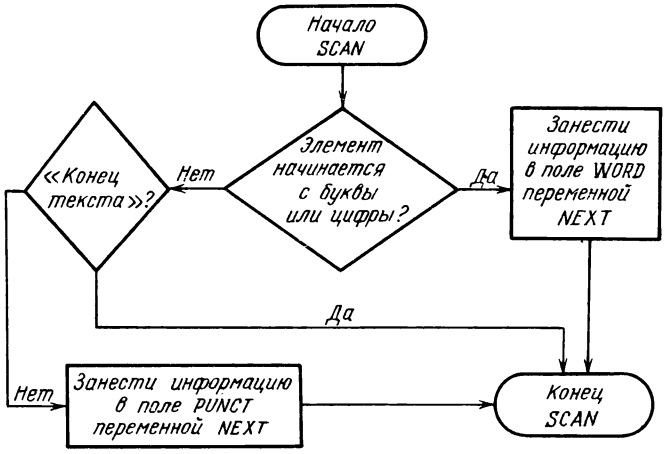


Рис. 15

Приведем полный текст программы. Здесь:

INMAX — максимальная длина входной строки плюс единица,

OUTMAX — максимальная длина выходной строки,

EOT — признак конца текста,

EOL — признак конца строки,

LLINE — длина текущей входной строки,

THIS — предыдущий элемент текста,

NEXT — последующий элемент текста,

LINE — массив для хранения входной строки,

POS — номер текущей позиции в строке,

FREE — количество оставшихся свободных позиций в строке,

K — номер символа в слове,

SP — массив для хранения слова,

SPACE — число позиций, требуемое под очередное слово в выходной строке.

```
   PROGRAM INOUT (INPUT,OUTPUT);
   (* ПРЕОБРАЗОВАНИЕ 80-СИМВОЛЬНОЙ СТРОКИ
   В 60-СИМВОЛЬНУЮ *)
CONST INMAX=81; EOL='!'; EOT='*';
   OUTMAX=60;
TYPE ITEM=(WORD,PUNCT,CONTR,EOT);
   TEXTITEM=RECORD
    CASE KIND:ITEM OF
     WORD:(L:1..16; SP:ARRAY (.1..16.) OF CHAR);
     PUNCT:(P:CHAR);
     CONTR:(C:(JUMP,ABZ));
     EOT:( )
    END;
   LLINE=1..INMAX; (* ДЛИНА ВХОДНОЙ СТРОКИ *)
VAR THIS, NEXT:TEXTITEM;
   LINE:ARRAY(.LLINE.) OF CHAR;
   POS:LLINE; (*НОМЕР ПОЗИЦИИ В СТРОКЕ *)
   FREE:0..OUTMAX;
PROCEDURE RNEXT;
   (* ЧТЕНИЕ ЭЛЕМЕНТА ИЗ LINE *)
PROCEDURE RLINE;
   (* ВВОД СТРОКИ ИЗ INPUT *)
VAR M:LLINE;
BEGIN (* RLINE *)
   M:=1;
   WHILE NOT EOLN(INPUT) DO
    BEGIN READ(LINE(.M.));M:=M+1 END;
    READLN;
    LINE(.M.):=EOL; POS:=1
END; (* RLINE *)
PROCEDURE SCAN;
```

```
   (* ОБРАБОТКА ЭЛЕМЕНТОВ — 'СЛОВО' И
   'ЗНАК ПРЕПИНАНИЯ' *)
VAR K:0..16;
   ST:SET OF CHAR;
BEGIN (* SCAN *)
   ST:=['.',',',':',';',' '.];
   WITH NEXT DO
   CASE LINE (.POS.) OF
    'А','Б','В','Г','Д','Е','Ж','З',
    'К','Л','М','Н','О','П','Р','С',
    'У','Ф','Х','Ц','Ч','Ш','Щ','''','Ы',
    'Э','Ю','Я','И','Т','Ь','Ъ',
    '0','1,'2','I','D','3','F','G','4',
    'J','5','L','6','N','7','8','Q',
    'S','9','U','V','W','-','Y','Z':
      BEGIN (* НАЧИНАЕТСЯ С БУКВЫ *)
       KIND:=WORD; K:=0;
       REPEAT K:=K+1;
        SP(.K.):=LINE(.POS.);
        POS:=POS+1
       UNTIL (LINE(.POS.) IN ST);
       L:=K
      END;
    '.',',',':',';':
      BEGIN (* ЗНАК ПРЕПИНАНИЯ *)
       KIND:=PUNCT;
       P:=LINE(.POS.);
       POS:=POS+1
      END;
    '*':KIND:=EOT (* КОНЕЦ ТЕКСТА *)
   END (* CASE *)
END; (* SCAN *)
BEGIN (* RNEXT *)
   WHILE LINE(.POS.)=' ' DO POS:=POS+1;
   IF LINE (.POS.)=EOL
   THEN IF EOF (INPUT)
        THEN NEXT.KIND:=EOT
        ELSE BEGIN RLINE;
```

```
            WHILE LINE (.POS.)=' '
            DO POS:=POS+1;
            WITH NEXT DO
             IF LINE(.POS.)=EOL
             THEN BEGIN
              KIND:=CONTR;
              C:=JUMP
              END
             ELSE IF POS > 2
              THEN BEGIN
               KIND:=CONTR;
               C:=ABZ
               END
             ELSE SCAN
        END
   ELSE SCAN
END; (* RNEXT *)
PROCEDURE WTHIS;
(* ЗАПИСЬ THIS В OUTPUT *)
VAR SPACE:1..18;
   M:1..16;
   PROCEDURE NEWL;
   (* ПЕРЕХОД К НОВОЙ ВЫХ. СТРОКЕ *)
   BEGIN
    WRITELN;
    WRITE(' ');
    FREE:=OUTMAX-1
   END; (* NEWL *)
BEGIN (* WTHIS *)
   WITH THIS DO
    CASE KIND OF
      WORD:BEGIN (* 'СЛОВО' *)
       IF NEXT.KIND=PUNCT
       THEN SPACE:=L+2
       ELSE SPACE:=L+1;
       IF SPACE > FREE
       THEN NEWL
       ELSE IF FREE < > OUTMAX
```

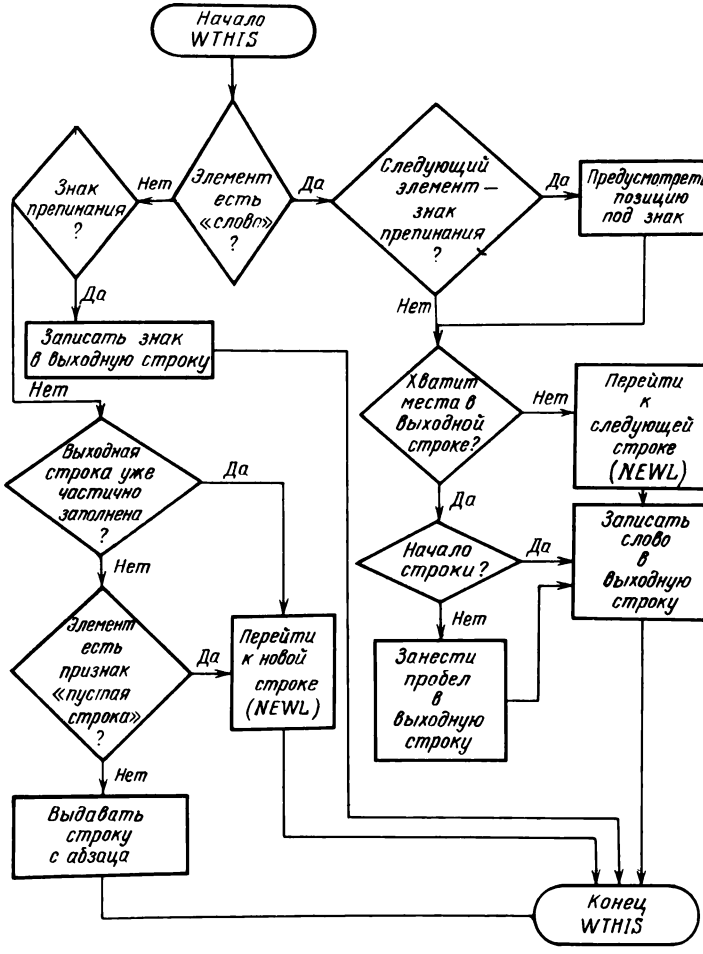


Рис. 16

```
  THEN BEGIN
  WRITE (' ');
  FREE:=FREE-1
END;
```

```
        FOR M:=1 TO L
          DO WRITE (SP(.M.));
        FREE:=FREE-L
        END;
      PUNCT:BEGIN (* ЗНАК ПРЕПИНАНИЯ *)
        WRITE(P); FREE:=FREE-1
        END;
      CONTR:BEGIN (* 'ПРОПУСК СТРОКИ' ЛИБО 'АБЗАЦ' *)
        IF FREE < OUTMAX
        THEN NEWL;
        ELSE BEGIN
        WRITE (' ');
        FREE:=FREE-2
        END
      END;
     EOT:BEGIN (* КОНЕЦ ТЕКСТА *)
      FOR M:=1 TO L DO
        WRITE (SP(.M.));
      FREE:=FREE-L
      END
    END (* CASE *)
END; (* WTHIS *)
BEGIN (* INOUT *)
  LINE(.1.):=EOL; POS:=1;
  FREE:=OUTMAX;
  RNEXT;
  WRITE (' '); FREE:=FREE-1;
  REPEAT
   THIS:=NEXT; (* ПЕРЕПИСЬ ЭЛЕМЕНТА
                  ТЕКСТА В THIS *)
   RNEXT;
   WTHIS (* ВЫДАЧА THIS В ВЫХ. СТРОКУ *)
  UNTIL NEXT.KIND=EOT;
END. (* INOUT *)
```

ПРИЛОЖЕНИЕ 2

ПРОГРАММА РЕШЕНИЯ ЗАДАЧИ О ХАНОЙСКОЙ БАШНЕ

(пример использования рекурсивной процедуры)

В одной из древних легенд говорится следующее. «В храме Бенареса находится бронзовая плита с тремя алмазными стержнями. На один из стержней бог при сотворении мира нанизал 64 диска разного диаметра из чистого золота так, что наибольший диск лежит на бронзовой плите, а остальные образуют пирамиду, сужающуюся кверху. Это — башня Брамы. Работая день и ночь, жрецы переносят диски с одного стержня на другой, следуя законам Брамы:

1) диски можно перемещать с одного стержня на другой только по одному;

2) нельзя класть больший диск на меньший.

Когда все 64 диска будут перенесены с одного стержня на другой, и башня, и храмы, и жрецы-брамины превратятся в прах и наступит конец света».

Эта древняя легенда породила задачу о Ханойской башне: переместить m дисков с одного из трех стержней на другой, соблюдая «законы Брамы».

Назовем стержни левым (LEFT), средним (MIDDLE) и правым (RIGHT). Задача состоит в переносе m дисков с левого стержня на правый (рис. 17).

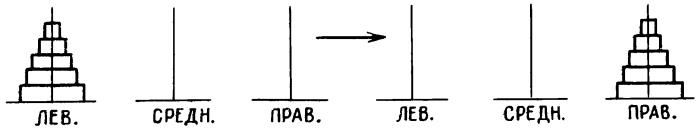


Рис. 17

Задача может быть решена одним перемещением только для одного (m = 1) диска. В общем случае потребуется 2^{m-1} перемещений.

Построим рекурсивное решение задачи, состоящее из трех этапов:

а) перенести башню, состоящую из m – 1 диска, с левого стержня на средний (рис. 18).

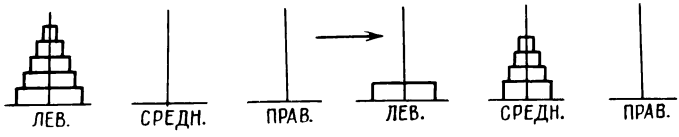


Рис. 18

б) перенести один оставшийся диск с левого стержня на правый (рис. 19):

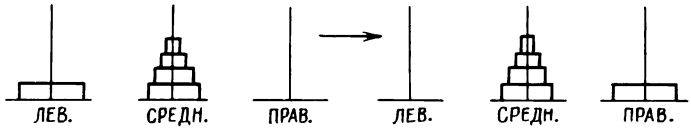


Рис. 19

в) перенести башню, состоящую из m – 1 диска, со среднего стержня на правый (рис. 20):

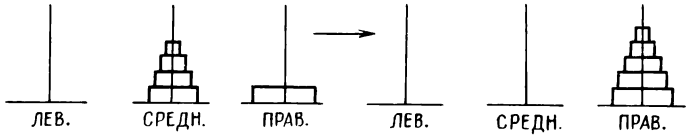


Рис. 20

Таким образом, задача о перемещении m дисков сводится к задаче о перемещении m – 1 диска. Обращаясь опять к этому же алгоритму, сведем задачу к перемещению m – 2 дисков. Продолжая этот процесс, получим в конце концов задачу о

перемещении одного диска. Эта задача решается за один ход. Таким образом, в процессе решения возникают промежуточные задачи: переместить башню из нескольких (n) дисков с одного стержня на другой.

Обозначим тот стержень, с которого следует снять диски, через S_1, тот, на который надо надеть — через S_K, а вспомогательный стержень — через S_W.

Оформим алгоритм решения задачи о переносе башни из n дисков с S_1 на S_K в виде процедуры MOVE с четырьмя параметрами: N, S1, SW, SK; алгоритм для n = 1 выделим в отдельную процедуру STEP, которая «перемещает» один диск со стержня S1 на SK.

Приведем блок-схему задачи о Ханойской башне (рис. 21).

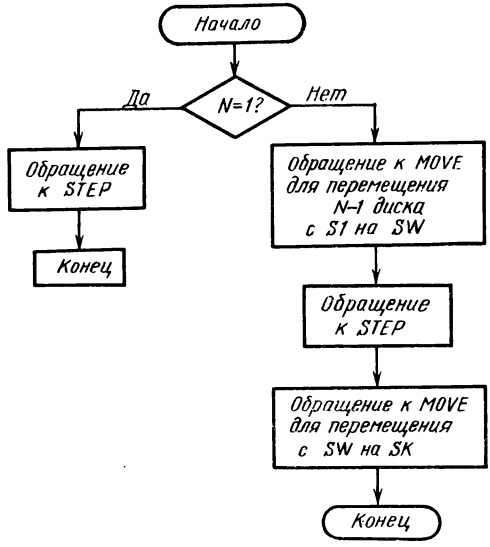


Рис. 21

Ниже приводится программа задачи о Ханойской башне и результат для перемещения пяти дисков.

```
PROGRAM HANOY (INPUT, OUTPUT);
(* ЗАДАЧА О ХАНОЙСКОЙ БАШНЕ *)
```

```
TYPE ST=(LEFT, MIDDLE, RIGHT);
(* NAT – МНОЖЕСТВО НАТУРАЛЬНЫХ ЧИСЕЛ *)
NAT=1..MAXINT;
VAR M:NAT; (* M – ЧИСЛО ДИСКОВ *)
PROCEDURE MOVE (N:NAT; S1, SW, SK:ST);
(* ПЕРЕМЕЩЕНИЕ N ДИСКОВ С S1 НА SK;
SW – ВСПОМОГАТЕЛЬНЫЙ СТЕРЖЕНЬ *)
 PROCEDURE STEP;
(* ПЕРЕМЕЩЕНИЕ ОДНОГО ДИСКА С S1 НА SK *)
     PROCEDURE PRINT (S:ST);
     BEGIN
      CASE S OF
       LEFT:WRITE ('ЛЕВ.');
       MIDDLE:WRITE ('СРЕДН.');
       RIGHT:WRITE ('ПРАВ.')
      END (* CASE *)
     END; (* PRINT *)
BEGIN (* STEP *)
  WRITE (' СНЯТЬ ДИСК С ');
  PRINT(S1);
  WRITE(' НАДЕТЬ НА ')
  PRINT(SK);
  WRITELN
 END; (* STEP *)
BEGIN (* MOVE *)
 IF N=1 THEN STEP
        ELSE BEGIN
          MOVE (N–1, S1, SK, SW);
          STEP;
          MOVE (N–1, SW, S1, SK)
          END
END; (* MOVE *)
BEGIN (* HANOY *)
 READ(M); (* M – ЧИСЛО ДИСКОВ *)
 WRITELN ('ДЛЯ', M:3, 'ДИСКОВ СЛЕДУЕТ',
     'ПРОИЗВЕСТИ СЛЕДУЮЩИЕ ДЕЙСТВИЯ:');
 MOVE (M,LEFT,MIDDLE,RIGHT)
END.
```

Результат работы программы:

```
5

ДЛЯ 5 ДИСКОВ СЛЕДУЕТ ПРОИЗВЕСТИ
СЛЕДУЮЩИЕ ДЕЙСТВИЯ:

СНЯТЬ ДИСК С ЛЕВ. НАДЕТЬ НА ПРАВ.
СНЯТЬ ДИСК С ЛЕВ. НАДЕТЬ НА СРЕДН.
СНЯТЬ ДИСК С ПРАВ. НАДЕТЬ НА СРЕДН.
СНЯТЬ ДИСК С  ЛЕВ. НАДЕТЬ НА ПРАВ.
СНЯТЬ ДИСК С СРЕДН. НАДЕТЬ НА ЛЕВ.
СНЯТЬ ДИСК С СРЕДН. НАДЕТЬ НА ПРАВ.
СНЯТЬ ДИСК С ЛЕВ. НАДЕТЬ НА ПРАВ.
СНЯТЬ ДИСК С ЛЕВ. НАДЕТЬ НА СРЕДН.
СНЯТЬ ДИСК С ПРАВ. НАДЕТЬ НА СРЕДН.
СНЯТЬ ДИСК С ПРАВ. НАДЕТЬ НА ЛЕВ.
СНЯТЬ ДИСК С СРЕДН. НАДЕТЬ НА ЛЕВ.
СНЯТЬ ДИСК С ПРАВ. НАДЕТЬ НА СРЕДН.
СНЯТЬ ДИСК С ЛЕВ. НАДЕТЬ НА ПРАВ.
СНЯТЬ ДИСК С ЛЕВ. НАДЕТЬ НА СРЕДН.
СНЯТЬ ДИСК С ПРАВ. НАДЕТЬ НА СРЕДН.
СНЯТЬ ДИСК С ЛЕВ. НАДЕТЬ НА ПРАВ.
СНЯТЬ ДИСК С СРЕДН. НАДЕТЬ НА ЛЕВ.
СНЯТЬ ДИСК С СРЕДН. НАДЕТЬ НА ПРАВ.
СНЯТЬ ДИСК С ЛЕВ. НАДЕТЬ НА ПРАВ.
СНЯТЬ ДИСК С СРЕДН. НАДЕТЬ НА ЛЕВ.
СНЯТЬ ДИСК С ПРАВ. НАДЕТЬ НА СРЕДН.
СНЯТЬ ДИСК С ПРАВ. НАДЕТЬ НА ЛЕВ.
СНЯТЬ ДИСК С СРЕДН. НАДЕТЬ НА ЛЕВ.
СНЯТЬ ДИСК С СРЕДН. НАДЕТЬ НА ПРАВ.
СНЯТЬ ДИСК С ЛЕВ. НАДЕТЬ НА ПРАВ.
СНЯТЬ ДИСК С ЛЕВ. НАДЕТЬ НА СРЕДН.
СНЯТЬ ДИСК С ПРАВ. НАДЕТЬ НА СРЕДН.
СНЯТЬ ДИСК С ЛЕВ. НАДЕТЬ НА ПРАВ.
СНЯТЬ ДИСК С СРЕДН. НАДЕТЬ НА ЛЕВ.
СНЯТЬ ДИСК С СРЕДН. НАДЕТЬ НА ПРАВ.
СНЯТЬ ДИСК С ЛЕВ. НАДЕТЬ НА ПРАВ.
```

Для пирамиды из трех дисков требуется всего пять перемещений. При решении поставленной задачи для пирамиды из пяти дисков необходимо произвести 31 перемещение. Это вселяет в нас оптимизм: жрецам придется долго трудиться, прежде чем они переставят 64 диска, и наступит «конец света».

ПРИЛОЖЕНИЕ 3

НАЧИНАЮЩЕМУ ПОЛЬЗОВАТЕЛЮ ПЕРСОНАЛЬНОГО КОМПЬЮТЕРА О ТУРБО-ПАСКАЛЕ

Как уже упоминалось выше, трансляторы с языка Паскаль имеются практически на ЭВМ всех типов. Особенно удобна для работы система Турбо-Паскаль на персональных компьютерах (ПК), совместимых с IBM PC (ЕС-1840/41, ЕС-1850 и др.). Эта система состоит из транслятора, сервисных и прикладных программ. Транслятор реализует стандарт языка Паскаль и предоставляет ряд дополнительных возможностей, в частности, работу с данными типа BYTE (последовательность бит) и STRING (строка символов). Разрешается также использовать и прописные, и строчные буквы в любом сочетании.

В этом кратком приложении мы не ставим себе цель дать последовательное и полное описание языка Паскаль для ПК. Мы хотим помочь пользователю преодолеть психологический барьер и начать работать в системе Турбо-Паскаль; удобный и мощный язык, комфортный сервис экранного редактора, богатые графические возможности не могут никого оставить равнодушным. Получив начальные навыки, пользователь затем в процессе работы сам освоит все необходимые ему возможности системы.

Здесь мы приводим краткое руководство по работе на ПК, снабженное конкретными примерами. Предполагается, что пользователь знаком с клавиатурой ПК и имеет хотя бы небольшой опыт работы в системе DOS*).

---

*) П у л Л. Работа на персональном компьютере / Пер. с англ. — М.:Мир, 1986. — 383 с.

Х а у з е р Д., Х и р т Д ж., Х о у к и н с Б. Операционная система MS-DOS / Пер. с англ. А. Б. Пандре. — М.: Финансы и статистика, 1987. — 168 с.

I. Отличия языка Турбо-Паскаль от стандарта

В основном Турбо-Паскаль реализует стандартный Паскаль, но некоторые различия все же есть — как в сторону расширения возможностей, так и некоторых ограничений.

Перечислим коротко для посвященных читателей те и другие различия. Тому, кто еще не знаком с языком Паскаль, рекомендуем сначала прочесть описание языка, а затем вернуться и прочесть особенности версии Турбо.

Ограничения, накладываемые на стандартный Паскаль, состоят в следующем.

1. Отсутствуют стандартные процедуры PACK и UNPACK — упаковки/распаковки данных. Это объясняется тем, что в Турбо-Паскале упаковка в целях экономии места делается во всех допустимых случаях.

Ключевое слово PACKED не запрещено, но никакой роли не играет.

2. При работе с файлами отсутствуют стандартные процедуры GET и PUT. Их функции переданы процедурам READ и WRITE.

Это сделано в целях экономии памяти, для ускорения процессов обмена и облегчения написания программ.

3. Не допускаются записи (RECORD) с вариантами для динамических переменных.

Тем не менее можно реализовать подобную возможность, используя стандартную процедуру GETMEM.

4. Отсутствуют параметры-функции и параметры-процедуры.

5. Метки, используемые в процедуре или функции, надо описывать в этой процедуре или функции.

6. Вместо символа ↑ используется символ ^.

Эти немногочисленные ограничения компенсируются богатыми расширениями, предоставляемыми системой Турбо-Паскаль.

Ниже приводятся некоторые дополнительные, по сравнению со стандартом, возможности Турбо-Паскаля.

1. Расширен словарь языка. В идентификаторах можно, наряду с буквами и цифрами, использовать знак подчеркивания.

П р и м е р. PROCEDURE P_1;

Добавлены два символа: # и $.

Дополнительно введены ключевые слова:

ABSOLUTE	OVERLAY	STR
EXTERNAL	SHL	XOR
INLINE	SHR	

2. Введен новый тип данных: BYTE — байты, являющиеся подмножеством целого типа: 0..255.

3. Наряду с целыми и вещественными допускаются шестнадцатеричные константы. В написании таких констант первым символом является $.

П р и м е р. $125, 125, 125.0

Первая константа шестнадцатеричная, вторая целая, третья вещественная. Диапазон шестнадцатеричных констант от $0000 до $FFFF.

4. Структура программы допускает следующие вольности:

а) заголовок программы (PROGRAM...) может отсутствовать;

б) разделы LABEL, CONST, TYPE, VAR могут следовать в произвольном порядке, при этом заголовок раздела может быть указан более одного раза.

5. В качестве меток допускаются не только числа, но и любые идентификаторы.

6. В Турбо-Паскале зарезервированы следующие константы:

PI=3.1415936536E+00,
MAXINT=32767,
FALSE ("ложь"),
TRUE ("истина"),
NIL

7. Введены следующие дополнительные операции:

SHL — сдвиг кода на заданное число двоичных разрядов влево;

SHR — сдвиг кода на заданное число двоичных разрядов вправо;

XOR — исключающее ИЛИ.

8. Оператор CASE может иметь и часть, начинающуюся ключевым словом ELSE. Эта часть начинает работать в том случае, когда переключатель (селектор) не совпадает ни с одной меткой CASE.

9. После завершения цикла FOR параметр цикла всегда равен конечному значению, а не является неопределенным.

10. Для работы с портами ввода/вывода введены два стандартных байтовых массива MEM и PORT.

11. Максимальное число элементов множества установлено равным 256.

12. Большим удобством является инициализация переменных с помощью "констант с типом".

13. Введен ряд процедур и функций для работы с файлами. Помимо удобства, они обеспечивают прямой доступ к файлам.

14. Внешние устройства, подключенные к компьютеру (консоль, терминал, принтер), Турбо-Паскаль воспринимает как логические устройства. С ними работа идет просто как с текстовыми файлами.

Поэтому в Турбо-Паскале есть несколько дополнительных стандартных текстовых файлов: CON (консоль), TRM (терминал), KBD (клавиатура), LST (листинг), AUX (вспомогательное устройство), USR (устройство пользователя).

15. Низкоуровневые каналы ввода-вывода описываются в Турбо-Паскале как FILE без типа.

П р и м е р. VAR FF:FILE;

Размер компонент таких файлов равен 128 байтам. Файлы без типа совместимы с любыми другими, так как система в этом случае не контролирует совместимость типов.

Для обмена с этими файлами используются следующие процедуры: BLOCKREAD и BLOCKWRITE вместо READ и WRITE. Обмен происходит блоками по 128 байт.

16. При работе со "ссылками" есть возможность стирания всех динамических переменных из некоторой области. Для этого используются стандартные процедуры Турбо-Паскаля MARK и RELEASE.

Кроме того, используя стандартную функцию MEMAVAIL, можно в любой момент узнать объем свободного места в памяти для динамических переменных.

Процедурой GETMEM можно затребовать под динамические переменные заданный объем памяти.

Антиподом этой процедуры является FREEMEM, освобождающая заданный объем памяти.

По значению стандартной функции MAXAVAIL можно узнать длину максимального сплошного свободного "куска" памяти под динамические переменные.

17. Реализована возможность опроса клавиатуры.

18. Турбо-Паскаль снабжен богатыми возможностями работы с графикой. Помимо базисных графических средств, здесь есть и режим окон, и "черепашка".

Этот краткий перечень преимуществ Турбо-Паскаля далеко не исчерпывает всех дополнительных его возможностей, но даже и описанные здесь расширения делают Турбо-Паскаль достаточно привлекательным в глазах программиста.

II. Описание версии 5.0 языка Турбо-Паскаль

1. Общие сведения о системе. Для работы с системой в полном объеме на диске необходимо иметь файлы:

TURBO.EXE — основной файл, содержащий систему Турбо-Паскаль версии 5.0;

TURBO.TPL — файл, содержащий библиотеку стандартных программ;

TURBO.HLP — файл HELP с полной информацией о работе с данной версией Паскаля;

GRAPH.TPU — файл, реализующий графические возможности системы;

TURBO3.TPU — файл, осуществляющий совместимость с версией 3.0;

GRAPH3.TPU — файл, осуществляющий совместимость с GRAPH.P версии 3.0;

CGA.BGI/EGAVGA.BGI — драйверы графических устройств, если подключены нестандартные устройства.

Когда Вы освоитесь с версией 5.0, Вам могут потребоваться и другие файлы:

*.CHR — файлы, предоставляющие графические шрифты;

*.DOC — листинги стандартных программ;

TPUMOVER.EXE — файл, позволяющий помещать программы в файл TURBO.TPL либо убирать их оттуда;

UPGRADE.DTA и UPGRADE.EXE — файлы, преобразующие файлы, написанные на версии 3.0, в файлы версии 5.0;

MAKE.EXE — для автоматического включения в программу последней редакции всех используемых в ней файлов;

GREP.COM — файл, позволяющий проводить поиск заданного фрагмента по нескольким файлам одновременно;

TOUCH.COM — файл, работающий, как правило, на MAKE; меняет дату и время в указываемых файлах;

BINOBJ.COM — файл, используемый для преобразования двоичного файла в файл .OBJ;

TPCONFIG.EXE — используется в случае перехода от компиляции в TPC.EXE к компиляции в TURBO.EXE;

README — позволяет увидеть последние внесенные изменения;

README.COM — позволяет вывести на экран информацию из README.

Богатые возможности версии 5.0 требуют довольно много памяти.

Если версия 3.0 спокойно умещается на одной дискете, куда можно еще поместить множество небольших программ, то для работы с 5.0 желательно наличие винчестера.

Правда, для реализации некоторого усеченного Турбо-Паскаля версии 5.0 можно иметь только файлы:
TURBO.EXE, TURBO.HLP, GRAPH.TPU, TURBO.TPL, EGAVGA.BGI

2. Начинающим работать с версией 5.0. После вызова системы TURBO на экране в верхней строке появляется меню:
FILE EDIT RUN COMPILE OPTIONS DEBUG BREAK/WATCH

Здесь:

FILE — работа с файлами;

EDIT — редактирование программы;

RUN — запуск на счет;

COMPILE — компиляция программы;

OPTIONS — задание режимов;

DEBUG — работа с отладчиком;

BREAK/WATCH — информация для отладчика.

Под первой строкой — пустой экран в рамке, курсор установлен в верхней левой позиции. Под рамкой — строка:

F1—HELP, F5—ZOOM, F6—SWITCH, F7—TRACE, F8—STEP, F9—MAKE, F10—MENU

Здесь:

⟨F1⟩ — вызов системы HELP, которая следит за действиями и подсказывает, что нужно сделать, либо объясняет работу текущей процедуры или функции системы;

⟨F5⟩ — увеличение размеров окна;

⟨F6⟩ — переключение экрана;

⟨F7⟩ — трассировка;

⟨F8⟩ — пошаговая отладка;

⟨F9⟩ — создание файла .EXE;

⟨F10⟩ — вход в главное меню, т. е. в верхнюю строку экрана.

Итак, компьютер готов к работе. Дальнейшие действия пользователя проиллюстрируем конкретным примером.

П р и м е р. Пусть пользователь решил набрать текст программы, записать в файл на диске и запустить программу на счет. Для этого надо просто начать набирать программу.

После того как текст набран, следует перейти в главное меню (верхнюю строчку), т. е. нажать клавишу ⟨F10⟩. Курсор пропадает, но в верхней строке экрана закрашивается одно из полей меню. Клавишами ⟨→⟩, ⟨←⟩ этот индикатор-указатель можно перемещать от одного поля меню к другому.

Для наших целей надо установить индикатор на FILE и нажать ⟨ENTER⟩. На экране появляется слева сверху меню действий с файлами:

LOAD ⟨F3⟩ — загрузить с диска;

PICK ALT—⟨F3⟩ — загрузить предыдущий файл;

NEW — создать новый файл;

SAVE ⟨F2⟩ — спасти файл на диске;

WRITE TO — записать файл в ...;

DIRECTORY — выдать директорию;

CHANGE DIR — изменить директорию;

OS SHELL — подключить командный процессор;

QUIT ALT—X — выйти в DOS.

Индикатор теперь можно перемещать по этому меню стрелками ⟨↓⟩ и ⟨↑⟩. Так как прежде всего надо спасти файл на диске, то следует подогнать указатель к SAVE и нажать ⟨ENTER⟩, что эквивалентно нажатию клавиши ⟨F2⟩.

На экране появляется рамка со следующим текстом:

```
Rename NONAME
(Переименуйте NONAME)
<каталог>NONAME.PAS
```

Мигающий курсор приглашает набрать имя файла. Например, пользователь решил назвать этот файл N1. Он набирает эти два символа и нажимает <ENTER>. Файл запоминается на диске под именем N1.PAS, о чем система информирует в верхней строке рамки:

```
C:N1.PAS
```

Расширение .PAS система приписывает сама. После спасения файла пользователь опять входит в режим редактирования — перед ним текст программы.

Чтобы запустить программу на счет, надо нажать <F10> и, попав в меню, переместить индикатор на RUN, затем дважды нажать <ENTER>. Система выполнит программу и, на мгновенье выдав результаты, опять войдет в режим редактирования — выведет на экран текст программы, а предыдущая картинка сотрется. Чтобы результат счета остался на экране, следует выполнить после вывода оператор READLN;

П р и м е р.

```
. . .
BEGIN
  WRITELN('OK');
  READLN
END.
```

Есть другое средство увидеть результаты: нажать клавиши <ALT>/<F5> — на экран выведется файл OUTPUT. Тогда READLN в программе не требуется. Повторное нажатие <ALT>/<F5> возвращает на экран текст программы.

Если в процессе компиляции система обнаружила ошибки, то диагностика выводится в верхнюю строку рамки, а курсор устанавливается на неверном операторе текста программы.

Предположим, работа прервана, а затем компьютер опять включен для редактирования имеющегося на диске файла N1.PAS.

В этом случае действия будут такими:

1) вызвать TURBO.EXE, <ENTER>;

2) нажать <F10> — перейти в главное меню;

3) установить индикатор на FILE, нажать <ENTER>;

4) перейти по малому меню на LOAD, нажать <ENTER> либо вместо выполнения пп. 2 — 4 просто нажать <F3>;

5) на запрос системы набрать имя N1.PAS, нажать <ENTER>.

После этого на экране появится текст программы, который подлежит редактированию. Для отладки очень удобными являются режимы отладчика, включаемые комбинациями клавиш

<CTRL>/<F6> и <CTRL>/<F4>

Набрав имя переменной, можно увидеть на экране ее текущее значение.

Клавиши <F7> и <F8> включают режим пошаговой отладки, т. е. старт-стопный режим выполнения программы. Подробнее обо всем этом говорить здесь мы не будем, но Вы, уважаемый читатель, уже можете садиться за компьютер и набираться практического опыта работы с системой Турбо-Паскаль версии 5.0. Желаем успеха!

3. Типы переменных и констант. Рассмотрим более подробно набор типов данных, который предоставляет пользователю версия 5.0.

Скалярные типы. Ц е л ы й т и п (INTEGER). Данные этого типа — целые числа в диапазоне от –32768 до 32767, — занимают 2 байта памяти.

BYTE — также целые числа, но из диапазона от 0 до 255; занимают 1 байт памяти.

SHORTINT — целые из диапазона от –128 до 127, занимают 1 байт памяти.

WORD — целые из диапазона от 0 до 65535, занимают 2 байта памяти.

LONGINT — целые из самого широкого диапазона от –2147483648 до 2147483647, занимают 4 байта памяти.

Все эти типы совместимы при соблюдении соответствия диапазонам. Но действия со смесью типов опасны тем, что не диагностируется выход за диапазон.

Пример.

```
PROGRAM TEST3;
VAR I:SHORTINT; J:WORD;
BEGIN
  J:=200;
  I:=J;
  WRITELN ('I=',I)
END.
```

В результате работы программы значение I вместо 200 будет равно 56. Это произошло потому, что тип I не допускает чисел, больших 127. Опасность заключается в том, что система ничего не сообщила, а заслала неверный результат.

Вещественный тип. REAL — числа вида десятичных дробей с десятичной точкой либо числа, записанные через степень десяти.

Пример. 1.25; 0.3; 1.7E3

Такого типа данные принадлежат диапазону от –2.9E–38 до 1.7E+38, имеют 11–12 значащих цифр, занимают 6 байт.

Если компьютер снабжен сопроцессором INTEL 8087, 80287 или 80387, то аппаратно поддерживаются еще четыре разновидности вещественных чисел в режиме $E+N+ (см. п. 24).

SINGLE — вещественные числа, абсолютная величина которых принадлежит диапазону от 1.5E–4 до 3.41E+3. Они имеют 8 – 9 значащих цифр, занимают 4 байта памяти.

DOUBLE — вещественные числа, абсолютная величина которых изменяется в диапазоне от 5E–32 до 1.7E+30. Они имеют 16 – 17 значащих цифр, занимают 8 байт памяти.

EXTENDED — вещественные числа, по абсолютной величине принадлежащие диапазону от 1.9E–495 до 1.1E–493, имеют 20 значащих цифр, занимают 10 байт памяти.

COMP — вещественные числа без дробной части из диапазона от –2E63 до 2E63–1, имеют 20 значащих цифр, занимают 8 байт памяти.

Если вещественное число превышает по абсолютной величине максимально допустимое значение, возникает программное прерывание и выдается соответствующая диагностика. Если

абсолютная величина числа становится меньше допустимого, то компьютер выдает диагностику "Порядок исчез".

Вещественному типу не могут принадлежать индексы массивов, минимальное и максимальное значения параметра цикла FOR и ограниченного типа, переключатель (селектор) оператора CASE; не может быть вещественным и базовый тип множеств.

Символьный тип (CHAR). Данные этого типа принимают значение символа в коде ASCII, занимают 1 байт.

Тип STRING — строка символов.

В строке могут смешиваться символы, коды символов и управляющие символы. Последовательность символов заключается в апострофы, код символа предваряется знаком #, управляющий символ (т. е. который нажат одновременно с <CTRL>) предваряется знаком ^.

Пример. 'DUBNA'#105'MOSCOW'^B

Эта строка состоит из 13 символов, один из которых задан непосредственно кодом (105), другой — управляющий (^B), остальные — обычные символы.

Общий вид описания переменной S типа "строка":

```
S:STRING [n];
```

Здесь n — целая положительная константа — максимальное число символов в данной строке.

Процедуры и функции для работы со строками.

Функция **LENGTH(S)** целого типа выдает длину строки S, т. е. текущее число символов в ней.

Функция **COPY (S,NN,N)** типа STRING выдает подмножество строки S, состоящее из N символов и начинающееся с символа с номером NN. Здесь N и NN — целого типа.

Пример.

```
USES CRT;
VAR
  I:INTEGER;S1,S2:STRING[10];S3:STRING[20];
BEGIN
   CLRSCR;
   S1:='DUBNA';S3:=COPY(S1,1,3);
     WRITELN ('  ', S3);
```

```
READLN
END.
```

Переменная S3 будет содержать строку 'DUB'.

В приведенном примере впервые встретилось описание USES CRT. Здесь USES — ключевое слово, обозначающее заголовок раздела, CRT — имя модуля, который надо подключить к программе пользователя. Этот модуль содержит, в частности, процедуру CLRSCR. Процедура CLRSCR производит очистку экрана перед тем, как на него выдается результат. После оператора WRITELN, производящего вывод результатов на экран, указан "пустой" оператор READLN. Таким образом, программа содержит три "лишних" оператора. Однако они необходимы для того, чтобы получить результат на "чистом" экране.

Операция + дает строку, представляющую собой сцепление строк-операндов. Суммарная длина не должна превышать 255 символов.

П р и м е р.

```
PROGRAM N1;
USES CRT;
VAR
 I:INTEGER;S1,S2:STRING[10];S3:STRING[20];
BEGIN
   CLRSCR;
   S1:='DUB'; S2:='NA'; S3:=S1+S2;
       WRITELN (' ',S3);
   READLN
END.
```

Строка S3 будет содержать слово 'DUBNA'.

Операция =, примененная к строкам, дает значение TRUE, если строки равны по длине и все их символы совпадают; иначе — значение FALSE.

К строкам можно применять и другие операции отношения:

< , > , <= , >= , < >.

Процедура **DELETE (S, NN, N)** стирает в строке S N символов, начиная с символа с номером NN. Здесь S — исходная

строка-переменная, NN — типа INTEGER — номер первого стираемого символа, N — типа INTEGER — число стираемых символов.

Если номер NN больше длины строки, то строка S не меняется; если число стираемых символов N больше числа оставшихся от номера NN, то стираются все оставшиеся.

Процедура **INSERT (S1, S, NN)** вставляет строку S1 в строку S перед символом с номером NN. Здесь S1 — выражение типа STRING — вставляемая строка; S — переменная типа STRING, содержит строку, куда вставляется S1; NN — номер позиции в S, начиная с которой будет располагаться S1 — тип INTEGER.

П р и м е р.

```
VAR S1,S:STRING[5];
BEGIN
  S:='DA'; S1:='UBN';
  INSERT(S1,S,2);
  WRITELN(S);
  READLN
END.
```

Если в результате получается строка длиной более 255 символов, то она обрезается до 255 символов.

Функция **POS (ETALON, S)** целого типа отыскивает вхождение ETALON в строку S.

Здесь выражение ETALON типа STRING содержит эталон для поиска последовательности символов; S — строка, где ищется эталон.

Если POS = 0, то эталон в строке не найден, если POS = = n, то на n-м месте в S стоит первый символ эталона. Нумерация символов начинается с 1.

П р и м е р.

```
PROGRAM N1;
USES CRT;
VAR
 I:INTEGER;S1,S2:STRING[10]; S3,S4:STRING[10];
BEGIN
   CLRSCR;
```

```
    S3:='DUBNA';
    I:=POS('A',S3);
    WRITELN (' ',I);
READLN
END.
```

На экран выдается число 5 — номер позиции символа A в слове DUBNA.

Процедура **STR (N,S)** преобразует число N в строку символов S. Здесь N — выражение типа INTEGER или REAL; S — переменная типа STRING, куда посимвольно пересылается число.

П р и м е р.

```
PROGRAM N11;
USES CRT;
VAR
 I:INTEGER;S1,S2:STRING[10];S3,S4:STRING[10];
BEGIN
  CLRSCR;
  I:=567;
  STR(I,S1);
  WRITELN ( ' ',S1);
  READLN
END.
```

Процедура **VAL (S,IR,FLAG)** преобразует строку S в число IR. Здесь S — исходная строка символов, IR — переменная типа INTEGER или REAL, FLAG — переменная типа INTEGER. Строка преобразуется в целое или вещественное число в соответствии с типом IR.

Переменная FLAG после работы процедуры содержит либо 0, либо натуральное число. Если FLAG = 0, то преобразование прошло успешно, если FLAG = n, то на n-м месте в строке S система обнаружила недопустимый для числа символ.

П р и м е р.

```
PROGRAM N1;
USES CRT;
VAR
```

```
FLAG,IR,I:=INTEGER;S1,S2:STRING[10];S3,S4:STRING[10];
BEGIN
  CLRSCR;
  S1:='12345';
  VAL (S1,IR,FLAG);
  WRITELN (' ',FLAG);
  READLN
END.
```

Результатом будет число 12345; FLAG = 0.

П р и м е р.

```
PROGRAM N2;
USES CRT;
VAR
FLAG,IR,I:INTEGER;S1,S2:STRING[10];S3,S4:STRING[10];
BEGIN
  CLRSCR;
  S1:='125 350';
  VAL (S1,IR,FLAG);
  WRITELN (' ',FLAG);
  READLN
END.
```

Исходная строка содержит недопустимый символ — пробел, поэтому FLAG = 4, а преобразование строки в число не произойдет.

Ф а й л ы. Файл — это структура, состоящая из компонент одного типа. Число компонент не фиксировано.

В Турбо-Паскале существует также вид файлов, в которых только требуется, чтобы компоненты были одинаковой длины. Тип компонент в этом случае системе безразличен. Такие файлы называют "файлами без типа". Подробно тип FILE описан в п. 21. Здесь же мы кратко опишем "нормальные файлы", определение которых было дано выше.

Итак, общий вид описания типа:

TYPE R=FILE OF ⟨тип компонент⟩;

Тип компонент может быть любым, кроме файла. С файлами работают следующие процедуры и функции.

ASSIGN (F, 'имя файла на диске') — эта процедура должна быть выполнена прежде, чем начнется работа с файлом F. Данная процедура подключает к переменной F (типа файл) файл на диске. Если указанного имени нет на диске, файл автоматически заводится заново.

RESET (F) — имитация перемотки магнитной ленты (файла) в начало без стирания. Эта процедура готовит файл F к чтению. Но можно после этого и писать в файл.

READ (F,A,B,C,...); — чтение из файла F.

Здесь F — файл, откуда читается информация;

A, B, C, ... — переменные типа компонент файла, куда покомпонентно считывается информация из файла.

REWRITE (F) — "перемотка в начало со стиранием". Эта процедура устанавливает указатель на начало файла и стирает информацию, если она в файле была.

WRITE (F,A,B,C,...); — запись в файл F.

Здесь F — файл, куда пишется информация;

A, B, C, ... — переменные типа компонент файла, в них содержится информация, которая покомпонентно записывается в файл F.

Функция **FILESIZE (F)** типа LONGINT выдает число компонент файла F.

SEEK (F,N) — процедура, устанавливающая указатель файла F на компоненту с номером N. Нумерация начинается с нуля.

Здесь N — выражение типа LONGINT. С помощью этой процедуры можно дописать информацию в конец файла:

```
SEEK (F,FILESIZE (F));
WRITE (F,A);
```

CLOSE (F) — процедура, закрывающая файл F.

Эту процедуру необходимо выполнить после того, как закончена запись в файл, а также перед окончанием программы, если была другая работа с файлами.

Функция **EOF (F)** булевского типа дает TRUE, если указатель файла "наткнулся" на маркер "конец файла".

Одной из часто используемых разновидностей файлов является текстовый файл. Это файлы, состоящие из символов,

разбитых на строки. В конце каждой строки стоит маркер "конец строки". Такие файлы имеют стандартный тип TEXT.

Помимо перечисленных выше процедур и функций, с текстовыми файлами работают следующие.

ASSIGN (T,CON) "подключает" текстовый файл T к консоли.

ASSIGN (T,PRN) "подключает" текстовый файл T к печатающему устройству.

ASSIGN (F,LST1) либо **ASSIGN (F,LST2)** "подключает" текстовый файл T к первому или второму принтеру.

Процедура **ASSIGN (T,NUL)** "подключает" текстовый файл T к фиктивному устройству. При чтении из такого файла сразу же поступает сообщение "конец строки", а вывод игнорируется.

Процедура **ASSIGNCRT (T)** "подключает" текстовый файл T к экрану терминала, т. е. вывод в текстовый файл T соответствует выводу на экран. Для обращения к ASSIGNCRT (T) необходимо в программе задать строку

USES CRT;

Процедура **APPEND (T)** открывает текстовый файл T для дозаписи информации. Если файла T на диске нет, то выдается "ошибка ввода/вывода".

Процедура **READLN (T,A,B,C, ...)** читает информацию из текстового файла T и засылает в переменные A, B, C, ... После прочтения указанной порции информации происходит переход к новой строке текстового файла.

Процедура **WRITELN (T,A,B,C, ...)** записывает информацию из переменных A, B, C, ... в текстовый файл T и ставит маркер "конец строки".

Процедура **SETTEXTBUF (T,B,N)** задает для обмена с текстовым файлом T буфер B размером N байт.

Здесь T — переменная — текстовый файл, B — любая переменная, N — целое выражение типа WORD — может отсутствовать.

Эту процедуру имеет смысл использовать только для задач с большим объемом ввода/вывода. Увеличение буфера по сравнению со стандартным (в 128 байт) ускоряет процесс обмена и создает более щадящий режим работы дисководов.

Если параметр N опустить, то по умолчанию будет использоваться вся область памяти, занятая буфером.

Пример.

```
. . .
VAR B:CHAR; T:TEXT;
. . .
SETTEXTBUF (T,B);
. . .
```

Эту процедуру можно применять либо только сразу после

ASSIGN, RESET, REWRITE, APPEND

либо надо предварительно закрыть файл, а затем вызвать SETTEXTBUF. Иначе могут произойти крупные неприятности.

Функция **EOLN (T)** типа BOOLEAN выдает TRUE, если указатель файла "наткнулся" на маркер "конец строки".

Функция **SEEKEOLN (T)** типа BOOLEAN выдает TRUE, когда до маркера "конец строки" остались одни пробелы или символы горизонтальной табуляции либо встретился сам маркер "конец строки".

Функция **SEEKEOF (T)** типа BOOLEAN выдает TRUE, когда до маркера "конец файла" остались одни пробелы, концы строк и символы горизонтальной табуляции либо попался сам маркер "конец файла".

Стандартными текстовыми файлами являются файлы INPUT (ввода) и OUTPUT (вывода). Имена этих файлов можно не указывать в процедурах работы с текстовыми файлами. Так, если в процедурах

READ, READLN, EOF, EOLN, SEEKEOLN, SEEKEOF

не указан файл, то по умолчанию предполагается, что это — INPUT. Аналогично, в процедурах WRITE и WRITELN, при отсутствии в параметрах имени файла, предполагается, что это — OUTPUT (экран).

Остановимся кратко на "файлах без типов". Итак, о компонентах файла известно только, что они имеют одинаковую длину, кратную 128 байтам.

Общий вид описания типа:

```
TYPE R=FILE;
```

Для работы с такими файлами используются, во-первых, все процедуры и функции для "файлов с типами". Кроме них существуют и специально предназначенные для "файлов без типов".

Процедура **RESET (F,M)** готовит к чтению файл F, в котором каждая компонента длины 128*M байт. Если M отсутствует, то длина компонент 128 байт.

Процедура **REWRITE (F,M)** готовит файл к записи, переводит указатель в начало, стирает информацию в файле. Размер компонент 128*M.

Процедура **BLOCKREAD (F,B,N,I)** читает из файла F в переменную B (буфер) N компонент. I — число реально считанных компонент. I можно не указывать. В этом случае, если реально считается менее N компонент, произойдет прекращение выполнения программы — "ошибка ввода/вывода".

Здесь F — переменная "файл без типа"; B — любая переменная; N — выражение типа WORD; I — переменная типа WORD, необязательный параметр.

Примечание. Если последняя компонента файла неполная, то она отбрасывается.

Если установлен режим компиляции $I–, то при успешном срабатывании процедуры BLOCKREAD значение функции IORESULT будет нуль: ненулевое значение IORESULT сигнализирует об ошибке при чтении файла.

Процедура **BLOCKWRITE (F,B,N,I)** записывает в файл F из переменной B (буфера) N компонент. I — число реально записанных компонент. Если I не указано и реально запишется в файл менее N компонент, то произойдет прекращение выполнения программы — "ошибка ввода/вывода".

Здесь F — переменная "файл без типа"; B — любая переменная; N — выражение типа WORD; I — переменная типа WORD, необязательный параметр.

Если последняя компонента неполная (т. е. содержит менее 128*M байт), то она отбрасывается.

Если установлен режим компиляции $I–, то значение функции IORESULT будет нулевым для безошибочной записи в файл; если были обнаружены ошибки, то функция IORESULT имеет ненулевое значение.

Процедуры и функции для работы с указателями.

Процедура **NEW (P)** отводит место в памяти под динамическую переменную, связанную с указателем P; адрес этой переменной засылается в P.

Процедура **DISPOSE (P)** освобождает память, занятую динамической переменной, адрес которой находится в указателе P.

Процедура **GETMEM (P,N)** выделяет блок памяти в N байт и адрес начала блока помещает в указатель P.

Процедура **FREEMEM (P,N)** освобождает блок памяти в N байт, начиная с адреса, указанного в P.

Процедуры **MARK (P)** и **RELEASE (P)**: в результате работы этих двух процедур освобождается вся память в "хипе", начиная с адреса, являющегося значением указателя P.

Нельзя смешивать в одной программе освобождение памяти с помощью FREEMEM, DISPOSE и RELEASE! Результат может быть трагическим.

Функция **MAXAVAIL** типа LONGINT, дающая размер наибольшего свободного участка в "хипе". В "хипе" обычно чередуются занятые и свободные участки памяти, получающиеся в результате работы DISPOSE и FREEMEM. Поэтому бывает необходимо узнать, смогут ли быть расположены подряд определенные данные. Эту информацию и дает функция MAXAVAIL.

Функция **MEMAVAIL** дает размер суммы всех свободных участков памяти. Эту функцию можно использовать для проверки возможности размещения больших объемов данных в памяти. Тип функции — LONGINT.

Функция **PTR (SG,OFST)** типа указатель. Здесь SG и OFST — типа WORD.

Эта функция формирует адрес, воспринимая SG как базу сегмента, а OFST — смещение в этом сегменте.

Функция **OFS (A)** типа WORD выдает смещение в адресе произвольной переменной A.

Функция **SEG (A)** типа WORD выдает базу сегмента в адресе переменной A.

4. Структура программы. Турбо-Паскаль версии 5.0 предполагает следующую структуру программы.

Каждая программа состоит из трех основных частей: заголовка, раздела USES и блока.

Р а з д е л USES содержит перечисление стандартных *модулей*, используемых данной программой.

Существует семь стандартных модулей:

CRT, DOS, GRAPH, GRAPH3, OVERLAY, PRINTER, SYSTEM, TURBO3.

Модуль **SYSTEM** содержит функции и процедуры для ввода/вывода, работы со строками, динамического распределения памяти и др. Этот модуль присутствует по умолчанию, и его можно не указывать в USES.

Модуль **CRT** необходим для вывода на экран в графическом режиме. Он содержит процедуры и функции, реализующие изменение размера букв, числа строк, работу с окнами, клавиатурой, перемещение курсора, очистку экрана и т. д.

Модуль **DOS** "заведует" работой с файлами, указателями и т. д. Он содержит процедуры обращения к процедурам и функциям MS-DOS.

Модуль **GRAPH** обеспечивает все разнообразие графических возможностей системы Турбо-Паскаль версии 5.0.

Модуль **GRAPH3** необходим для совместимости с программами, написанными в системе Турбо-Паскаль версии 3.0.

Модуль **OVERLAY** предназначен, как показывает его имя, для работы оверлейных программ, т. е. настолько больших, что их приходится разбивать на части, чтобы эти части умещались в оперативной памяти компьютера.

Модуль **PRINTER** надо указывать в разделе USES только в том случае, если программа выводит информацию на принтер.

Модуль **TURBO3** обеспечивает совместимость данной программы с программами, написанными на Турбо-Паскале версии 3.0.

Раздел USES, как и заголовок, может в программе отсутствовать.

Турбо-Паскаль требует, чтобы метки были описаны именно в той процедуре или функции, где они используются. Таким образом, глобальные метки — описанные в PROGRAM — нельзя указывать в тексте процедуры или функции.

Раздел констант имеет общий вид

CONST C_1 = ⟨константа или выражение⟩;
C_2 = ⟨константа или выражение⟩;
. . .

Здесь C_1, C_2, ... — произвольные идентификаторы, а справа — конкретные выражения (так как константа — тоже выражение), которые можно вычислить без использования переменных и "констант с типом". Из функций в выражениях для констант можно использовать только следующие:

ABS, CHR, HI, LENGTH, LO, ODD, ORD, PRED, PTR,
ROUND, SIZEOF, SUCC, SWAP, TRUNC, PI.

Пример.

```
CONST C1=5.3;
      C21=C1-3;
      ALFA=30+2*PI;
```

Система Турбо-Паскаль версии 5.0 "знает" сама следующие константы:

FALSE, TRUE,
MAXINT=32767,
MAXLONGINT=2147483647,
NIL.

Турбо-Паскаль позволяет при описании констант указывать и *их тип*. Такие константы называются "константы с типом" и имеют общий вид описания:

C_1 : ⟨тип$_1$⟩ = ⟨значение$_1$⟩;
C_2 : ⟨тип$_2$⟩ = ⟨значение$_2$⟩;

Здесь после имени константы ставится двоеточие, затем указывается тип и справа от равенства задается значение константы.

Пример. M:INTEGER=2*314;

Константы, которым приписан тип, получают право играть роль фактических параметров-переменных. Известно, что параметрами-переменными могут быть только переменные. Но для таких констант сделано исключение.

На число констант с типом накладывается следующее ограничение: они должны занимать не более 64 К памяти.

Если константа — многомерный массив, то компоненты каждой размерности заключаются в круглые скобки, а друг от друга отделяются запятыми.

Пример.

```
CONST
  A:ARRAY[1..5] OF INTEGER = (3,7,25,4,8);
  B:ARRAY[1..2,1..3] OF INTEGER = ((3,4,1),(2,7,6));
```

Раздел действий содержит операторы.

Начинающему читателю предлагаем познакомиться с операторами в п. 9. Здесь мы приведем лишь некоторые дополнительные возможности, предоставляемые версией 5.0.

1. Знак @ перед именем переменной означает адрес этой переменной.

Пример. P:=@A; — переменной-указателю P будет присвоен адрес переменной A.

2. Стандартные арифметические процедуры и функции.

DEC (X,N) уменьшает значение X на величину N. Если N отсутствует, то по умолчанию N = 1. Здесь X, N — типа LONGINT.

INC (X,N) увеличивает значение X на целую величину N. Если N отсутствует, то предполагается, что N = 1. Здесь X и N — типа LONGINT.

HI (X) — функция типа BYTE — выдает старший байт первого слова аргумента X; X — типа INTEGER.

GO (X) — функция типа BYTE — выдает младший байт первого слова аргумента X; X — типа INTEGER.

SWAP (X) — функция типа INTEGER — выдает значение аргумента X, в котором поменялись местами младший и старший байты; X — типа INTEGER.

RANDOMIZE — процедура — запускает генератор случайных чисел.

RANDOM — функция типа REAL, генератор случайных чисел, равномерно распределенных на отрезке [0,1].

RANDOM (N) — функция типа WORD — выдает случайное число с равномерным распределением на [0,N]; N типа WORD.

SIZEOF (X) — функция типа WORD — выдает число байт, занимаемых аргументом X; X — любая переменная или имя типа.

UPCASE (X) — функция типа CHAR — выдает большую латинскую букву, соответствующую букве-аргументу. Если X содержит символ, отличный от малой буквы, то он выдается без изменений.

FILLCHAR (X,COUNT,C) — процедура заполнения памяти объемом COUNT байт, начиная с того места, где располагается произвольная переменная X, символом, заданным в переменной C.

Здесь:

X — переменная произвольного типа;

COUNT — переменная типа WORD;

C — переменная типа CHAR.

MOVE (SOURSE,DEST,COUNT) — процедура пересылки COUNT байт из SOURSE в DEST.

Здесь SOURSE,DEST — произвольные переменные, отмечающие области памяти "откуда — куда".

PARAMCOUNT — функция типа WORD — выдает число параметров, переданных в командной строке.

PARAMSTR (I) — функция типа STRING — выдает I-й параметр командной строки. I — типа WORD.

Параметры-переменные без типа. Формальные параметры могут не иметь описания через тип.

Пример. (VAR X,Y; A,B:REAL);

Здесь переменные X и Y не описаны через тип; параметры A и B имеют привычное описание.

В случае, когда формальный параметр имеет вид "без типа", соответствующим фактическим параметром может быть переменная любого типа. Так, например, можно сравнить две переменные произвольных, но одинаковых типов. О том, чтобы типы были одинаковы, должна позаботиться вызывающая программа.

Подчеркнем, что параметрами без типа могут быть только параметры-переменные.

5. Графика. При работе с графической системой Турбо-Паскаль версии 5.0 необходимо наличие файлов GRAPH.TPU и системных файлов с расширениями CHR и BGI.

Файл GRAPH.TPU содержит все процедуры и функции работы с графикой, файлы .CHR содержат графические шрифты, а BGI — драйверы графических устройств.

Для включения в программу графики необходимо указать в разделе USES:

```
USES GRAPH;
```

Опишем некоторые процедуры и функции, входящие в модуль GRAPH.TPU.

1. Процедура **INITGRAPH (N,M 'подкаталог')** устанавливает графический режим.

Здесь:

N — переменная типа INTEGER, содержащая номер графического драйвера;

M — целая переменная типа INTEGER, содержащая номер графического режима;

'подкаталог' — типа STRING — подкаталог, в котором находятся графические драйверы.

Если работать с автоматическим выбором графического драйвера, то обращение к процедуре выглядит следующим образом: надо задать N = 0 и подкаталог — пробелом.

Режим M может иметь одно из трех значений: 0, 1 или 2. Каждому режиму соответствует определенный номер палитры, число точек на экране и основные цвета.

M	№ палитры	Число точек	Цвета
0	1	320*200	Красный, желтый, зеленый
1	2	320*200	Светло-бирюзовый, белый, малиновый
2	–	640*200	Черно-белый

Пример.

```
VAR M,N:INTEGER;
BEGIN
```

```
N:=0; M:=0;
INITGRAPH (N,M,' ');
```

Задается автоматический выбор графического драйвера и режим первой палитры, 320*200 точек, основные цвета — красный, желтый, зеленый. Процедура INITGRAPH оповещает пользователя через системную переменную GRAPHRESULT о благополучном исходе. Только если GRAPHRESULT=GROK (где GROK=0 — зарезервированная константа), то все в порядке.

После такой проверки можно начинать работу в графическом режиме. При этом с экраном связана система координат с началом в левом верхнем углу экрана, ось x направлена вправо, ось y — вниз.

Для перехода к привычной системе координат с началом в центре экрана (320*200) и осью ординат, направленной вверх, достаточно сделать замену:

```
XX: = X–160;
YY: = –Y+100;
```

П р и м е ч а н и е. Координаты точек предполагаются целочисленными. Чтобы работать в графической системе с вещественными координатами, их следует привести к заданному диапазону и округлить.

Например, пусть координата X — вещественное число, меняющееся от X0 до X1. Привести к диапазону целых координат от 0 до 320 можно следующим оператором:

```
XX: = ROUND((X–X0)*320/(X1–X0));
```

Здесь XX может иметь тип INTEGER, а еще лучше — ограниченный тип (0..320).

2. Процедура **CLOSEGRAPH** выводит систему из графического режима.

П р и м е р.

```
. . .
BEGIN
. . .
 INITGRAPH (M,N, ' ');
{ Работа с графикой }
 CLOSEGRAPH;
 . . .
END.
```

3. Процедура **RESTORECRTMODE** восстанавливает текстовый режим, из которого произошел переход в графический режим. Все атрибуты текстового режима были запомнены процедурой INITGRAPH.

4. Процедура **SETCOLOR (N)**. Здесь N — цвет — целая переменная из диапазона от 0 до 15. Эта процедура задает цвет вычерчиваемых в дальнейшем линий. Число 15 — максимально возможное для самой богатой палитры. Но многие палитры допускают всего четыре цвета, соответствующие значениям цвета 0, 1, 2, 3.

Пример.

```
M:=2;
SETCOLOR(M);
```

5. Функция **GETMAXCOLOR** — типа WORD — выдает максимально допустимый номер цвета в данной палитре. Эту функцию можно указывать в качестве фактического параметра при обращении к процедурам и функциям, требующим задания цвета.

6. Процедура **SETLINESTYLE (M,N,K)** задает способ вычерчивания линий в последующих операторах.

Здесь M — выражение типа WORD — определяет характер линии:

M = 0 — сплошная,

M = 1 — точечная,

M = 2 — штрих-пунктирная,

M = 3 — пунктирная,

M = 4 — образец задается в параметре N,

N — выражение типа WORD — образец, заданный шестнадцатеричным числом,

K — выражение типа WORD — толщина линии: K = 1 — нормальная, K = 2 — жирная.

Пример.

```
M:=3;
K:=2;
SETLINESTYPE (M,0,K);
. . .
SETLINESTYPE (4,$C3,2);
```

7. Процедура **LINE** (X_1, Y_1, X_2, Y_2) рисует отрезок прямой, соединяющей точки с координатами (X_1, Y_1) и (X_2, Y_2).

Здесь X_1, Y_1, X_2, Y_2 — целые выражения; цвет и характер линии соответствует результату работы процедур SETCOLOR и SETLINESTYLE. Если к этим процедурам обращения не было, то последующие линии будут сплошными, нормальной толщины и цвета, установленного по умолчанию.

П р и м е р. Нарисуем отрезок пунктирной прямой, соединяющей точки (10,20) и (100,69). Решение:

```
PROGRAM M1;
USES GRAPH,CRT;
VAR I,N,M:INTEGER;
BEGIN
  N:=0; M:=0;
  INITGRAPH (N,M,' ');
  CLEARDEVICE;
  SETLINESTYLE (3,0,1);
  LINE (10,20,100,69);
  READLN
END.
```

Здесь процедура CLEARDEVICE очищает экран в графическом режиме.

8. Процедура **LINEREL (DX,DY)** рисует отрезок прямой, соединяющий точку (X, Y) с точкой (X + DX, Y + DY).

П р и м е р. LINEREL (15,0);

Нарисуется горизонтальный отрезок, соединяющий данную точку с точкой, находящейся на 15 единиц правее данной.

9. Процедура **LINETO (X,Y)** соединяет отрезком прямой *текущую* точку с точкой (X,Y) и текущей становится точка (X, Y). Здесь X, Y — целые выражения.

П р и м е р. LINETO (100,100);

Текущая точка будет соединена отрезком прямой с точкой (100,100). В начале работы текущей точкой является точка (1,1). Поэтому если LINETO (100,100) — первый оператор, рисующий прямые линии, то будет построен отрезок, соединяющий точки (1,1) и (100,100), и текущей точкой станет точка (100,100).

Если затем выполнить оператор LINETO (20,20), то будет построен угол с вершиной в (100,100). Заметим, что процедура LINE не меняет положение текущей точки.

10. Процедура **MOVEREL (DX,DY)** смещает текущую точку (X, Y) в точку (X + DX, Y + DY).

Здесь DX, DY — целые выражения.

П р и м е р. MOVEREL (0, –15);

Текущая точка "подпрыгнула" на 15 единиц вверх.

11. Функция **GETX** — целого типа — выдает координату X текущей точки.

12. Функция **GETY** — целого типа — выдает координату Y текущей точки.

П р и м е р.

```
VAR XX, YY:INTEGER;
    . . .
    XX:=GETX;
    YY:=GETY;
```

13. Процедура **MOVETO (X,Y)** перемещает текущую точку в точку (X,Y). Здесь X, Y — целые выражения. Эта процедура не рисует никаких линий.

14. Функция **GETMAXX** — целого типа — выдает максимально допустимое значение координаты X.

П р и м е р.

```
VAR MX:INTEGER;
. . .
MX:=GETMAXX;
```

Если установлен экран 320*200, то MX = 320; если установлено окно, то значение MX будет равно ширине окна.

15. Функция **GETMAXY** — целого типа — выдает максимально допустимое значение координаты Y.

П р и м е р.

```
VAR MY:INTEGER;
. . .
MY:=GETMAXY;
```

В обычном случае MY = 200; если установлено окно, то это значение будет равно высоте окна.

16. Процедура **PUTPIXEL (X,Y,N)** ставит на плоскости точку (X, Y) цвета N.

Здесь X, Y — целые выражения, N — целое число — номер цвета из диапазона 0..15, но не более максимально допустимого в данной палитре. Верхнюю границу N выдает функция GETMAXCOLOR.

П р и м е р.

```
VAR X,Y:INTEGER;
. . .
X:=25; Y:=100;
PUTPIXEL (X,Y,GETMAXCOLOR);
```

На экране будет поставлена точка (X,Y) цвета, соответствующего максимальному номеру в данной палитре.

17. Функция **GETPIXEL (X,Y)** — целого типа — выдает номер цвета точки экрана с координатами (X, Y).

П р и м е р.

```
VAR X,Y,N:INTEGER;
. . .
X:=25; Y:=100;
N:=GETPIXEL (X,Y);
```

Значение N будет соответствовать номеру цвета точки экрана с координатами (25,100).

18. Процедура **RECTANGLE** (X_1,Y_1,X_2,Y_2) рисует прямоугольник.

Здесь (X_1,Y_1) — левый верхний угол, (X_2,Y_2) — правый нижний угол прямоугольника, т. е. $X_1 < X_2$, $Y_1 < Y_2$.

Стороны прямоугольника параллельны осям координат. X_1, X_2, Y_1, Y_2 — целые выражения.

Цвет и вид линии либо выбираются по умолчанию, либо устанавливаются предварительно процедурами SETCOLOR и SETLINESTYPE.

П р и м е р.

```
PROGRAM M1;
USES GRAPH,CTR;
VAR N,M:INTEGER;
BEGIN
  N:=0;M:=0;
```

```
    INITGRAPH (N,M,' ');
    CLEARDEVICE;
    SETLINESTYLE (3,0,2);
    RECTANGLE (10,20,100,69);
    READLN
END.
```

В этом примере рисуется прямоугольник жирной штриховой линией, левая верхняя вершина — (10,20), правая нижняя — в точке (100,69).

19. Процедура **SETFILLPATTERN (B,N)** задает способ и цвет закрашивания областей.

Здесь B — переменная типа ARRAY [1..8] OF BYTE, определяющая способ закрашивания; N — целая переменная — номер цвета, которым закрашивается область.

Переменная B содержит байты, каждый из которых представляет собой двоичное число. Единицы в этом числе соответствуют ярким элементам, нули — темным. Например, байт 10101010 задает чередование темных и ярких точек, а массив

```
1 0 1 0 1 0 1 0
0 1 0 1 0 1 0 1
1 0 1 0 1 0 1 0
0 1 0 1 0 1 0 1
1 0 1 0 1 0 1 0
0 1 0 1 0 1 0 1
1 0 1 0 1 0 1 0
0 1 0 1 0 1 0 1
```

задает закрашивание области в шахматном порядке.

Массив, состоящий из чередующихся шестнадцатеричных чисел $FF и $00, определяет горизонтальную штриховку области

```
1 1 1 1 1 1 1 1  — число $FF,
0 0 0 0 0 0 0 0  — число $00,
1 1 1 1 1 1 1 1  — число $FF,
0 0 0 0 0 0 0 0  — число $00,
1 1 1 1 1 1 1 1  — число $FF,
0 0 0 0 0 0 0 0  — число $00,
1 1 1 1 1 1 1 1  — число $FF,
0 0 0 0 0 0 0 0  — число $00.
```

Выбирая тип закрашивания, следует изобразить картинку в виде двоичных чисел — чередования ярких и темных точек (единиц и нулей), а затем найти соответствующие шестнадцатеричные числа и задать массив в разделе CONST.

Подбирая двоичные изображения байт, можно осуществить любой характер закрашивания: штриховку, в полоску, точками и т. д. Байты задаются как шестнадцатеричные числа. Так, двоичное число 10101010 соответствует шестнадцатеричному числу $AA, а двоичное число 01010101 — это шестнадцатеричное $55.

В системе определен тип

```
FILLPATTERNTYPE=ARRAY[1..8] OF BYTE;
```

поэтому можно использовать этот тип, не определяя его в программе, как это делается, например, для REAL.

20. Процедура **BAR** (X_1, Y_1, X_2, Y_2) рисует прямоугольник и закрашивает его.

Здесь:

X_1, Y_1, X_2, Y_2 — целые выражения, $X_1 < X_2$, $Y_1 < Y_2$;

(X_1, Y_1) — левый верхний угол;

(X_2, Y_2) — правый нижний угол прямоугольника.

Характер линии и ее цвет, если не установлены предварительно процедурами SETCOLOR и SETLINESTYLE, то выбираются по умолчанию. Цвет закраски задается предварительно процедурой SETFILLPATTERN.

П р и м е р. Пусть требуется закрасить в полоску прямоугольник, построенный по точкам (12,14) и (59,61). Приведем полностью программу:

```
PROGRAM N1;
USES CRT,GRAPH;
CONST
 B:FILLPATTERNTYPE=($FF,$00,$FF,$00,$FF,$00,$FF,$00);
VAR
  NN,N,M:INTEGER;
BEGIN
  CLRSCR;NN:=3; N:=2; M:=2;
  INITGRAPH (N,M, ' ');
  SETFILLPATTERN (B,NN);
```

```
  BAR (12,14,59,61);
  READLN
END.
```

21. Процедура **BAR3D (X_1, Y_1, X_2, Y_2, D, F)** рисует параллелепипед и закрашивает его.

Параллелепипед задается концами диагонали передней грани (X_1, Y_1), (X_2, Y_2) и длиной перпендикулярного ей ребра D.

Здесь:

X_1, Y_1, X_2, Y_2 — целые выражения — координаты: (X_1, Y_1) — левая верхняя точка, (X_2, Y_2) — правая нижняя;

D — выражение типа WORD — длина ребра;

F — булевское выражение.

Если F = TRUE, то верхняя грань параллелепипеда рисуется; если F = FALSE, не рисуется. В этом случае предполагается, что над параллелепипедом будут надстраиваться еще какие-либо фигуры.

В зависимости от ориентации оси 0Y верхняя грань может оказаться и нижней на экране.

П р и м е р. Построить параллелепипед и закрасить его переднюю грань в полоску можно с помощью программы:

```
PROGRAM N1;
USES CRT,GRAPH;
CONST
 *B:FILLPATTERNTYPE=($FF,$00,$FF,$00,$FF,$00,$FF,$00);
VAR
  NN,N,M:INTEGER;
BEGIN
  CLRSCR;NN:=3; N:=2; M:=2;
  INITGRAPH (N,M, ' ');
  SETFILLPATTERN (B,NN);
  BARD3D(12,14,54,61,20,TRUE);
  READLN
END.
```

22. Процедура **FLOODFILL (X, Y, N)** — "выливает" краску в точку (X, Y), и оттуда краска "растекается" по области, ограниченной цветом N. Если граница не замкнута, краска "вытекает" за границу и может "залить" весь экран.

Здесь X, Y — целые выражения — координаты точки; N — номер цвета, которым нарисована граница.

Значение N лежит в диапазоне от 0 до значения, выдаваемого функцией GETMAXCOLOR, но не более 15.

В качестве X, Y можно брать координаты любой точки, лежащей внутри закрашиваемой замкнутой области. Если (X,Y) лежит вне области, ограниченной цветом N, то закрасится экран вне области. Если взять N = 0, то закрашивание будет вестись цветом фона, что соответствует очистке области.

П р и м е р.

```
PROGRAM M1;
USES GRAPH,CRT;
VAR N,M:INTEGER;
BEGIN
  N:=0; M:=0;
  INITGRAPH (N,M,' ');
  CLEARDEVICE;
  LINE (10,10,100,100); MOVETO (100,100);
  LINETO (10,100); LINETO (10,10);
  FLOODFILL (15,90, GETMAXCOLOR);
  READLN
END.
```

В результате будет закрашен треугольник, построенный по трем точкам: (10,10), (100,100) и (10,100). Краска "выливается" в точку (15,90), находящуюся внутри треугольника. "Стереть" эту краску можно обращением

```
FLOODFILL (15,90,0);
```

(Значение 0 присвоено цвету фона.)

Закрасить весь экран и на его фоне оставить незакрашенным треугольник можно обращением

```
FLOODFILL (100,10,GETMAXCOLOR);
```

Так как точка (100,10) находится вне треугольника, краска, "растекаясь", "зальет" весь экран, но не попадет внутрь треугольника.

23. Процедура **CIRCLE (X,Y,R)** — рисует окружность с центром в (X,Y) и радиусом R. Цвет и характер линии могут

быть установлены программистом процедурами SETCOLOR и SETLINESTYLE либо берутся по умолчанию.

Здесь:

X, Y — целые выражения — координаты центра;

R — целое выражение типа WORD — радиус окружности.

П р и м е р. CIRCLE (50,50,10);

Рисуется окружность с центром в точке (50,50) и радиусом R = 10.

24. Процедура **ARC (X,Y,ALFA,BETA,R)** рисует дугу окружности радиуса R с центром в (X,Y) от начального угла ALFA до конечного угла BETA.

Здесь X, Y, ALFA, BETA, R — целые выражения.

Отсчет углов ведется от горизонтального диаметра против часовой стрелки.

П р и м е р. Нарисовать окружность радиуса 30 с центром в точке (100,50) и четверть окружности с тем же центром, но радиусом 50 можно программой

```
PROGRAM N1;
USES CRT GRAPH;
VAR N,M:INTEGER;
BEGIN
  CLRSCR; N:=2; M:=2;
  INITGRAPH(N,M,' ');
  ARC(100,50,0,90,50);
  CIRCLE(100,50,30);
  READLN
END.
```

25. Процедура **GETARCCOORDS (A)** позволяет узнать координаты начала и конца построенной (с помощью процедуры ARC) дуги. Здесь A — переменная типа "запись", причем специально зарезервированного типа ARCCOORDSTYPE. Этот тип описан в системе как

```
TYPE ARCCOORDSTYPE=RECORD
                     X,Y:INTEGER;
                     XSTART,YSTART,XEND,YEND:WORD
                   END;
```

Пусть A:ARCCOORDSTYPE, тогда после обращения к процедуре GETARCCOORDS(A) получим A.X и A.Y — координаты центра окружности; A.XSTART и A.YSTART — координаты начала дуги; A.XEND и A.YEND — координаты конца дуги.

П р и м е р. Конец дуги окружности надо соединить отрезком с началом координат. Окружность задана центром (40,40) и радиусом R = 20, начальный угол $\alpha = 15°$, конечный угол $\beta = 40°$.

Фрагмент программы:

```
VAR A:ARCCOORDSTYPE;
. . .
ARC(40,40,15,40,20);
GETARCCOORDS(A);
WITH A DO
  LINE (XEND,YEND,0,0);
  . . .
```

26. Процедура **FILLELLIPSE (X,Y,A,B)** рисует и закрашивает эллипс с центром в точке (X,Y) горизонтальной полуосью A, вертикальной полуосью B.

Здесь:

X, Y — выражение типа INTEGER — координаты центра;

A, B — выражение типа WORD — горизонтальная и вертикальная полуоси.

П р и м е р. FILLELLIPSE (160,100,70, 40);

В результате будет нарисован эллипс с центром в точке (160,100), с горизонтальной полуосью A = 70 и вертикальной B = 40. Характер линии и цвет закраски предварительно устанавливаются программистом с помощью процедур SETLINESTYLE и SETCOLOR либо берутся по умолчанию.

27. Процедура **ELLIPSE (X,Y,ALFA,BETA,A,B)** рисует дугу эллипса.

Здесь:

X, Y — целые выражения — координаты центра эллипса;

ALFA, BETA — выражения типа WORD — начальный и конечный углы в градусах;

A, B — целые выражения типа WORD — горизонтальная и вертикальная полуоси эллипса.

П р и м е р. ELLIPSE (160,100,30,90,70,40);

В результате этого обращения будет нарисована дуга эллипса предыдущего примера от 30° до 90°.

Угол отсчитывается от положительного направления горизонтальной оси против часовой стрелки.

28. Процедура **SECTOR (X,Y,BETA,A,B)** рисует сектор эллипса.

Здесь:

X, Y — целые выражения — координаты центра;

ALFA, BETA — выражения типа WORD — начальный и конечный углы сектора;

A, B — выражения типа WORD — горизонтальная и вертикальная полуоси эллипса.

П р и м е р. SECTOR (160,100,30,90,70,40);

В результате обращения будет нарисован сектор эллипса от 30° до 90°.

Отсчет углов ведется против часовой стрелки от положительного направления горизонтальной оси эллипса.

29. Процедура **PIESLICE (X,Y,ALFA,BETA,R)** закрашивает сектор окружности.

Здесь:

X, Y — целые выражения — координаты центра окружности;

ALFA, BETA — выражения типа WORD — начальный и конечный углы дуги;

R — выражение типа WORD — радиус.

П р и м е р. PIESLICE (100,100,30,60,50);

Будет нарисован и закрашен сектор от 30° до 60° окружности с центром в (100,100) и радиусом 50.

Характер и цвет линии могут быть предварительно установлены программистом с помощью процедур SETLINESTYLE и SETCOLOR, а характер и цвет закраски — с помощью процедуры SETFILLPATTERN.

П р и м е р. Закрашивание четверти круга:

```
PROGRAM M1;
USES GRAPH,CRT;
VAR N,M:INTEGER;
```

```
BEGIN
  N:=0; M:=0;
  INITGRAPH (N,M,' ');
  CLEARDEVICE;
  PIESLICE (100,100,0,90,50);
  READLN
END.
```

30. Процедура **SETBKCOLOR (N)** задает цвет фона, соответствующий номеру цвета в палитре.

Здесь N — выражение типа WORD — номер цвета из диапазона 0..GETMAXCOLOR.

Верхняя граница значений N не меньше 3 и не больше 15. Вообще значение N — порядковый номер цвета в палитре. Нумерация начинается с нуля.

П р и м е р. SETBKCOLOR (0) задает в качестве фона первый цвет палитры.

31. Функция **GETBKCOLOR** — типа WORD — выдает номер цвета фона.

П р и м е р. M:=GETBKCOLOR;

Если фон был установлен в предыдущем примере, то целая переменная M будет иметь значение 0.

32. Функция **GETCOLOR** — типа WORD — выдает номер цвета, которым рисуются линии. Этот цвет — либо тот, что был принят по умолчанию, либо был установлен процедурой SETCOLOR.

33. Процедура **CLEARDEVICE** задает всем величинам значения, принятые по умолчанию. При этом отменяются все изменения, которые установил программист для этих величин, и чистится экран.

34. Функция **IMAGESIZE** (X_1,Y_1,X_2,Y_2) — типа WORD — выдает число байт, необходимое для спасения изображения из прямоугольника с диагональю, соединяющей левую верхнюю вершину (X_1,Y_1) с (X_2,Y_2). Стороны прямоугольника параллельны соответственно осям. Это число необходимо знать при обращении к процедуре GETIMAGE.

Здесь X_1, Y_1, X_2, Y_2 — целые выражения — координаты концов диагонали прямоугольника.

П р и м е р. S:=IMAGESIZE (10,10,30,30) выдает размер памяти в байтах, необходимый для запоминания картинки, расположенной на экране в прямоугольнике, заданном концами диагонали, со сторонами, параллельными осям координат.

35. Процедура **GETIMAGE (X_1,Y_1,X_2,Y_2,B)** спасает в буфере B изображение из прямоугольника с диагональю, соединяющей левую верхнюю вершину (X_1,Y_1) с (X_2,Y_2).

Здесь:

X_1, Y_1, X_2, Y_2 — целые выражения — координаты двух противоположных вершин прямоугольника;

B — переменная "без типа" — определяет область памяти для буфера.

Первая ячейка этого буфера содержит ширину прямоугольника плюс 1, т. е. $X_2 - X_1 + 1$, вторая ячейка — высоту прямоугольника плюс 1: $Y_2 - Y_1 + 1$. Далее в буфере B следует двоичное изображение картинки из заданного прямоугольника. Удобнее всего для B использовать динамическую переменную.

П р и м е р. Спасти картинку из области, заданной вершинами прямоугольника (10,10,30,30):

```
PROGRAM IMG;
VAR P:POINTER; I,J:INTEGER;
    S:WORD;
BEGIN I:=0; J:=0;
  INITGRAPH (I,J,' ');
  IF GRAPHRESULT<>GROK THEN HALT (1);
  . . .
  S:=IMAGESIZE (10,10,30,30);
  GETMEM (P,S); {Выделение памяти в S байт}
  GETIMAGE (10,10,30,30,P); {Спасение картинки}
  . . .
  CLOSEGRAPH
END.
```

36. Процедура **PUTIMAGE (X,Y,B.F)** осуществляет пересылку картинки из буфера B на экран в прямоугольник, левая верхняя вершина которого имеет координаты (X,Y).

В зависимости от значения флага F над двоичным изображением картинки производятся различные логические опера-

ции. Флаг F имеет следующие значения:

F = 0 — пересылка картинки без изменения;

F = 1 — пересылка картинки и одновременное выполнение операции XOR с соответствующими байтами экрана: B XOR SC, где B — пересылаемый байт, SC — байт, находящийся на экране в том месте, куда пересылается байт B;

F = 2 — пересылка и одновременное выполнение операции OR с соответствующими байтами экрана;

F = 3 — пересылка и одновременное выполнение операции AND с соответствующими байтами экрана;

F = 4 — пересылка и одновременное выполнение операции NOT над пересылаемыми байтами, т. е. получение "негатива".

Здесь:

X, Y — целые выражения координаты левого верхнего угла прямоугольника на экране;

B — переменная "без типа" — буфер, содержащий спасенную ранее картинку;

F — целая переменная типа WORD — флаг.

П р и м е р.

```
PROGRAM TEST14;
USES GRAPH,CRT;
VAR X1,X2,Y1,Y2,I,M,N:INTEGER; P:POINTER;
BEGIN
  CLRSCR; M:=0; N:=0;
  INITGRAPH (M,N,' ');
  X1:=10; Y1:=10;
  X2:=100; Y2:=100;
  LINE(X1,Y1,X2,Y2);
  I:IMAGESIZE(X1,Y1,X2,Y2,P,I);
  PUTIMAGE(X1+140,Y1,P,4)
END.
```

Программа рисует отрезок прямой, запоминает эту область экрана и пересылает запомненную информацию в виде негатива в другую область экрана.

37. Процедура **SETWRITEMODE (F)** рисует линию и одновременно может выполнять логическую операцию XOR между байтом

линии и байтом экрана в том месте, куда попадает байт линии. Определяет режим рисования флаг F.

Если F = 0, то просто рисуется линия, затирается та информация, которая попадает "под" линию.

Если F = 1, то на экран попадает результат логической операции XOR между байтами линии и байтами экрана, попадающими "под" линию.

38. Процедура **GETLINESETTINGS (B)** позволяет получить информацию о характере и цвете рисуемых линий.

Здесь B — переменная зарезервированного типа LINESETTINGSTYPE:

```
TYPE LINESETTINGSTYPE=RECORD
                        LINESTYLE:WORD;
                        PATTERN:WORD;
                        THICKNESS:WORD
                      END;
```

Первое поле LINESTYLE этой переменной содержит информацию о характере линии:

если B.LINESTYLE=0, то линия сплошная,

если B.LINESTYLE=1, то линия точечная,

если B.LINESTYLE=2, то линия штрих-пунктирная,

если B.LINESTYLE=3, то линия пунктирная,

если B.LINESTYLE=4, то характер линии определяется программистом.

Второе поле PATTERN содержит информацию о номере цвета, которым рисуется линия. Третье поле THICKNESS определяет толщину линии:

если B.THICKNESS=1, то линия нормальная;

если B.THICKNESS=2, то линия жирная.

П р и м е р. Программист может извлечь информацию о рисуемой линии с помощью следующего фрагмента программы:

```
VAR B:LINESETTINGSTYPE; I,J,K:INTEGER;
      . . .
BEGIN
  GETLINESETTINGS(B);
  I:=B.LINESTYLE;
  J:=B.PATTERN;
```

```
K:=B.THICKNESS;
. . .
END.
```

Переменная I будет равна одному из чисел 0, 1, 2, 3, 4. По значению I можно определить характер линии. Переменная J будет содержать номер цвета линии, а переменная K — информацию о толщине линии (1 — линия нормальная, 2 — жирная).

39. Процедура **SETVIEWPORT** (X_1, Y_1, X_2, Y_2, F) задает графическое окно его левой верхней вершиной (X_1, Y_1) и правой нижней (X_2, Y_2).

Здесь X_1, Y_1, X_2, Y_2 — целые выражения — координаты точек, причем $X_1 < X_2$, $Y_1 < Y_2$, F — булевская переменная.

В зависимости от значения F все линии, проводимые в режиме окна либо обрезаются по краю окна, либо продолжаются за пределами окна. Если F = TRUE, то происходит пропадание всех элементов создаваемого рисунка, которые выходят за пределы окна; если F = FALSE, то даже рисуя в режиме окна, можно выходить за пределы окна.

В системе предусмотрены специальные типизованные константы, которые можно указывать в качестве фактическнх параметров F при вызове процедуры SETVIEWPORT:

```
CONST
  CLIPON=TRUE;
  CLIPOFF=FALSE;
```

П р и м е р.

```
SETVIEWPORT (50,50,100,100,CLIPON);
```

или

```
F:=TRUE;
SETVIEWPORT (50,50,100,100,F);
```

Выполнятся одни и те же действия: установится окно с запретом выхода за его пределы.

Когда устанавливается окно, то автоматически начало координат переносится в левый верхний угол окна и отсчет ведется уже от нового начала.

Пример.

```
PROGRAM TEST15;
USES GRAPH,CRT;
VAR X1,X2,Y1,Y2,M,N:INTEGER;
    F:BOOLEAN;
BEGIN
  CLRSCR; M:=0; N:=0;
  INITGRAPH (M,N,' ');
  X1:=10; Y1:=10;
  X2:=100; Y2:=100;
  F:=TRUE;
  SETVIEWPORT (X1,Y1,X2,Y2,F);
  CIRCLE(20,0,20);
  LINE(-10,-10,100,100);
  READLN
END.
```

Программа рисует прямую и окружность в режиме окна. Так как F=TRUE, то части окружности и прямой, выходящие за пределы окна, пропадают.

Система резервирует специальный тип:

```
TYPE VIEWPORTTYPE = RECORD
                      X1,X2,Y1,Y2:WORD;
                      CLIP :BOOLEAN
                    END;
```

Это полезно знать при использовании процедуры GETVIEWSETTINGS.

40. Процедура **GETVIEWSETTINGS (B)** позволяет получить информацию об установленном окне.

Здесь:

B — переменная типа VIEWPORTTYPE;

B.X1 и B.Y1 — типа WORD — координаты левой верхней вершины окна;

B.X2 и B.Y2 — типа WORD — координаты правой нижней вершины окна;

B.CLIP — булевская переменная: если B.CLIP=TRUE, то окно обрезает все выходящие за нее части рисунков; если B.CLIP=FALSE, то рисунки не обрезаются.

41. Процедура **CLEARVIEWPORT** очищает окно, указатель текущей точки переходит в левый верхний угол окна.

42. Процедура **OUTTEXT (S)** выводит текст на экран, начиная с текущей позиции, в графическом режиме. Здесь S — строка с текстом.

П р и м е р.

```
OUTTEXT ( 'ЗАДАВАЙТЕ НОМЕР ЦВЕТА' );
```

или, что то же самое,

```
VAR S:STRING[50];
  . . .
S:= 'ЗАДАВАЙТЕ НОМЕР ЦВЕТА';
OUTTEXT(S);
  . . .
```

43. Процедура **OUTTEXTXY (X,Y,S)** выводит текст на экран, начиная с точки (X, Y).

Здесь X, Y — целые выражения — координаты точки, куда попадет текст; S — строка текста. Если строка слишком длинная, она будет обрезана.

П р и м е р. `OUTTEXTXY (1,10, 'TEST');`

Будет выведен текст TEST таким образом, что левый верхний угол первой буквы T будет иметь координаты (1, 10).

44. Функция **GRAPHRESULT** целого типа выдает код — целое число, по которому можно судить об успешности срабатывания последней графической операции; если была ошибка, то по коду можно установить ее причину.

Код	Сообщение
0(GROC)	Нет ошибки
−1	Не было INITGRAPH
−2	Нет таких аппаратных средств
−3	Нет файла драйвера
−4	Ошибка в файле драйвера
−5	Не хватает памяти
−6	Выход за границы памяти
−7	Ошибка при работе FLOODFILL
−8	Нет такого шрифта
−9	Шрифт не загружен в память — нет места
−10	Недопустимый режим для данного драйвера
−11	Затребована слишком большая область
−12	Ошибка ввода/вывода графики
−13	Нет такого файла шрифта
−14	Недопустимый номер шрифта

Текст сообщения можно получить с помощью функции GRAPHERRORMSG.

45. Функция **GRAPHERRORMSG (N)** — типа STRING — по коду N выдает текст сообщения. Здесь N — целое число из диапазона –14..0 Удобно воспользоваться этой функцией в программе.

Пример.

```
PROGRAM TEST16;
USES CRT,GRAPH;
CONST
  B:FILLPATTERNTYPE=($FF,$00,$FF.$00,$FF,$00,$FF,$00);
VAR NN,N,M:INTEGER;
BEGIN
  CLRSCR; NN:=3; N:=2; M:=2:
  INITGRAPH (N,M,' ');
  IF GRAPHRESULT<>0 THEN
    BEGIN
     WRITELN('ERROR:',GRAPHERRORMSG(GRAPHRESULT));
     READLN; HALT(1)
    END;
  ARC(100,50,0,90,50);
  CIRCLE (100,50,30);
  READLN
END
```

6. Функциональные клавиши. Очень удобным средством вмешательства в работу системы является использование функциональных клавиш. Опишем действия, выполняемые при нажатии каждой из них.

<F1> — вызов HELP. Причем система сама анализирует ситуацию и предоставляет нужную информацию. Так, если клавиша нажата при получении сообщения об ошибке, то на экране появляется предположение о возможной причине.

<ALT>/<F1> — повторение предыдущей информации HELP.

<CTRL>/<F1> — вывод информации о той процедуре или функции, на имени которой установлен курсор.

<F2> — спасение набранного файла на диск.

⟨CTRL⟩/⟨F2⟩ — стирание всех следов предыдущего сеанса отладки программы и подготовка к следующему сеансу.

⟨F3⟩ — загрузка файла с диска.

⟨ALT⟩/⟨F3⟩ — указание файла, который загружался ранее.

⟨CTRL⟩/⟨F3⟩ — вывод стека вызова процедур-функций.

⟨F4⟩ — выполнение программы до строки, указанной курсором, затем вход в пошаговый режим.

⟨CTRL⟩/⟨F4⟩ — калькулятор для вычисления и замены значений переменных. В окне EVALUATE можно набрать вычисляемое выражение и нажать ⟨ENTER⟩. В окне RESULT появится результат.

⟨F5⟩ — переключение на многооконный вывод и обратно, увеличение/уменьшение размера окна.

⟨ALT⟩/⟨F5⟩ — переключение на файл вывода и обратно, спасение экрана.

⟨F6⟩ — переключение на окно редактора и обратно, вывод результата.

⟨CTRL⟩/⟨F6⟩ — переключение на новый EDIT экран.

⟨ALT⟩/⟨F6⟩ — переключение на окно OUTPUT и обратно.

⟨F7⟩ — пошаговая отладка.

⟨CTRL⟩/⟨F7⟩ — занесение переменной в окно отладчика.

⟨F8⟩ — пошаговая отладка, не "шагающая" по процедурам и функциям.

⟨CTRL⟩/⟨F8⟩ — установка/стирание точки останова.

⟨F9⟩ — компиляция программы, создание файла.

⟨ALT⟩/⟨F9⟩ — компиляция программы.

⟨CTRL⟩/⟨F9⟩ — выполнение программы.

⟨F10⟩ — вход в основное меню.

Для опроса клавиатуры в версии 5.0 есть удобные средства. Функция READKEY типа CHAR выдает код нажатой клавиши, т. е. можно этот код получить оператором присваивания, например

```
C:=READKEY;
```

Если C = 0, то это значит, что нажата функциональная клавиша и ее код надо прочитать вторым обращением к READKEY (см. Приложение 5).

ПРИЛОЖЕНИЕ 4

СВОДКА КОМАНД РЕДАКТИРОВАНИЯ

Команды экранного редактора.

1: На символ влево: CTRL/S либо ←.
3: На символ вправо: CTRL/D либо →.
4: На слово влево: CTRL/A либо CTRL/←.
5: На слово вправо: CTRL/F либо CTRL/→.
6: На строку вверх: CTRL/E либо ↑.
7: На строку вниз: CTRL/X либо ↓.
8: Экран на строку вверх: CTRL/W.
9: Экран на строку вниз: CTRL/Z.
10: На страницу вверх: CTRL/R либо PgUp.
11: На страницу вниз: CTRL/C либо PgDn.
12: На начало строки: CTRL/Q CTRL/S либо Home.
13: На конец строки: CTRL/Q CTRL/D либо END.
14: На начало страницы: CTRL/Q CTRL/E либо CTRL/Home.
15: На конец страницы: CTRL/Q CTRL/X либо CTRL/End.
16: На начало файла: CTRL/Q CTRL/R либо CTRL/PgUp.
17: На конец файла: CTRL/Q CTRL/C либо CTRL/PgDn.
18: На начало блока: CTRL/Q CTRL/B.
19: На конец блока: CTRL/Q CTRL/K.
20: На последнюю позицию курсора: CTRL/Q CTRL/P.
21: Режим вставки/замены: CTRL/V или INS.
22: Вставка строки: CTRL/N.
23: Стирание целой строки: CTRL/Y.
24: Стирание до конца строки: CTRL/Q CTRL/Y.
25: Стирание слова справа от курсора: CTRL/T.
26: Стирание символа: CTRL/G.
27: Стирание символа слева от курсора:DEL.
29: Метка "начало блока": CTRL/K CTRL/B или F7.
30: Метка "конец блока": CTRL/K CTRL/K или F8.

31: Метка одиночного слова: CTRL/K CTRL/T.
32: Появление/исчезновение блока на экране: CTRL/K CTRL/H.
33: Копирование блока: CTRL/K CTRL/C.
34: Перепись блока: CTRL/K CTRL/V.
35: Стирание блока: CTRL/K CTRL/Y.
36: Чтение блока с диска: CTRL/K CTRL/R.
37: Запись блока на диск: CTRL/K CTRL/W.
38: Выход из редактора: CTRL/K CTRL/D или F10.
41: Запоминание строки: CTRL/Q CTRL/L.
42: Поиск фрагмента (фрагмент указывается в процессе диалога): CTRL/Q CTRL/F.
43: Поиск-замена фрагмента: CTRL/Q CTRL/A.

Параметры:

U — игнорировать отличия прописных и строчных букв;

W — найти слово целиком;

B — двигаться в обратном направлении;

G — искать заданный фрагмент во всем файле; при нахождении фрагмента система каждый раз переспрашивает: "Заменять — да/нет?";

N — система сама без переспрашивания заменяет все найденные одинаковые фрагменты.

Параметры можно задавать подряд в любом порядке.

44: Повторение последней команды поиска: CTRL/L.
45: Признак управляющего символа: CTRL/P.

Любую команду редактора можно прервать клавишей <ESC>.

ПРИЛОЖЕНИЕ 5

КОДЫ КЛАВИАТУРЫ

Шестнадцатеричные коды клавиатуры:

Клавиша	Код	Клавиша	Код	Клавиша	Код
ESC	01		0F	F1	3B
! 1	02	Q	10	F2	3C
@2	03	W	11	F3	3D
# 3	04	E	12	F4	3E
$4	05	R	13	F5	3F
%5	06	T	14	F6	40
^6	07	Y	15	F7	41
&7	08	U	16	F8	42
* 8	09	I	17	F9	43
(9	0A	O	18	F10	44
)0	0B	F	19	F11	D9
_ -	0C		1A	F12	DA
+ =	0D		1B		46
BS	0E	RETURN	1C	DGUP	49
CTRL	1D	? /	2B	-	4A
A	1E	Z	2C		4B
D	20	C	2D	5	4C
F	21	V	2E		4D
G	22	B	2F	+	4E
H	23	N	30	END	4F
J	24	M	31		50
K	25	,	32	PGDN	51
L	26	.	33	INS	52
; :	27	.	34	DEL	53
" '	28	? /	35	NUM LOCK	45
	29		36		
		PRTSCR	37		
Левый сдвиг		2A	ALT	38	
		39	HOME	47	
CAPSLOCK		3A		48	

Те символы, что не имееют стандартных кодов ASCII, генерируют расширенный код, состоящий из двух частей — предварительный код и собственно код.

Предварительный код, прочитанный с помощью функции READKEY, во всех расширенных кодах — это нуль. Расширен-

ный код также генерируют сочетания одновременно нажатых клавиш.

Ниже приводится таблица второй части кода таких нестандартных символов и сочетаний клавиш. (Коды даны в десятичной системе.)

Клавиши	Код	Клавиши	Код	Клавиши	Код
0	3	ALT/K	38	HOME	71
SHIFT TAB	15	ALT/L	39		72
ALT/Q	16	ALT/Z	44	PGUP	73
ALT/W	17	ALT/X	45		75
ALT/E	18	ALT/C	46		77
ALT/R	19	ALT/V	47	END	79
ALT/T	20	ALT/B	48		80
ALT/Y	21	ALT/N	49	PGDN	81
ALT/U	22	ALT/M	50	INS	82
ALT/I	23	F1	59	DEL	83
ALT/O	24	F2	60	SHIFT/F1	84
ALT/P	25	F3	61	SHIFT/F2	85
ALT/A	30	F4	62	SHIFT/F3	86
ALT/S	31	F5	63	SHIFT/F4	87
ALT/D	32	F6	64	SHIFT/F5	88
ALT/F	33	F7	65	SHIFT/F6	89
ALT/G	34	F8	66	SHIFT/F7	90
ALT/H	35	F9	67	SHIFT/F8	91
ALT/I	36	F10	68	SHIFT/F9	92
SHIFT/F10	93	CTRL/PRTSC	114	SHIFT/F11	
CTRL/F1	94	CTRL/	115	SHIFT/F12	
CTRL/F2	95	CTRL/	116	CTRL/F11	
CTRL/F3	96	CTRL/END	117	CTRL/F12	
CTRL/F4	97	CTRL/PGDN	118	ALT/F11	
CTRL/F5	98	CTRL/HOME	119	ALT/F12	
CTRL/F6	99	ALT/1	120		
CTRL/F7	100	ALT/2	121		
CTRL/F8	101	ALT/3	122		
CTRL/F9	102	ALT/4	123		
CTRL/F10	103	ALT/5	124		
ALT/F1	104	ALT/6	125		
ALT/F2	105	ALT/7	126		
ALT/F3	106	ALT/8	127		
ALT/F4	107	ALT/9	128		
ALT/F5	108	ALT/0	129		
ALT/F6	109	ALT/-	130		
ALT/F7	110	ALT/=	131		
ALT/F8	111	CTRL/PGDN	132		
ALT/F9	112	F11	133		
ALT/F10	113	F12	134		

ПРИЛОЖЕНИЕ 6

ПРОЦЕДУРЫ И ФУНКЦИИ ВЕРСИИ 5.0

В данном приложении приведены наиболее часто используемые стандартные процедуры и функции версии 5.0. Они перечислены в алфавитном порядке. Те из них, что описаны в основном тексте книги, снабжены ссылками на соответствующие страницы, остальные процедуры и функции описаны прямо в приложении.

Здесь описывается назначение каждой процедуры либо функции и, как правило, приводится ее заголовок, из которого пользователь получает информацию о типе и виде параметров, типе результата — для функции.

Напомним, что если параметр предваряется ключевым словом VAR, то соответствующим фактическим параметром может быть только переменная; если слово VAR отсутствует, то соответствующим фактическим параметром может быть любое выражение соответствующего типа (в том числе и константа, и переменная).

Обращаем внимание читателя, что для многих процедур/функций необходимо указывать конкретный модуль в разделе USES. Если этого не сделать, то система не опознает стандартную процедуру/функцию.

1. Функция ABS(X) — см. с. 19.
2. Функция ADDR(X) — типа "указатель" — выдает адрес переменной произвольного типа.
3. Процедура APPEND(F) открывает текстовый файл F для дозаписи информации. Заголовок:
   PROCEDURE APPEND(VAR F:TEXT);
4. Процедура ARC(X,Y,ALFA,BETA,R) — см. с. 169. В программе требуется указать USES GRAPH;

5. Функция ARCTAN(X) выдает арктангенс угла X, заданного в радианах. Заголовок:
   FUNCTION ARCTAN(X:REAL):REAL;
6. Процедура ASSIGN(F,'имя') — см. с. 150, 151.
7. Функция ASSIGNCRT(F) — см. с. 151. В программе требуется указать USES CRT;
8. Процедура BAR(X_1,Y_1,X_2,Y_2) — см. с. 166. В программе требуется указать USES GRAPH;
9. Процедура BAR3D(X_1,Y_1,X_2,Y_2,D,F1) — см. с. 167. В программе требуется указать USES GRAPH;
10. Процедура BLOCKREAD(F,B,N,I) — см.с. 153.
11. Процедура BLOCKWRITE(F,B,N,I) — см. с. 153.
12. Процедура CHDIR(DR) меняет текущую директорию на директорию, указанную в параметре, т. е. в DR задается путь к новому подкаталогу либо просто новый дисковод. Заголовок:
    CHDIR(DR:STRING);
13. Функция CHR(N) выдает N-й символ. Заголовок:
    FUNCTION CHR(X:BYTE):CHAR;
14. Процедура CIRCLE(X,Y,R) — см. с. 168.
15. Процедура CLEARDEVICE — см. с. 172.
16. Процедура CLEARVIEWPORT очищает экран (окно), вызывает процедуру BAR и устанавливает указатель в точку (0,0). В программе требуется указать USES GRAPH;
17. Процедура CLOSE(F) — см. с. 150.
18. Процедура CLOSEGRAPH — см. с. 160.
19. Процедура CLREOL стирает символы от курсора до конца текущей строки. В программе требуется указать USES CRT;
20. Процедура CLRSCR очищает экран и перемещает курсор в верхний левый угол. В программе требуется указать USES CRT;
21. Функция CONCAT(S_1,S_2,...) — типа STRING — сцепляет строки S_1,S_2,... и выдает суммарную строку.
22. Функция COPY(S,I,N) — см. с. 145.
23. Функция COS(X) — см. с. 20.
24. Функция CSEG выдает текущее значение регистра CS — адрес сегмента программы, содержащей обращение к CSEG.

Заголовок:

FUNCTION CSEG: WORD;

25. Процедура DEC(X,N) заменяет значение X на (X – N), а в случае DEC(X) заменяет значение X на (X – 1). Заголовок:

    PROCEDURE DEC(VAR X,N:INTEGER);

    X — любого скалярного типа (но не REAL).

26. Процедура DELAY(T) включает режим ожидания на T мсек. Заголовок:

    PROCEDURE DELAY(T:WORD);

    В программе требуется указать USES CRT;

27. Процедура DELETE(S,I,N) — см. с. 146.

28. Процедура DELLINE стирает строку, на которой находится курсор.

29. Процедура DETECTGRAPH(D,R) задает значение D (драйвера) и R (режима), определяя их по конфигурации компьютера.

Значение	1	2	3	4	5	6
Драйвер	CGA	MCGA	EGA	EGA64	EGAMONO	RESERVED
Значение	7	8	9	10		
Драйвер	HERCMONO	ATT400	VGA	PC3270		

    Если D=0, это означает требование автоматического распознавания драйвера. Заголовок:

    PROCEDURE DETECTGRAPH(VAR D,R:INTEGER);

30. Функция DISKSIZE(N) выдает системе число свободных байт диска на дисководе с номером N: N = 0 — рабочий дисковод, N = 1 — дисковод A, N = 2 — дисковод B, N = 3 — дисковод C и т. д. Заголовок:

    FUNCTION DISKSIZE (N:WORD):LONGINT;

31. Процедура DISPOSE (P) — см. с. 154.

32. Функция DOSEXITCODE дает код завершения (в младшем байте). Заголовок:

    FUNCTION DOSEXITCODE:WORD;

33. Функция DSEG дает текущее значение регистра DS. Заголовок:

    FUNCTION DSEG:WORD;

34. Процедура ELLIPS(X,Y,ALFA,BETA,A,B) — см. с. 170. В программе необходимо указать USES GRAPH;

35. Функция EOF(F) — см. с. 150.
36. Функция EOLN(T) — см. с. 152.
37. Процедура ERASE(F) стирает на диске файл, с которым связана файловая переменная F.
38. Процедура EXIT выполняет выход из текущего олока программ.
39. Функция EXP(X) дает e^x. Заголовок:
    FUNCTION EXP(X:REAL):REAL;
40. Функция FILEPOS(F) — типа INTEGER — выдает номер компоненты, на которую установлен указатель файла.
41. Функция FILESIZE(F) — см. с. 150.
42. Процедура FILLELLIPS(X,Y,A,B) — см. с. 170. В программе необходимо указать USES GRAPH;
43. Процедура FLOODFILL(X,Y,B) — см. с. 167. В программе необходимо указать USES GRAPH;
44. Процедура FLUSH(F) сбрасывает на диск буфер файла F, независимо от степени его заполненности.
45. Функция FRAC(X) дает дробную часть аргумента X. Заголовок:
    FUNCTION FRAC(X:REAL):REAL;
46. Процедура FREEMEM(P,N) — см. с. 154.
47. Процедура GETARCCOORDS — см. с. 169. В программе необходимо указать USES GRAPH;
48. Функция GETBKCOLOR — см. с. 172. В программе необходимо указать USES GRAPH;
49. Функция GETCOLOR дает номер цвета вычерчиваемых линий. Заголовок:
    FUNCTION GETCOLOR:WORD;
    В программе необходимо указать USES GRAPH;
50. Процедура GETDATE(Y,M,D,W) помещает в параметры год, месяц, число, день недели соответственно. Заголовок:
    PROCEDURE GETDATE(VAR Y,M,D,W:WORD);
51. Процедура GETDIR(N,S) выдает директорию с дисковода номер N. Здесь значение N = 0 означает текущий дисковод, N = 1 — дисковод A, N = 2 — дисковод B и т. д. Заголовок:
    PROCEDURE GETDIR(N:BYTE,VAR S:STRING);

52. Функция GETDRIVERNAME выдает имя графического драйвера. Заголовок:
FUNCTION GETDRIVERNAME:STRING;
В программе необходимо указать USES GRAPH;
53. Процедура GETFATTR(F,A) засылает в переменную A статус файла F:
A=$01 — только чтение,
A=$02 — скрытый файл,
A=$04 — системный файл,
A=$08 — имя тома,
A=$10 — каталог,
A=$20 — архивированный файл,
A=$3F — прочие файлы.
Файл F не должен быть открыт. Заголовок:
PROCEDURE GETFATTR(VAR F; VAR A:BYTE);
54. Процедура GETFTIME(F,T) заносит в переменную T время последней записи в файл F. Заголовок:
PROCEDURE GETFTIME(VAR F, VAR T);
Файл F не должен быть открыт.
55. Процедура GETIMAGE(X_1,Y_1,X_2,Y_2,B) — см. с. 173. В грамме необходимо указать USES GRAPH;
56. Процедура GETLINESETTINGS — см. с. 175. В программе необходимо указать USES GRAPH;
57. Процедура GETMAXCOLOR — см. с. 161. В программе необходимо указать USES GRAPH;
58. Функция GETMAXMODE выдает максимальный номер режима для имеющегося графического драйвера. Заголовок:
FUNCTION GETMAXMODE:WORD;
В программе необходимо указать USES GRAPH;
59. Функция GETMAXX — см. с. 163. В программе необходимо указать USES GRAPH;
60. Функция GETMAXY — см. с. 163. В программе необходимо указать USES GRAPH;
61. Процедура GETMEM(P,S) — см. с. 154.
62. Процедура GETPIXEL(X,Y) — см. с. 164. В программе необходимо указать USES GRAPH;
63. Процедура GETTIME(H,M,HS) засылает в параметры текущее

время: часы, минуты, секунды, сотые секунды. Заголовок:
PROCEDURE GETTIME(VAR H,M,S,HS:WORD);

64. Процедура GETVIEWSETTINGS (V) — см. с. 177. В программе необходимо указать USES GRAPH;
65. Функция GETX — см. с. 163. В программе необходимо указать USES GRAPH;
66. Функция GETY — см. с. 163. В программе необходимо указать USES GRAPH;
67. Процедура GOTOXY(X,Y) перемещает курсор в точку с координатами X,Y, где X,Y — целые выражения. В программе необходимо указать USES CRT;
68. Процедура GRAPHDEFAULTS востанавливает параметры работы с графикой, принятые по умолчанию. В программе необходимо указать USES GRAPH;
69. Функция GRAPHERRORMSG (I) — см. с. 179. В программе необходимо указать USES GRAPH;
70. Функция GRAPHRESULT — см. с. 178. В программе необходимо указать USES GRAPH;
71. Процедура HALT(I) прерывает работу программы и передает управление операционной системе. Заголовок:
PROCEDURE HALT (I:WORD);
I — необязательный параметр-код выхода из программы.
72. Функция HI(X) — см. с. 157.
73. Процедура HIGHVIDEO устанавливает режим подсветки выводимых символов. В программе необходимо указать USES CRT;
74. Процедура IMAGESIZE (X_1,Y_1,X_2,Y_2) — см. с. 172. В программе необходимо указать USES GRAPH;
75. Процедура INC(X,N) заменяет значение переменной X на (X + N). В случае обращения INC(X) принимается по умолчанию N = 1. Заголовок:
PROCEDURE(VAR X;N:LONGINT);
X — скалярного типа (но не REAL).
76. Процедура INITGRAPH(D,R,K) — см. с. 159. В программе необходимо указать USES GRAPH;
77. Процедура INSERT(S1,S2,I) — см. с. 147.

78. Процедура INSLINE вставляет пустую строку с позиции, отмеченной курсором. В программе необходимо указать USES CRT;
79. Функция INT(X) выдает целую часть аргумента в виде вещественного числа. Заголовок:
FUNCTION INT (X:REAL):REAL;
80. Функция IORESULT — см. с. 153.
81. Функция KEYPRESSED имеет тип BOOLEAN и принимает значение TRUE, если нажата какая-либо клавиша на клавиатуре. В программе необходимо указать USES CRT;
82. Функция LENGTH (S) — см. с. 145.
83. Процедура LINE(X_1,Y_1,X_2,Y_2) — см. с. 162. В программе необходимо указать USES GRAPH;
84. Процедура LINEREL(DX,DY) — см. с. 162. В программе необходимо указать USES GRAPH;
85. Процедура LINETO(X,Y) — см. с. 162. В программе необходимо указать USES GRAPH;
86. Функция LN(X) выдает значение ln(X). Заголовок:
FUNCTION LN(X:REAL):REAL;
87. Функция LO(X) выдает младший байт X. Заголовок:
FUNCTION(X:INTEGER):BYTE;
88. Процедура LOWVIDEO ликвидирует режим подсветки символов. В программе необходимо указать USES CRT;
89. Процедура MARK(P) — см. с. 154.
90. Функция MAXAVAIL — см. с. 154.
91. Функция MEMAVAIL — см. с. 154.
92. Процедура MOVEREL(DX,DY) — см. с. 163. В программе необходимо указать USES GRAPH;
93. Процедура MOVETO(X,Y) — см. с. 163. В программе необходимо указать USES GRAPH;
94. Процедура NEW(P) — см. с. 154.
95. Процедура NOSOUND выключает звук. В программе необходимо указать USES CRT;
96. Функция ODD(X) выдает значение TRUE для нечетного X и FALSE — для четного. Заголовок:
FUNCTION ODD(X:LONGINT):BOOLEAN;
97. Функция OFS(X) — см. с. 154.
98. Функция ORD(X) — см. с. 23.

99. Процедура OUTTEXT(S) — см. с. 178. В программе необходимо указать USES GRAPH;
100. Процедура OUTTEXTXY(X,Y,S) — см. с. 178. В программе необходимо указать USES GRAPH;
101. Процедура OVRCLEARBUF очищает оверлейный буфер. В программе необходимо указать USES OVERLAY;
102. Функция OVRGETBUF выдает размер оверлейного буфера. Заголовок:
FUNCTION OVRGETBUF:LONGINT;
В программе необходимо указать USES OVERLAY;
103. Процедура OVRINT(S) открывает оверлейный файл и инициализирует работу с оверлеями. В программе необходимо указать USES OVERLAY;
104. Процедура OVRINITEMS загружает оверлейный файл в расширенную память.
105. Процедура OVRSETBUF(N) задает размер N оверлейного буфера. Обращение к ней должно предшествовать обращению к INITGRAPH и работе с динамическими переменными. В программе необходимо указать USES OVERLAY;
106. Функция PI дает значение PI = 3.1415926535897932385. Заголовок:
FUNCTION PI:REAL;
107. Процедура PIESLICE(X,Y,ALFA,BETA,R) — см. с. 171. В программе необходимо указать USES GRAPH;
108. Функция POS(S1,S) — см. с. 147.
109. Функция PRED(X) — см. с. 23.
110. Функция PTR(S,D) — см. с. 154.
111. Процедура PUTIMAGE(X_1,Y_1,X_2,Y_2,D,F) — см. с. 173. В программе необходимо указать USES GRAPH;
112. Функция RANDOM(N) выдает случайное число из интервала (0,N), где N — целое выражение типа WORD, тип результата — INTEGER. Если обратиться к RANDOM без параметра, то выдается случайное число типа REAL из интервала (0,1). Случайные числа равномерно распределены на соответствующих интервалах.
113. Процедура RANDOMIZE запускает генератор случайных чисел. Случайное число получается в системной переменной RANDSEED.

114. Процедура READ — см. с. 150.
115. Функция READKEY — см. с. 180.
116. Процедура READLN — см. с. 151.
117. Процедура RECTANGLE(X_1,Y_1,X_2,Y_2) — см. с. 164. В программе необходимо указать USES GRAPH;
118. Процедура RELEASE(P) — см. с. 154.
119. Процедура RENAME (F, 'имя') заменяет прежнее имя файла F на 'имя'. F — переменная-файл, 'имя' — строка.
120. Процедура RESET(F) — см. с. 150, 153.
121. Процедура RESTORECRTMODE восстанавливает тот режим, в котором находился экран до выполнения процедуры INITGRAPH. В программе необходимо указать USES GRAPH;
122. Процедура REWRITE(F) — см. с. 150, 153.
123. Функция ROUND(X) — см. с. 19.
124. Процедура SECTOR(X,Y,ALFA,BETA,A,B) — см. с. 171.
125. Процедура SEEK(F,N) — см. с. 150.
126. Функция SEEKEOF(F) — см. с. 152.
127. Функция SEEKEOLN(F) — см. с. 152.
128. Функция SEG(X) — см. с. 154.
129. Процедура SETALLPALETTE(P) изменяет все цвета палитры в соответствии с параметром-переменной P. Переменная P — без типа — содержит N+1 байт, где в I-м байте указывается число цветов в палитре, а каждый следующий байт содержит номер цвета: 0 — черный, 1 — синий, 2 — зеленый, 3 — бирюзовый, 4 — красный, 5 — малиновый, 6 — коричневый, 7 — светло-серый, 8 — темно-серый, 9 — светло-голубой, 10 — светло-зеленый, 11 — светло-бирюзовый, 12 — розовый, 13 — светло-малиновый, 14 — желтый, 15 — белый, (–1) — прежний цвет палитры. Число цветов в палитре определяется графическим драйвером и режимом и потому может быть меньше пятнадцати. В программе необходимо указать USES GRAPH;
130. Процедура SETBKCOLOR(N) — см. с. 172. В программе необходимо указать USES GRAPH;
131. Процедура SETCOLOR(N) — см. с. 161. В программе необходимо указать USES GRAPH;

132. Процедура SETDATE(Y,M,D,W) устанавливает в системе дату: год, месяц, число, день недели. Заголовок:
PROCEDURE SETDATE(VAR Y,M,D,W:WORD);
133. Процедура SETATTR(F,A) устанавливает статус файла F. Заголовок:
PROCEDURE SETATTR(VAR F; VAR A:BYTE);
134. Процедура SETFILLPATTERN — см. с. 165. В программе необходимо указать USES GRAPH;
135. Процедура SETFTIME(F,T) заносит время T для F. Впоследствии это время можно прочесть обращением к процедуре GETFTIME(F,T).
136. Процедура SETGRAPHMODE(N) устанавливает режим с номером N. Эта процедура может также использоваться для переключения на графический режим с текстового. В программе необходимо указать USES GRAPH;
137. Процедура SETLINESTYLE(M,N,K) — см. с. 161. В программе необходимо указать USES GRAPH;
138. Процедура SETTEXTBUF — см. с. 151.
139. Процедура SETTIME(H,M,S,HS) устанавливает системное время: часы, минуты, секунды, сотые секунды. Заголовок:
PROCEDURE SETTIME(VAR H,M,S,HS:WORD);
140. Процедура SETVIEWPORT — см. с. 176. В программе необходимо указать USES GRAPH;
141. Процедура SETWRITEMODE — см. с. 174. В программе необходимо указать USES GRAPH;
142. Функция SIN(X) выдает sin(X), где X — угол в радианах. Заголовок:
FUNCTION SIN(X:REAL):REAL;
143. Функция SIZEOF — см. с. 158.
144. Процедура SOUND(F) включает звук. F — частота звука в герцах. Заголовок:
PROCEDURE SOUND (F:WORD);
В программе необходимо указать USES CRT;
145. Функция SQR(X) выдает квадрат аргумента X. Тип функции совпадает с типом X: может быть REAL или INTEGER.
146. Функция SQRT(X) выдает заголовок:
FUNCTION SQRT(X:REAL):REAL;

147. Процедура STR(X,S) — см. с. 148.
148. Функция SUCC(X) — см. с. 23.
149. Функция SWAP(X) — см. с. 157.
150. Процедура TEXTBACKGROUND(N) устанавливает цвет фона для текстового режима. Заголовок:
PROCEDURE TEXTBACKGROUND(N:BYTE);
В программе необходимо указать USES CRT;
151. Процедура TEXTCOLOR(N) устанавливает цвет символов для текстового режима. Заголовок:
PROCEDURE TEXTCOLOR(N:BYTE);
В программе необходимо указать USES CRT;
152. Процедура TEXTMODE(R) устанавливает текстовый режим. Параметр R — типа INTEGER — задает номер режима. Ниже приведем таблицу номеров и соответствующих им режимов работы экрана.

Значение R	Режим экрана
0	40×25 символов, черно-белый на цветном адаптере.
1	40×25 символов, цветной на цветном адаптере.
2	80×25 символов, черно-белый на цветном адаптере.
3	80×25 символов, цветной на цветном адаптере.
7	80×25 символов, черно-белый на монохроматическом адаптере.
256	для адаптера EGA/VGA.
C040	для совместимости с версией 3.0 — 40 символов в строке.
C080	для совместимости с версией 3.0 — 80 символов в строке.
LASTMODE	восстановление предыдущего текстового режима.

З а м е ч а н и е. В программах версии 3.0 заменить обращение TEXTMODE на TEXTMODE (LASTMODE). Заголовок:
PROCEDURE TEXTMODE (R:INTEGER);
153. Функция TRUNC(X) — см. с. 19.

154. Процедура TRUNCATE(F) ставит маркер "конец файла" на то место, куда установлен указатель файла.
155. Функция UPCASE(C) выдает символ C в виде прописной буквы. Заголовок:
FUNCTION UPCASE(C:CHAR):CHAR;
156. Процедура VAL(S,N) — см. с. 148.
157. Функция WHEREX — типа INTEGER — выдает координату X текущего положения курсора. В программе требуется указать USES CRT;
158. Функция WHEREY — типа INTEGER — выдает координату Y текущего положения курсора. В программе требуется указать USES CRT;
159. Процедура WINDOW(X_1,Y_1,X_2,Y_2) устанавливает окно. X_1,Y_1,X_2,Y_2 — целые выражения — координаты левого верхнего и правого нижнего угла окна. В программе требуется указать USES CRT;
160. Процедура WRITE — см. с. 150.
161. Процедура WRITELN — см. с. 151.

ПРИЛОЖЕНИЕ 7

КРАТКИЕ СВЕДЕНИЯ О NORTON COMMANDER

Программа NORTON COMMANDER (для краткости будем называть в дальнейшем NORTON) предоставляет пользователю большие удобства при небольших затратах памяти: 100 килобайт на диске и 150 килобайт в оперативной памяти.

Для работы с NORTON необходимо на диске иметь файлы:

NC.EXE, NC.EXT, NC.INI, NC.MNU, NCSMALL.EXE.

NORTON опекает пользователя в течение всего сеанса работы, предоставляя возможность простыми средствами выполнять различные запросы к системе.

Так директория текущего диска появляется на экране в рамке "сама", без каких-либо усилий пользователя.

По этой директории можно перемещать индикатор с помощью клавиш <↓>, <→>, <↑>, <←>.

В нижнюю строку рамки выводится информация о файле, на котором находится индикатор: имя, размер в байтах, дата создания.

Если подвести индикатор к файлу и нажать <ENTER>, то файлы с расширениями BAT, COM и EXE будут запущены на выполнение.

Если индикатор был установлен в корневой директории на какой-либо поддиректории, то при нажатии <ENTER> произойдет переход в эту поддиректорию. Имя ее будет выведено на верхней строке рамки.

Чтобы пользователь мог отличить на экране поддиректории от файлов, имена поддиректорий выводятся заглавными буквами, а имена файлов — малыми.

Программа NORTON позволяет выполнять некоторые манипуляции сразу с группой файлов. Для того, чтобы образовать из нескольких файлов группу, следует подвести индикатор к каждому из этих файлов и нажать клавишу <INS>.

Для исключения файла из группы достаточно подвести к нему индикатор и нажать клавишу <INS>.

С группой файлов можно выполнить следующие действия.

1. Для копирования нажать клавишу <F5>, на экране появляется сообщение

copy n files to
(копирование n файлов в ...)

Компьютер ожидает указаний, куда копировать группу файлов: либо на другой дисковод (например, a:), либо в другую поддиректорию (например, SYSTEM).

2. Для переименования или переписи надо нажать клавишу <F6>. Компьютер запрашивает, как переименовать либо в какие файлы переслать.

3. Для стирания группы файлов надо нажать клавишу <F8>. Компьютер не торопится выполнять приказ, переспрашивает

Do you wish to delete <имя файла>
(Хотите ли Вы стереть <имя файла>)

Delete	Cancel
(стереть	отменить)

Индикатор установлен на Delete, так что при нажатии <ENTER> — файл сотрется. Если пользователь раздумал стирать данный файл, то ему следует перевести индикатор на Cancel и нажать <ENTER>.

Копировать, переименовывать и стирать можно, конечно, и одиночный файл. В этом случае надо на нем установить индикатор и нажать соответственно <F5>, <F6> или <F8>. Диалог с компьютером и в этом случае будет подобен описанному для группы файлов.

Очень удобными являются и следующие возможности, предоставляемые программой NORTON COMMANDER.

4. Для получения дополнительной информации о назначении ряда клавиш служит клавиша <F1> (HELP).

5. Можно вывести на экран содержимое любого файла, подведя к нему на экране индикатор и нажав <F3>. Возврат в NORTON — нажатие <ESC>.

6. Для выбора файла на редактирование следует на экране подвести к нему индикатор и нажать клавишу <F4>. Для создания нового текстового файла надо нажать <F4>/<Shift>.

7. Для создания поддиректории следует нажать <F7>.

8. Для того, чтобы провести поиск по всем директориям, надо нажать одновременно <ALT>/<F7>. Компьютер запросит имя файла. Набрав имя, расширение и нажав <ENTER>, получите на экране информацию:

директорию, имя файла, размеры файла, дату создания.

Если имя файла не известно точно, можно использовать символы * и ?.

9. Предыдущую набранную команду можно вывести клавишами <CTRL>/<E>. Повторное нажатие этой комбинации выведет команду, предыдущую данной и т. д. Чтобы перейти к следующей команде из ранее набранных, надо нажать <CTRL>/<X>.

10. Можно на экране вывести и директорию другого диска, не текущего. Для этого надо нажать <ALT>/<F1> или <ALT>/<F2>.

Клавишами <→>, <←> переводят индикатор на нужную букву и нажимают <ENTER>.

Если нажаты клавиши <ALT>/<F1>, то директория выводится на левой половине экрана, если <ALT>/<F2> — на правой.

11. Для перевода индикатора на другую рамку с директорией служит клавиша <TAB>. Чтобы убрать вторую рамку, достаточно нажать одновременно клавиши <CTRL>/<P>.

12. Вывести/убрать левую рамку можно клавишами <CTRL>/<F1>; вывести/убрать правую рамку — клавишами <CTRL>/<F2>. Чтобы убрать/восстановить обе рамки, надо нажать <CTRL>/<0>. Убирают рамки в том случае, если экран нужен для других целей.

13. Информацию о текущем диске, дисководе, директории можно получить через меню NORTONa. Чтобы туда попасть, надо нажать <F9>. Появится меню, состоящее из полей:

LEFT, FILES, COMMANDS, OPTIONS, RIGH.

Индикатор следует перевести на LEFT (левая рамка) или RIGHT (правая рамка) и нажать <ENTER>. Появится вертикально расположенное меню. На нем надо подвести индикатор к INFO и нажать <ENTER>. На экране появится информация об объеме оперативной памяти компьютера, дисковода, количество файлов, их суммарный размер в текущей директории.

14. Имена файлов, выдаваемые на экран программой NORTON, обычно упорядочены по алфавиту. Но, по желанию пользователя, их можно упорядочивать по другому признаку. Для этого, как и в п. 13, надо перейти в меню NORTONa (<F9>), затем перевести индикатор на LEFT либо RIGHT, нажать <ENTER>. В появившемся вертикальном меню есть поля, отвечающие за сортировку файлов:

NAME — по именам,
EXTENSION — по расширениям,
TIME — по убыванию даты записи,
SIZE — по убыванию размера.

Переведя индикатор на соответствующее поле и нажав <ENTER>, пользователь задаст желаемый способ упорядочивания файлов при выводе директории на экран.

15. Программа NORTON COMMANDER открыта для включения новых возможностей. Пользователь может сам создать собственное меню, записав команды в файл NC.MNU. Так можно задействовать для своих нужд функциональные клавиши. Но это не означает, что функциональные клавиши будут перепрограммированы: их действие останется прежним в рамках NORTON.

Структура файла NC.MNU следующая: первая строка, относящаяся к какой-либо пользовательской команде, должна набираться с первой позиции и содержать строку меню, т. е. просто информацию о команде. Начиная со второй строки записываются команды DOS, реализующие этот пункт меню. В этих строках в первой позиции должен стоять пробел.

П р и м е р.

```
PCT — ВЫЗОВ PC TOOLS
 CD..
 CD TOOLS
 PCTOOLS
```

Если выполнение этой команды меню возложить на функциональную клавишу, то в первой строке с первой позиции набирается обозначение этой клавиши, затем ставится двоеточие и набирается строка меню.

После того, как пользователь записал свои команды в файл NC.MNU, он может их использовать простым способом: нажать клавишу <F2> — появится созданное им меню, состоящее из первых строк команд.

Если задействованы функциональные клавиши, то можно работать через них; если — нет, то надо перевести индикатор меню на нужную команду и нажать <ENTER>.

17. Программа NORTON позволяет вывести в виде меню все команды, которые набирал пользователь во время сеанса работы. По этому меню можно, перемещая индикатор, выполнить любую из предыдущих команд. Такую возможность предоставляет комбинация клавиш <ALT>/<F8>.

Мы привели только небольшую часть возможностей NORTON/COMMANDER. Предлагаем читателю самому познакомиться ближе с этой полезной сервисной программой, исследовав все пункты основного меню NORTON COMMANDER (клавиша <F9>), переводя индикатор последовательно с одного поля на другое и нажимая <ENTER>.

Более подробно на русском языке NORTON COMMANDER описан в [29].

З а м е ч а н и е. Программа NORTON COMMANDER не накладывает никаких ограничений на использование команд MS DOS.

СПИСОК ЛИТЕРАТУРЫ

1. Абрамов В. Г., Трифонов Н. П., Трифонов Г. Н. Введение в язык Паскаль: Учеб. пособие. — М.: Наука, 1988.
2. Абрамов С. И., Зима Е. В. Начала программирования на языке Паскаль. — М.: Наука, 1987.
3. Алкок Д. Язык Паскаль в иллюстрациях / Пер. с англ.; Под ред. А. Б. Ходулева. — М.: Мир, 1991.
4. Белецкий Я. Турбо-Паскаль с графикой для персональных компьютеров / Пер. с польск. — М.: Машиностроение, 1991.
5. Боон К. Паскаль для всех / Пер. с гол.; Под ред. Н. И. Слепова. — М.: Энергоатомиздат, 1988.
6. Бутомо И. Д., Самочадин А. В., Усанова Д. В. Программирование на алгоритмическом языке Паскаль для микро-ЭВМ: Учеб. пособие. — Л.: Изд-во ЛГУ, 1985.
7. Вальвачев А. В., Крисевич В. С. Программирование на языке Паскаль для персональных ЭВМ ЕС: Справ. пособие. — Минск: Вышэйш. шк., 1989.
8. Васюкова Н. Д., Тюляева В. В. Практикум по основам программирования. Язык Паскаль: Учеб. пособие. — М.: Высш. шк., 1991.
9. Вирт Н. Алгоритмы + структуры данных = программы/ Пер. с англ.; Под ред. Д. Б. Подшивалова. — М.: Мир, 1985.
10. Вирт Н. Алгоритмы и структуры данных / Пер. с англ. — М.: Мир, 1989.
11. Вирт Н. Систематическое программирование. Введение / Пер. с англ.; Под ред. Ю. М. Баяковского. — М.: Мир, 1977.

12. Вирт Н. Язык программирования Паскаль / Пер. с англ.; Отв. ред. И. В. Романовский. — Л.: Изд-во ЛГУ, 1974.
13. Грогоно П. Программирование на языке Паскаль / Пер. с англ.; Под ред. Д. Б. Подшивалова. — М.: Мир, 1982.
14. Грэхем Р. Практический курс языка Паскаль для микро-ЭВМ / Пер. с англ. — М.: Радио и связь, 1986.
15. Дедков А. Ф. Абстрактные типы данных в языке АТ-Паскаль. — М.: Наука, 1989.
16. Джонс Ж., Харроу К. Решение задач в системе Турбо Паскаль / Пер. с англ. — М.: Финансы и статистика, 1991.
17. Йенсен К., Вирт Н. Паскаль. Руководство для пользователя / Пер. с англ. — М.: Финансы и статистика, 1989.
18. Касьянов В. Н. Основы программирования на языке Паскаль: Учеб. пособие. — Новосибирск: Изд-во НГУ, 1987.
19. Кернеган Б., Плоджер Ф. Инструментальные средства программирования на языке Паскаль / Пер. с англ.; Под ред. Б. А. Ашкинази. — М.: Радио и связь, 1985.
20. Перминов О. Н. Язык программирования Паскаль: Справ. пособие. — М.: Радио и связь, 1989.
21. Перминов О. Н. Программирование на языке Паскаль. — М.: Радио и связь, 1988.
22. Пильщиков В. Н. Сборник упражнений по языку Паскаль: Учеб. пособие. — М.: Наука, 1989.
23. Прайс Д. Программирование на языке Паскаль/ Пер. с англ.; Под ред. О. Н. Перминова. — М.: Мир, 1987.
24. Пул Д. Работа на персональном компьютере / Пер. с англ.; Под ред. Е. К. Масловского. — М.: Мир, 1986.
25. Семашко Г. Л., Салтыков А. И. Программирование на языке Паскаль / Под ред. В. П. Ширикова. — М.: Наука, 1988.
26. Турбо-Паскаль. Версия 3.0 / Пер. с англ.: Справочное пособие. — М.: Наука, 1986.

27. Уилсон И., Эддиман А. Практическое введение в Паскаль / Пер. с англ.; Под ред. Л. Д. Райкова. — М.: Радио и связь, 1983.
28. Фигурнов В. Э. IBM PC для пользователя. — М.: Финансы и статистика, 1990.
29. Фигурнов В. Э. IBM PC для пользователя. — 2-е изд., перераб. и доп. — М.: Финансы и статистика, 1991.
30. Форсайт Р. Паскаль для всех / Пер. с англ.; Под ред. Ю. И. Топчеева. — М.: Машиностроение, 1986.
31. Хаузер Д., Хирт Дж., Хоукинс Б. Операционная система MS-DOS / Пер. с англ. с доп. А. Б. Пандре. — М.: Финансы и статистика, 1987.
32. Шаньгин В. Ф., Поддубная Л. М., Голубев-Новожилов Ю. С. Программирование на языке Паскаль. — М.: Высш. шк., 1988.
33. Эрбс Х.-Э., Штольц О. Введение в программирование на языке Паскаль / Пер. с нем.; Под ред. В. Н. Соболева. — М.: Мир, 1989.

ОГЛАВЛЕНИЕ

G.L.Semashko, A.I.Saltykoff

PROGRAMMING IN PASCAL LANGUAGE

Moscow, Nauka, Main Editorial Board for Literature on
Physics and Mathematics, 1992, 208 pp.

Readership: Students; researchers; enginers; all those concerned with computer science.

The book: Is given description the widespread programming language Pascal. Basically the standard Pascal is hear descripted, but also some futures of its usage for computer of several types: for instace, IBM-compatible mainframe and Personal computers. The style of explanation is simple and clear for nonspecialists. Numerous examples and detailed recommendations will help anybody to understand text easely, at once to make programs and to execute them in computer.
The first edition of this book was sold out in the several days and obtained the recognition of mase users. The present edition is supplemented with useful information about Turbo-Pascal system, version 5.0, including its list of procedures and functions.

Contents: The book consists of two chapters and seven appendixes. The first chapter is dedicated to the main Pascal conceptions: constants, variables, thear simple types, programm structure, statements, input/output of data, procedure and function definitions. The second chapter concerns the more complicated data types: records, sets, files, pointers.
In the appendixes short basic information about Tourbo-Pascal system, version 5.0 is represented. The list of the procedures and functions with comments is material for all users; short description of the NORTON COMMANDER and list of Editor commands is interesting for those who begin programming.

Autors: Galina Semashko, Albert Saltykoff, Joint Institute for Nuclear Research, Dubna, Russia.