МИНИСТЕРСТВО ВЫСШЕГО И СРЕДНЕГО СПЕЦИАЛЬНОГО ОБРАЗОВАНИЯ РСФСР

КУЙБЫШЕВСКИЙ ордена ТРУДОВОГО КРАСНОГО ЗНАМЕНИ АВИАЦИОННЫЙ ИНСТИТУТ имени АКАДЕМИКА С. П. КОРОЛЕВА

ВВЕДЕНИЕ В ЯЗЫК ПАСКАЛЬ

Утверждено
редакционно-издательским
советом института
в качестве
методических указаний
к практическим занятиям
для студентов

КУЙБЫШЕВ 1987

УДК 681.3.06

Методические указания содержат материал по основным средствам языка ПАСКАЛЬ. Методические указания предназначены для студентов I курса спец. 0646.

Составители: В. В. Пшеничников,
В. П. Сабило

Рецензенты: доц. В. М. Радомский,
нтьев

ДАННЫЕ

Идентификаторы

Каждая переменная должна иметь имя. Имена переменным даются с помощью идентификаторов. Идентификаторы состоят из букв и цифр, но первым символом идентификатора может быть только буква. Обычно система рассматривает только 8 первых символов идентификатора. В результате идентификаторы, различающиеся в тексте программы, для системы могут оказаться одинаковыми.

Примеры идентификаторов:

TIME, REGION, A576

Переменные и константы

Каждый элемент данных в программе является либо константой, либо переменной. В языке ПАСКАЛЬ имеются четыре стандартных типа:

целый *(INTEGER)*;
вещественный *(REAL)*;
булевский *(BOOLEAN)*;
символьный *(CHAR)*.

Константы не требуют специального описания. Тип констант определяется по форме их записи:

-100, 0, 2, 256 - целые;
-25.0, 5.3, 0.1, 0.25E-05 - вещественные;
TRUE, FALSE - логические (булевские);
'A', '=' - символьные.

Константы можно использовать непосредственно в выражениях, а можно в разделе констант дать им имена.

Пример раздела описания константы:

```
CONST
    PI=3.141593;
    NUMBER=15;
    B=2.5E-07;
    NAME='R';
```

Каждая переменная должна быть описана в разделе описания переменных.

Например:

```
VAR
  ST,SR: REAL;
  N,L,K: INTEGER;
  NAME: CHAR;
```

Целый тип. В ЭВМ может быть представлено конечное подмножество целых чисел, возможно различное для каждой конкретной ЭВМ. При попытке использовать целое значение, выходящее за пределы диапазона допустимых, возникает ошибка. Целые переменные обычно используют для организации счетчиков, в качестве индексов и т.д.

Для переменных и констант целого типа используются следующие операции:

+ (сложение),
- (вычитание),
* (умножение);
DIV (целочисленное деление);
MOD (вычисление остатка от целочисленного деления);
/ (деление).

Естественно, что результаты всех приведенных операций (кроме /) для целых чисел дают целый результат. Результат деления (/) дает всегда вещественный результат.

Вещественный тип. Вещественные переменные обладают двумя характеристиками - диапазонами и точностью, которые зависят от реализации языка на конкретной ЭВМ. Например, о ЭВМ могут говорить, что она обрабатывает числа в диапазоне $10^{\pm 75}$ с точностью до 7 десятичных цифр.

Знаки операций для вещественных операндов следующие:

+ (сложение);
- (вычитание);

* (умножение);

/ (деление).

Булевский тип. Булевские переменные могут иметь только одно из двух значений:

TRUE (истина),

FALSE (ложь).

Над булевскими переменными, константами или выражением, дающим булевские значения, можно выполнить следующие операции:

AND (операция И);

OR (операция ИЛИ);

NOT (операция НЕ).

Символьный тип. Значением символьной переменной является символ. В языке ПАСКАЛЬ не определено конкретное множество символов, поскольку используемые символы допустимы в конкретной ЭВМ. Обычно это цифры, прописные буквы, знаки операций, скобки и другие символы. Внутреннее представление прописных букв упорядочено по возрастанию в порядке алфавита. Во внутреннем представлении цифры упорядочены по возрастанию и следуют друг за другом. В ЭВМ серии СМ используется стандартный код *ASCII(American Standart Code for Information Interchange)*, для ЕС ЭВМ - код *EBCDIC(Extented Binary Coded Dectmal Information Code)*.

ВЫРАЖЕНИЯ И ОПЕРАТОР ПРИСВАИВАНИЯ

Оператор присваивания

Оператор присваивания имеет вид

⟨переменная⟩ : = ⟨выражение⟩ ,

где символ ": =" означает присвоить, а "выражение" - это арифметическое, логическое или символьное выражение. Выражение вычисляется и полученное значение присваивается переменной.

Арифметические выражения

В арифметическом выражении можно использовать целые и вещественные переменные и константы, знаки арифметических операций, круглые скобки и функции.

Последовательность выполнения операций в выражении соответствует обычным математическим правилам. Отметим некоторые особенности арифметических выражений:

для целочисленных переменных и констант операции "+", "-", "*", *div*, *mod* дают целый результат;

операция "/" дает вещественный результат (даже для целых операндов)";

значение целочисленного выражения может быть присвоено переменной целого и вещественного типов;

запрещается присваивать результат вещественного выражения целой переменной.

Для преобразования вещественного значения в целое необходимо использовать следующие встроенные функции:

TRUNC - преобразование к целому значению отбрасыванием дробной части;

ROUND - преобразование к целому значению округлением, Функция *ROUND* для положительных значений дает результат равный *TRUNC(X+0,5)*, и для отрицательных значений *TRUNC(X-0,5)*.

Пример. (X, Y, Z - вещественные, I, J, K - целые)

Математическая запись	Запись на языке ПАСКАЛЬ
$Z = \frac{X-Y}{X+Y}$	Z:=SIN(X)+COS(Y)
$Z = \sin X + \cos Y$	Z:=SQR(X)+SQR(Y)
$Z = X^2 + Y^2$	Z:=SQR(SIN(X))
$Z = \sin^2 X$	Z:=EXP(-X)
$Z = e^{-X}$	Z:=I+K
$Z = I + K$	I:=TRUNC(X-Y) или
$I = X - Y$	I:=ROUND(X-Y).

Логические выражения

Для выражений любого упорядоченного типа операции отношения или сравнения дают результат логического типа (*TRUE* или *FALSE*). В большинстве реализации языка это следующие операции:

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

Над логическими выражениями можно выполнять логические операции *AND* (И), *OR* (ИЛИ) и *NOT* (НЕ). Логические выражения можно использовать в операторах присваивания, операторах цикла и условных операторах. Обращаем внимание на запись таких условий, как $min \leqslant X \leqslant max$. На языке ПАСКАЛЬ это выражение запишется следующим образом:

```
(X>=MIN)AND(X<=MAX)
```

Пример.

```
VAR A,B:REAL;
    D,C:BOOLEAN;
C:=A>B;
C:=(A>5)AND(B<7);
C:=TRUE;
D:=NOT C;
```

Символьные выражения

Простые данные символьного типа имеют длину один символ. Для задания строки символов используются массивы. Для символьных данных предусмотрены встроенные функции *ORD* и *CHR*.

Функция *ORD* выдает порядковый номер символа (зависящий от используемого кода). Так например, *ORD('1')* в коде *ASCII* равен 48, *ORD(2')* - 49 и т.д. Буквы упорядочены по возрастанию кодов, но коды могут идти не подряд, т.е. справедливы утверждения,

что *ORD('A') < ORD('B')* или что *'A' < 'B'*. Функция *CHR* обратная по отношению к функции *ORD*. Она выдает символ, номер (код) которого задан ей в качестве аргумента. Над переменными символьного типа можно производить только операции сравнения (*<*, *<=*, *=*, *<>*, *>*, *>=*).

Пример.
```
VAR  S: CHAR;
     :
S = 'A';
S = CHR(номер символа).
```

Стандартные функции Паскаля приведены в таблице.

Наименование	Аргумент	Значение	Содержание
ABS	Целый	Целое	$\|X\|$
ABS	Вещественный	Вещественное	$\|X\|$
ARCTAN	Целый	Вещественное	$arctan X$
ARCTAN	Вещественный	Вещественное	$arctan x$
EXP	Целый	Вещественное	e^x
EXP	Вещественный	Вещественное	e^x
CHR	Целый	Символьное	Выбор символа по его порядковому номеру
EOF	Имя файла	Булевское	*TRUE* при попытке чтения после конца файла
EOLN	Имя файла	Булевское	Значение *TRUE*, если из ввода файла поступил конец строки
SQR	Целый	Целое	x^2
SQR	Вещественный	Вещественное	x^2
SUCC	Скалярный	Скалярное	Последующий элемент
TRUNC	Вещественный	Целое	Преобразование к целому отбрасыванием дробной части
SQRT	Целый	Вещественное	$\sqrt{x}$

Окончание табл.

Наименование	Аргумент	Значение	Содержание
SQRT	Вещественный	Вещественное	$\sqrt{x}$
ROUND	Вещественный	Целое	Преобразование к целому округлением
ORD	Скалярный	Целое	Порядковый номер элемента
PRED	Скалярный	Скалярное	Предшествующий элемент
SIN	Целый	Вещественное	$\sin x$
COS	Целый	Вещественное	$\cos x$
SIN	Вещественный	Вещественное	$\sin x$
COS	Вещественный	Вещественное	$\cos x$
LN	Целый	Вещественное	$\ln$
LN	Вещественный	Вещественное	$\ln$
ODD	Целый	Булевское	*TRUE* при нечетном аргументе, *FALSE* при четном аргументе

СТРУКТУРА ПРОГРАММЫ

Программа состоит из заголовка и блока и заканчивается точкой. Заголовок может иметь вид

PROGRAM имя программы *(INPUT, OUTPUT);*

INPUT, OUTPUT - имена файлов ввода и вывода. Затем следует блок, состоящий из определения констант, описания переменных и составного оператора. Составной оператор имеет вид

```
BEGIN
   оператор I;
   оператор 2;
      :
```

оператор *N*

END

Операторы 1,2,... *N* могут быть любыми, в том числе и составными. Операторы отделяются друг от друга символом ';'.

Пробелы не должны встречаться внутри зарезервированных слоев, идентификаторов или составных символов. В остальных случаях пробелы используются произвольно для улучшения читабельности программы. Комментарий имеет вид

(* последовательность символов *)

Комментарии помогают читателю понять программу и могут вставляться в программу всюду, где разрешены пробелы.

УСЛОВНЫЙ ОПЕРАТОР

Общий вид условного оператора

IF < условие >
THEN < оператор 1 >
ELSE < оператор 2 >

Условие - булевское выражение, если его значение ИСТИНА (*TRUE*), то выполняется < оператор 1 >, если ЛОЖЬ (*FALSE*), то < оператор 2 >. Операторы 1 и 2 могут быть любыми, в том числе условными и составными.

Составной оператор используется в условном в том случае, когда в *THEN* или *ELSE* ветвях должно стоять несколько операторов. Сравните два условных оператора:

```
1. IF  A>B
     THEN
       BEGIN
         MAX:=A;
         MIN:=B;
       END

2. IF  A>B
     THEN  MAX:=A;
           MIN:=B;
```

В первом операторе присваивание переменным *MAX* и *MIN* выполняется только при *A>B* , во втором - оператор присваивания

MIN:=B выполняется всегда, так как он не входит в состав условного оператора. Очень важен для ясности программы вид записи условных операторов. Основной принцип – альтернативные части записывать со сдвигом по отношению к условию.

Пример записи:

```
1. IF < условие >
       THEN < оператор 1 >
       ELSE < оператор 2 >
2. IF < условие >
     THEN
       BEGIN
         < операторы >
       END
     ELSE
       BEGIN
         < операторы >
       END
```

Если после *THEN* стоит условный оператор, например,

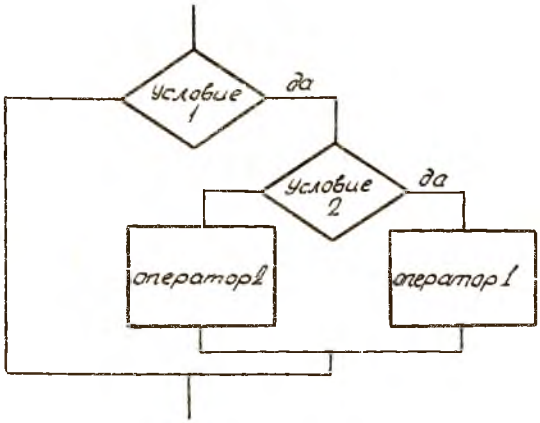
```
IF< условие 1 >
    THEN
  IF< условие 2 >
THEN<оператор 1>
ELSE<оператор 2 >,
```

то неоднозначность (к какому *IF* относится *ELSE*) разрешается с помощью правила:

ELSE относится к ближайшему условному оператору *IF* , не имеющему альтернативы *ELSE* . По этому правилу приведенный выше оператор реализует схему, приведенную на рисунке.

Приведем пример программы с использованием условного оператора. В этой программе используется оператор печати *WRITELN* и оператор ввода данных *READ*.

```
PROGRAM PRIM1(INPUT, OUTPUT).
(* ПРОГРАММА РЕШЕНИЯ КВАДРАТНОГО УРАВНЕНИЯ *)
  VAR
   A,B,C,D, RE,IM: REAL;
  BEGIN
    READ(A,B,C);          (* вводим значения коэффициентов *)
```




```
IF(A=0) AND(B=0)
 THEN WRITELN                ('уравнение вырождено')
  ELSE  IF  A=0
 THEN WRITELN                ('единств.корень равен', -C/B)
 ELSE IF  C=0
 THEN  WRITELN               ('корни', -B/A, 'и', 0)
  ELSE
  BEGIN
   RE:=-B/(2*A);
   D:=SQR(B)-4*A*C;
   IM:=SQRT(ABS(D))/(2*A);
   IF D>=0                   ('корни', RE+IM)
   THEN WRITELN              ('компл. корни равны',
   ELSE  WRITELN
   RE,'+I*',IM,'и',RE,'-I*',IM)
   END
END.      (* КОНЕЦ ПРОГРАММЫ PRIM1  *)
```

ОПЕРАТОР ЦИКЛА С ПОСТУСЛОВИЕМ (*REPEAT*)

Оператор цикла с ПОСТУСЛОВИЕМ имеет две части:
собственно цикл;
условие окончания.
Общий вид оператора:

REPEAT

операторы

UNTIL условие,

Цикл выполняется до тех пор, пока условие не будет удовлетворено. Условие проверяется в конце цикла повторения. Такой оператор удобно использовать, если заранее неизвестно, сколько именно повторений требуется. Например, при вычислении суммы ряда с заданной точностью или до определенного предела. Следует учитывать, что один раз цикл выполняется всегда.

Пример. Определить, сколько членов ряда потребуется, чтобы выполнилось неравенство

$$1+\frac{1}{2}+\frac{1}{3}+\cdots+\frac{1}{n}>L,$$

где L - заданный предел;

```
PROGRAM SUM(INPUT,OUTPUT);
  VAR
      N:INTEGER; (* число членов *)
    SUMMA,LIMIT:REAL;( * сумма, предел * )
    BEGIN
      N:=0;
      SUMMA:=0;
      READ(LIMIT);
    REPEAT
      N:=N+1;
      SUMMA:=SUMMA+1/N
    UNTIL SUMMA>LIMIT;
    WRITE(N)
END.                          ( * конец программы SUM * )
```

ОПЕРАТОР ЦИКЛА С ПРЕДУСЛОВИЕМ (*WHILE*)

В отличие от оператора цикла с постусловием здесь условие проверяется в начале цикла повторения.

Вид оператора:

WHILE < условие > *DO*
< оператор >

Цикл выполняется все время, пока условие истинно. Поэтому перед первым вхождением в цикл переменные входящие в <условие> должны быть обязательно определены и условие должно иметь определенное значение. Если условие будет иметь значение "ложь", то цикл выполняться не будет. При практическом программировании оператор цикла с предусловием оказывается более удобным, чем с постусловием, так как во многих случаях необходимо установить, не следует ли пропустить цикл целиком.

<Оператор> может быть и одиночным и составным оператором. Следует отметить, что проверка условия производится только после полного выполнения всех операторов составного оператора даже если один из них сделал условие ложным, так как проверка осуществляется только в начале каждого нового цикла.

ОПЕРАТОР ЦИКЛА С ПАРАМЕТРОМ (*FOR*)

Оператор цикла *FOR* обеспечивает выполнение заданное число раз некоторого оператора (может быть составного).

Оператор цикла имеет вид:

FOR < параметр > : = <выражение I>
{ *TO* / *DOWN TO* } < выражение 2 > *DO* < оператор > .

Здесь параметр является обычной переменной, которая должна быть описана в разделе переменных, а выражения должны быть простого типа, но не вещественные. Выражения вычисляются в начале выполнения оператора *FOR*.

Приведем алгоритм выполнения оператора *FOR* с положительным шагом для параметра целого типа:

```
        FOR < параметр >  : = < выражение I >
        TO  < выражение 2 > DO < оператор >
    параметр : = выражение I;
    переменная : = выражение 2;
    метка A : IF  (параметр > переменная)
              THEN GOTO метка B;
оператор;
параметр : = параметр + I;
   GOTO   метка A;
метка B: ...;
```

Соответственно для отрицательного шага (*DOWNTO* вместо *TO*) будет следующий алгоритм:

```
        параметр : = выражение I;
        переменная : = выражение 2;
        метка A : IF  (параметр  < переменная)
                      THEN GOTO метка B;
        оператор;
        параметр : = параметр -I;
        GOTO  метка A;
              метка B: ...;
```

Так как параметр цикла проверяется перед каждым выполнением цикла, возможно, что оператор, содержащийся в операторе цикла, ни разу не выполнится.

Следующая программа вычисляет сумму n первых членов ряда $1+\frac{1}{2}+\frac{1}{3}+\ldots+\frac{1}{n}$:

```
PROGRAM SUMMA(INPUT, OUTPUT);
  VAR
    I,N: INTEGER;
      S: REAL;
  BEGIN
    READ(N); S:=0;
    FOR I:=1 TO N S:=S+1/N;
    WRITE(S)
END.
```

Для данных символьного типа параметр цикла принимает значение следующего символа (для *TO*) или предыдущего (для *DOWNTO*). Например, следующая программа будет печатать символы от *B* до *R*:

```
VAR S CHAR;
```

```
BEGIN
FOR S:='B' TO 'R' DO WRITE(S);
END.
```

ОПЕРАТОР ПЕРЕХОДА *GOTO*

Любой оператор языка ПАСКАЛЬ может быть помечен с помощью метки. Метка - это целое число без знака, состоящее не более чем из четырех цифр. Каждая метка должна быть единственной в блоке и описана в разделе описаний меток.

Например,

LABEL 25, 476, 1020;

Оператор перехода имеет вид

GOTO < метка >

После его выполнения всегда следующим выполняется оператор, помеченной указанной в *GOTO* меткой. Широко использовать оператор *GOTO* не рекомендуется, так как он обычно усложняет программу. Дело в том, что передача управления "вперед", и особенно "назад", затрудняет чтение программы, ее отладку. Язык ПАСКАЛЬ имеет структурные операторы, которые позволяют вообще обойтись без *GOTO*.

Наиболее целесообразно использовать *GOTO* для выхода из операторов (например цикла) при возникновении особой ситуации в программе.

ПРОЦЕДУРЫ ВВОДА-ВЫВОДА

Язык ПАСКАЛЬ не имеет собственно операторов ввода-вывода. В их качестве используются четыре встроенные процедуры:

1. *READ* (переменная 1, переменная 2, ... переменная *n*) - обеспечивает чтение данных из стандартного файла с именем *INPUT* и присвоением прочитанных значений соответствующим переменным списка. Типы вводимых значений должны совпадать с типами соответствующих переменных.

2. *READLN* (переменная 1, переменная 2, ... переменная *N*) выполняет чтение, как и *READ*, но по окончании чтения осуществляется переход к началу новой строки файла *INPUT*.

3. *WRITE* (выражение I, выражение 2, ..., выражение *N*) выполняет вывод (в частности печать) в стандартный файл *OUTPUT* значений выражений, указанных в списке.

4. *WRITELN* (выражение I, выражение 2, ..., выражение *N*) выполняет вывод как и *WRITE*, но после вывода последнего выражения производится переход к новой строке файла *OUTPUT* (к новой строке печати).

Исходные данные целого и вещественного типа во входном файле следует отделять друг от друга произвольным числом пробелов. Число позиций, выделяемое для печати данных, определяется реализацией языка.

Пример I. Ввести значения трех переменных и вывести их на печать в одной строке:

```
READ(A,B,C);
WRITE(A,B,C);
```

Пример 2. Ввести массив из 5 чисел и вывести его на печать

```
VAR   M: ARRAY [1..5] OF REAL;
      I: INTEGER;
        ( * цикл ввода *)
FOR I:=1 TO 5 DO READ(M[I]);
        ( * цикл печати *)
FOR I:=1 TO 5 DO WRITE(M[I]);
```

Пример 3. Вывести на печать текст 'ТАБЛИЦА'

```
WRITE  ('ТАБЛИЦА')
```

Для ввода и вывода символьных упакованных массивов можно обойтись и без циклов, например:

```
VAR STR : PACKED ARRAY[1..10] OF CHAR;
    .
    .
READ(STR);( * ввод IO-ти символов *)
WRITE(STR);( * печать IO-ти символов *)
```

Язык имеет средства редактирования данных, с помощью которых можно указать длину поля, выделяемую для печати данного, и форму печати, причем значение в этом поле прижимается к правому краю. Так, для переменных и выражений целого типа указывается длина поля в форме *WRITE(K:M)*,

где *K* - выражение целого типа, значение которого будет выдано на печать,

M – выражение целого типа, определяющее длину поля.

Для вещественных значений можно указать и длину поля, и число цифр в дробной части в форме

$$WRITE(A:M:N),$$

где M – длина поля,

N – число цифр в дробной части (M и N выражения целого типа).

Если для вещественного числа N не задано, то оно печатается в форме с плавающей точкой. Для данных типа *CHAR* и *PACKED ARRAY OF CHAR* указывается длина поля для печати.

Пример 1. Для целых $K = 12$ и $L = 5$ по оператору

$$WRITE(K:3, L:4)$$

будет напечатана строка $\underbrace{\sqcup 1\,2}_{K}\underbrace{\sqcup\sqcup\sqcup 5}_{L}$

Пример 2. Для вещественных переменных $A = 25.6$ и $B = 347.2$ оператором

$$WRITE(A:6:1, B:13)$$

будет выдана строка

$\sqcup\sqcup 2\,5.6\sqcup 3.4\,7\,\emptyset\,\emptyset\,\emptyset\,\emptyset+E\,\emptyset\,2$

Пример 3. Оператор $WRITE('X', '\sqcup':5, 'Y')$ печатает строку $X\sqcup\sqcup\sqcup\sqcup\sqcup Y$

В некоторых реализациях языка первый символ строки файла печати является управляющим (в выше приведенных примерах это не учитывается):

'+'	означает подавить пропуск перед печатью;
'$\sqcup$'	перевод на одну строку перед печатью;
'$\emptyset$'	перевод на две строки перед печатью;
'–'	перевод на три строки перед печатью;
'1'	перевод на новую страницу.

Стандартно для устройств печати назначены: длина строки – 120 символов и ширина страницы – 60 строк.

МАССИВЫ

Одномерные массивы

Массив – это упорядоченный список элементов данных одного типа. Для обращения к элементу одномерного массива необходимо указать имя массива и значение одного индекса. Элементы массива расположены в следующих друг за другом ячейках памяти. Одномерный массив иногда называют вектором. В описании массивов задаются начальное и конечное значения индекса. Например,

```
VAR A: ARRAY [1..10] OF REAL;
```

Описан массив с именем A, состоящий из элементов A_1, A_2, ..., A_{10} вещественного типа

```
VAR B: ARRAY [1981..1985] OF INTEGER;
```

Массив B имеет пять элементов целого типа с индексами 1981, 1982, 1983, 1984, 1985.

Одномерными массивами представляются в языке символьные строки. Так, строка из 25 символов будет описана так:

```
VAR LINE: ARRAY [1..25] OF CHAR;
```

Для доступа к элементам массива необходимо запись имени массива (индексное выражение), где индексное выражение должно давать результат целого типа. Напишем полную программу для вычисления, например, суммы элементов одномерного массива:

```
PROGRAM SUMMA(INPUT, OUTPUT);
  CONST N=100;
  VAR A: ARRAY [1..N] OF REAL;
    I: INTEGER;
    S: REAL;
BEGIN   (* цикл ввода и место исходных данных *)
   FOR I:=1 TO N DO
       BEGIN READ(A[I]); WRITE(A[I]); END;
  S:=0; (* цикл накопления суммы *)
     FOR I:=1 TO N DO
        S:=S+A[I];
     WRITELN;
  WRITELN('␣ сумма элементов равна ',S );
END.
```

Двумерные массивы

Двумерные массивы (матрицы) описываются с указанием максимальных и минимальных значений индексов по двум измерениям.

Например, `VAR A:ARRAY[1..5,1..5]OF REAL;`

Пример описания двумерного массива с использованием раздела определения типов приведен в п. "Описание массивов в разделе определения типов".

Пример программы работы с двумерными массивами (умножение квадратных матриц)

```
PROGRAM MULT (INPUT,OUTPUT);
VAR A,B,C : ARRAY[1..5] OF REAL;
    I,J,K : INTEGER;
BEGIN (* цикл ввода матрицы A *)
   FOR I:=1 TO 5 DO
      FOR J:=1 TO 5 DO READ(A[I,J]);
       (* цикл ввода матрицы B *)
      FOR I:=1 TO 5 DO
      FOR J:=1 TO 5 DO READ(B[I,J]);
       (* умножение матрицы A и B *)
      FOR I:=1 TO 5 DO
         FOR J:=1 TO 5 DO
            BEGIN
              C[I,J]:=0;
              FOR K:=1 TO 5 DO
                C[I,J]:=C[I,J]+A[I,K]*B[K,J]
            END;
         (* печать матрицы-результата C *)
      FOR I:=1 TO 5 DO
         BEGIN
           WRITELN;
           FOR J:=1 TO 5 DO
           WRITE(A[I,J]:12:4)
           END
      END;
```

ТИПЫ ДАННЫХ, ОПРЕДЕЛЯЕМЫЕ ПОЛЬЗОВАТЕЛЕМ

Типы данных вводятся (описываются) в разделе определения типов.

Перечисляемые типы данных

Для наглядности будем использовать в этом разделе буквы русского алфавита. В перечисляемом типе указывается список констант, например:

```
TYPE    ЦВЕТ =(КРАСНЫЙ, ОРАНЖЕВЫЙ, ЖЕЛТЫЙ, ЗЕЛЕНЫЙ, ГОЛУБОЙ,
        СИНИЙ, ФИОЛЕТОВЫЙ);
TYPE    ДЕНЬ НЕДЕЛИ = (ПОНЕДЕЛЬНИК, ВТОРНИК, СРЕДА, ЧЕТВЕРГ,
        ПЯТНИЦА, СУББОТА, ВОСКРЕСЕНЬЕ);
```

Здесь ЦВЕТ и ДЕНЬ НЕДЕЛИ - это новые типы данных.
После того, как определен новый тип данных, могут быть объявлены переменные этого типа:

```
VAR     ДЕНЬ 1, ДЕНЬ 2: ДЕНЬ НЕДЕЛИ;
        СВЕТОФИЛЬТР: ЦВЕТ;
```

Переменные могут получать значения из соответствующего списка с помощью оператора присваивания.

Например,

```
ДЕНЬ: = ВТОРНИК;
СВЕТОФИЛЬТР: = ГОЛУБОЙ;
```

Константы в списке перечисляемого типа считываются упорядоченными, т.е. им ставится в соответствие последовательность целых чисел, начинающаяся с нуля. Значением стандартной функции *ORD* является порядковый номер значения переменной или константы в списке, например,

ORD (ГОЛУБОЙ) равен 5,
ORD (ВТОРНИК) равен 2.

Значениями стандартных функций *PRED* и *SUCC* являются предыдущий и последующий по отношению к аргументу элемент списка, например

PRED (СИНИЙ) есть ГОЛУБОЙ,
SUCC (ПЯТНИЦА) есть СУББОТА

Над данными перечисляемого типа могут выполняться только операции сравнения, при этом результат получаем сравнением порядковых номеров, например

```
                    ГОЛУБОЙ < СИНИЙ,
                    СРЕДА  > ВТОРНИК
```

Еще пример:

```
                    ДЕНЬ I : = СРЕДА;
                    ДЕНЬ 2 : =ПЯТНИЦА;
          IF        ДЕНЬ I = ДЕНЬ 2  THEN
                    WRITE ('ВРЕМЯ')
    ELSE  IF        ДЕНЬ I < ДЕНЬ 2
    THEN  WRITE       ('ЕЩЕ РАНО')
    ELSE  WRITE       ('УЖЕ ПОЗДНО');
```

Могут рассматриваться как перечисляемые целочисленные, символьные и булевы типы данных. Для них допустимы такие операции:
PRED, SUCC и ORD.

Ограниченный тип данных

К ограниченному типу данных относятся элементы подмножества подряд идущих величин некоторого базового типа, который может быть перечислимым или простым типом, кроме вещественного. Ограниченный тип устанавливается в разделе определения типов посредством указания первого и последнего элементов подмножества.

Например:

```
        TYPE    ЦИФРЫ = 0...9;
                БУКВЫ = A ... ;
                ЦВЕТА = красный ... голубой;
```

Здесь типы ЦИФРЫ имеют базовый тип *INTEGER* , тип БУКВЫ - тип *CHAR* , а тип ЦВЕТА - тип ЦВЕТ, который мы описали в п. "Перечисляемые типы данных".

Переменные ограниченного типа определяются в разделе переменных:

```
  VAR  A,B  : ЦИФРЫ
       C,D  : БУКВЫ
       S    : ЦВЕТА
       INDEX: 1..50;
```

Для переменных ограниченного типа при выполнении программы осуществляется контроль значений. Выход значений за границы указанного диапазона приводит к ошибке. Это обстоятельство следует учитывать

при работе с массивами и использовать в качестве индексов переменные ограниченного типа вместо типа *INTEGER*. Более того, для большинства задач, использующих целые переменные, целесообразно заменять их на переменные ограниченного типа.

Описание массивов в разделе определения типов

Программа будет более наглядной, если ввести новые типы в разделе определения типов.

Например:

```
TYPE  INDEX = 1..20;
      VECTOR = ARRAY [INDEX] OF REAL;
      MATR = ARRAY [INDEX] OF VECTOR;
VAR   A,B : VECTOR;
      I,J : INDEX;
      C,D : MATR;
```

Здесь *VECTOR* - новый тип, одномерный массив с индексами элементов от 1 до 20,

MATR - новый тип - матрица 20х20.

ОПЕРАТОР ВЫБОРА *CASE*

Оператор выбора *CASE* можно рассматривать как более общий случай условного оператора. В то время как условный оператор обеспечивает выполнение одного из двух операторов (группы операторов), оператор *CASE* производит выбор из нескольких операторов (группы операторов). Селектором в операторе *CASE* является выражение, значение которого принадлежит целочисленному, литерному и логическому типу, либо типу, определяемому программистом, - перечислимому или ограниченному.

Оператор выбора имеет *CASE* - список, в который каждый оператор помечен константой. Для выполнения будет выбран тот оператор, перед которым стоит константа, равная значению селектора. Разрешается помечать операторы более чем одной константой.

Например:

```
VAR  S: INTEGER;
     . . .
CASE S OF
```

```
    1 : WRITE    ('значение S = 1');
    2 : WRITE    ('значение S = 2');
3, 4 : WRITE     ('значение S = 2 или 3')
END;
```

Если селектор (здесь *S*) принимает значение, не указанное в *CASE*-списке, то действия оператора *CASE* не определены, но многие реализации языка ПАСКАЛЬ имеют в операторе *CASE ELSE* - ветвь,которая выполняется в этом случае.

Например :

```
TYPE SW = 'A'.. 'D';
VAR INCHAR : SW;
        . . .
  CASE INCHAR OF
   A,B : WRITE   ('значение A или B' );
   C,D : WRITE   ('значение C или D' )
   ELSE WRITE    ('недопустимое значение')
   END .
```

ПРОЦЕДУРЫ И ФУНКЦИИ

Процедуры и блочная структура программы

Проектирование программы "сверху вниз" разбивает программу на языке ПАСКАЛЬ на множество процедур. Процедура - это некоторая поименованная последовательность операторов. К такой последовательности можно обращаться по ее имени из различных точек программы с возвратом в точки вызова. Например, пусть имеется процедура *EXCHANGE*, которая меняет местами значения двух переменных. Тогда в тех точках программы, где необходимо произвести такую перестановку, необходимо вызвать процедуру, записав ее имя. Такая процедура имеет следующий вид:

```
PROCEDURE EXCHANGE;
BEGIN
      R: = X;
      X: = Y;
      Y: = R
END ;    ( * имеются местами значения X и Y *)
```

Очевидно, что использовать такую процедуру можно только для переменных с именами *X* и *Y* . Чтобы сделать ее универсальной, введем в заголовок список формальных параметров:

```
PROCEDURE EXCHANGE(VAR X,Y: REAL);
  VAR  R: REAL;
  BEGIN
      R:=X;
      X:=Y;
      Y:=R
  END;
```

Здесь *X* и *Y* - формальные параметры, которые описаны как вещественные переменные.

Вызов такой процедуры имеет вид

```
EXCHANGE(A,B),
```

где *A* и *B* - фактические параметры.

Результат вызова процедуры соответствует результату выполнения тела процедуры, в которой формальные параметры заменены на фактические. Для нашего примера выполнятся следующие условия:

```
BEGIN  R:=A; A:=B; B:=R  END,
```

т.е. меняются местами значения переменных *A* и *B* .

Приведем полный текст программы с использованием процедуры *EXCHANGE*:

```
PROGRAM PRIM(INPUT, OUTPUT);
VAR   M: ARRAY[1..10] OF REAL;
      A,B,C,D: REAL; I: INTEGER;
        ( * ниже идет текст процедуры *)
PROCEDURE EXCHANGE(VAR X,Y: REAL);
  VAR R: REAL;
  BEGIN R:=X; X:=Y; Y:=R  END;
    BEGIN       ( * тело основной программы *)
    READ(A,B,C,D);
    FOR I:=1 TO 10 DO READ(M[I]);
  EXCHANGE(A,B);   (* перестановка A и B *)
  EXCHANGE(C,D);   (* перестановка C и D *)
EXCHANGE(A[5],A[9]);  (* перестановка пятого и девятого
END.                    элементов массива *)
```

Формальные и фактические параметры должны соответствовать друг другу по количеству, типу и последовательности записи в списках.

В общем случае программа на языке ПАСКАЛЬ может содержать несколько процедур, которые, в свою очередь, могут также содержать процедуры внутри себя. В этой связи следует рассмотреть области действия переменных, описанных в различных процедурах. Следует руководствоваться следующим правилом:

переменная, описанная во внешней процедуре (блоке), действует во всех внутренних процедурах, кроме случая, когда одноименная переменная описана во внутренней процедуре.

Рассмотрим следующую структуру программы:

```
PROGRAM PRIM(INPUT, OUTPUT);
  VAR A,B,C : REAL;
  PROCEDURE S;   ( * начало процедуры S  *)
  VAR F,B : REAL;
  BEGIN
    :
  END;           ( * конец процедуры S  *)
  PROCEDURE T;       ( * начало процедуры T  *)
  VAR  R,C  INTEGER;
  BEGIN
    :
  END;                    ( * конец процедуры T  *)
  BEGIN                 ( * блок основной программы *)
    :
  END.
```

Здесь A, B и C описаны во внешней процедуре (глобальные переменные), R и F внутри процедур S и T (локальные переменные). Областью действия переменной F является только процедура S, а R только процедура T. Переменная A действует в процедуре $PRIM$ и во внутренних процедурах S и T. Переменная B, описанная в $PRIM$, действует и в $PRIM$, и в T (у процедуры S есть своя локальная переменная B). Переменная C, описанная в $PRIM$, действует в $PRIM$ и в S (в T своя локальная C). Согласно блочной структуре распределяется память: при входе в процедуру выделяется память для локальных переменных, при выходе освобождается. Таким образом, значение переменной B,

вычисленное в *S* будет потеряно при выходе из нее, т.е. недоступно для основной программы и для процедуры *T* . Аналогично для переменных *F* , *R* и *C* .

Способы передачи параметров

В языке ПАСКАЛЬ имеется два способа передачи:
вызов по значению (подстановка значения);
вызов по ссылке (подстановка переменных).

В процедуре *EXCHANGE* использована подстановка переменных. На это указывает слово *VAR* , стоящее перед формальными параметрами в заголовке процедуры. Для подстановки значения слово *VAR* просто опускается. При подстановке переменной процедуре передаются адреса фактических параметров, т.е. для формальных параметров память не выделяется, и процедура работает с областями памяти, выделенной для фактических параметров. Таким образом, выполнение процедуры может изменить "чужие" переменные. При передаче значения выделяется память для формальных параметров, и в нее помещаются значения соответствующих фактических параметров. Выполнение процедуры не может изменить фактических параметров.

Таким образом, если параметр является аргументом процедуры, то логично использовать подстановку значения, если параметр - результат процедуры, нужно использовать подстановку переменной. Но следует помнить, что при подстановке значений создаются "лишние" локальные переменные, что особенно невыгодно для массивов, так как требуется двойной объем памяти.

Передача массивов

Если в списке формальных параметров процедуры или функции есть имя массива, то оно должно быть описано следующим образом:

VAR имя массива: тип; (при подстановке переменной)
или имя массива: тип; (при подстановке значений).

Рассмотрим программу с процедурой поиска минимального элемента одномерного массива

```
PROGRAM G(INPUT,OUTPUT);
TYPE VECTOR=ARRAY[1..10] OF REAL;
VAR  A:VECTOR; I:INTEGER;
```

```
            AMIN : REAL;
PROCEDURE   MIN ( VAR  B: VECTOR; N: INTEGER;
                  VAR  BMIN : REAL);
  VAR  I: INTEGER;
  BEGIN
      BMIN: =A[1];
    FOR I: = 2  TO  N  DO
     IF  BMIN > B[I]  THEN  BMIN: = B [I]
   END;
BEGIN  FOR  I: =1  TO  1Ø  DO
                  READ (A[I]);

          G(A,1Ø,AMIN);  (* обращение к процедуре *)
          WRITE (AMIN);
END.
```

Эта процедура может искать минимальный элемент только в одномерном массиве из 10 элементов, т.е. *ARRAY[1..1Ø]OF REAL*. И даже то, что мы ввели в список формальных параметров переменную *N*, цикл поиска сделали до *N*, не делает ее более универсальной.

Функции

В отличие от процедуры в заголовке функции указывается ключевое слово *FUNCTION*, а в списке формальных параметров указываются только аргументы, а результат функции, тип которого указывается в заголовке, присваивается имени функции. Вызов функции представляет собой выражение (как и для стандартных функций, например, *SIN, COS* и т.п.), в отличие от вызова процедуры, являющегося оператором. Механизм передачи параметров, правила локализации переменных и функций те же, что и у процедур.

Запишем функцию для вычисления выражения $e^{-x}\sin x$:

```
FUNCTION   F(X : REAL): REAL;
  BEGIN
    F = EXP(-X) * SIN(X)
END;
```

Полная программа с использованием функции F будет иметь вид:

```
PROGRAM GLAV(INPUT, OUTPUT);
  VAR  ARG , FUN : REAL;
  FUNCTION F(X : REAL): REAL;
```

```
BEGIN
  F: = EXP(-X)*SIN(X)
END ;
BEGIN

        WRITELN          ('вводи значение');
        READ(ARG);       (*ввод аргумента *)
      FUN: = F(ARG);     (* обращение к функции *);
       WRITELN(FUN)      (* печать результата *)
  END.
```

Рекурсии

Постарайтесь самостоятельно разобраться в процедуре вычисления интеграла.

```
FUNCTION FACT(N: INTEGER): INTEGER;
BEGIN
 IF N=1
   THEN
     FACT: =1
  ELSE
     FACT: = FACT(N-1)*N
END.
```

Обратите внимание, что в тексте процедуры *FACT* есть обращение к ней самой. Такие процедуры называются рекурсивными, а обращение – рекурсивным. Для вычисления в вашей программе

$$Z = K!$$

необходимо записать $Z: = FACT(K)$.

При обращении к функции *FACT* с параметром K его значение передается переменной N, локальной в теле функции. При $K > 1$ выполняется рекурсивное обращение к *FACT* со значением параметра $K-1$. Возникает копия тела функции. Для нового параметра $K-1$ резервируется новая переменная, локальная в копии тела функции, и ей присваивается значение $K-1$. Затем вновь происходит обращение к функции со значением $K-2$ и так до тех пор, пока значение параметра не станет равным единице. Тогда начнется последовательное вычисление копий функций в обратном, чем был описан, порядке для получения значения $K!$.

Составители: Виктор Владимирович П ш е н и ч н и к о в,
Валентин Павлович С а б и л о

ВВЕДЕНИЕ В ЯЗЫК ПАСКАЛЬ

Редактор Е.Д.А н т о н о в а
Техн.редактор Н.М.К а л е н ю к
Корректор О.А.А б р а г а м о в а

Подписано в печать 14.12.87 г. Формат 60x84 1/16.
Бумага оберточная белая. Печать оперативная.
Усл.п.л. 1,62. Уч.-изд.л. 1,5. Т. 500 экз.
Заказ 1581 Бесплатно.

Куйбышевский ордена Трудового Красного Знамени
авиационный институт имени академика С.П.Королева,
г. Куйбышев, ул. Молодогвардейская, 151.

Типография им. В.П.Мяги Куйбышевского полиграфического
объединения. 443099, г. Куйбышев, ул.Венцека,60.