Министерство образования и науки Российской Федерации
федеральное агентство по образованию
государственное образовательное учреждение высшего профессионального образования
«ВОСТОЧНО-СИБИРСКИЙ ГОСУДАРСТВЕННЫЙ
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Иринчеев А.А., Мангадаев А.М.

Паскаль в примерах.

Учебное пособие.

Издательство ВСГТУ
Улан-Удэ
2004

УДК:681.3.06(075.3)
ВБК:32.973-018.1я723
Иринчеев А.А., Мангадаев А.М. Паскаль в примерах. Учебное пособие. ВСГТУ, - Улан-Удэ, 2005, 76 с.
ISBN

В настоящем пособии рассмотрены вопросы программирования на языке Паскаль, приведены типовые примеры реализованных программ. Помимо задач и примеров пособие содержит анализ некоторых текстов программ и справочные данные по языку программирования.

Все необходимые дополнительные данные приведены на дискете.

Пособие ориентировано на студентов 1-2 курсов, изучающих язык программирования Паскаль, а также для школьников старших классов.

Язык программирования, Паскаль, алгоритмизация, блок-схема, трассировка.

Рецензенты: Розова Н.В., к.ф.-м.н., доцент, заведующий кафедрой информатики, вычислительной техники и прикладной математики Читинского государственного университета,
Степанов Б.М., к.т.н., доцент, заместитель заведующего кафедрой Информационных технологий Бурятского государственного университета.

«Рекомендовано дальневосточным региональным учебно-методическим центром а качестве учебного пособия для студентов специальностей 140205 – Электроэнергетические системы и сети, 140211 – Электроснабжение, 140101 – Тепловые электрические станции» вузов региона.

Протокол № 24 от 17.05.2005 г.

ББК

ВВЕДЕНИЕ

Настоящее пособие предназначено для студентов младших курсов и школьников старших классов изучающих язык высокого программирования Паскаль. Структура учебного пособия состоит из двух частей – основная часть на бумажном носителе представляет основы языка программирования, вторая часть на дискете представляет вспомогательный материал, дополняющий тестовую часть пособия.

По своей структуре все разделы одинаковы и отражают процесс обучения программированию. Они содержат теоретическую часть, в которой кратко изложены основные аспекты языка по рассматриваемой теме, практические примеры реализации с полным текстом программ.

Последовательность рассмотрения материала позволяет приступить к практическому программированию с использованием ЭВМ.

Рассмотрен стандартный Паскаль без использования дополнительных возможностей, что позволяет использовать данное учебное пособие для обучения программированию по любой версии языка.

В данном учебном пособии приведено большое количество примеров, программ с листингом делает изучение языка программирования удобным и доступным.

Каждый параграф снабжен задачами для самостоятельного решения, приведенные в приложении на электронном носителе.

Основная часть состоит из 6 разделов.

В первом разделе рассмотрены вопросы представления информации в компьютере с приведением практических примеров.

В втором разделе рассмотрены язык программирования Паскаль. Приведены основные понятия о языке программирования, приведены практические примеры реализации по основным разделам языка.

Начиная с третьего раздела информация представлена на электронном носителе, что позволит пользователю более эффективно изучать язык программирования.

В третьем разделе приведены листинги примеров практической реализации поставленной задачи. Примеры приведены как на языке программирования, так и в редакторе WORD и могут быть использованы в качестве раздаточного материала.

Четвертый раздел представляет расширения Паскаль в области графики. Представлена библиотека Cgaph. Приведены практические примеры реализации.

В пятом разделе для самостоятельной работы представлены задачи для самостоятельного решения, приведены примеры составления трассировок программ.

Шестой раздел рассматривает среду программирования ТурбоПаскаль. Приведены основные опции меню, коды ошибок.

1. ПРЕДСТАВЛЕНИЕ ИНФОРМАЦИИ В КОМПЬЮТЕРЕ

В памяти компьютера числа хранятся в виде битов. Каждый бит может принимать значение одной двоичной цифры. Следовательно, значением бита может быть ноль или единица. Восемь битов объединены в байт. Максимальное число, которое можно записать при помощи восьми двоичных цифр – **11111111**, что соответствует десятичному числу **255**, минимальное - ноль. Поэтому, значением байта может быть число от нуля до 255.

Переменные хранятся в памяти. Так как переменные различных типов могут принимать различные значения, то для их хранения нужен разный объем памяти. Память под переменные выделяется в байтах (целое число). Например, значением переменной типа *CHAR* может быть любой из 256 символов. Поэтому для хранения переменной такого типа достаточно одного байта. Значением переменной типа *INTEGER* может быть число от **– 32768** до **+32767** (65536 значений). Для хранения переменной этого типа требуется два байта. Очевидно, что чем больше диапазон значений типа, тем больше байтов нужно для хранения переменной типа, Объем памяти и диапазон значений для переменных различных типов приведены в таблице 1.1.

Таблица 1.1.

Тип переменной	Занимаемая память, байт	Диапазон значений
CHAR	1	Любой символ
STRING	256	Строка до 256 символов
STRING [N]	1 x n	Строка до n символов
BYTE	1	0÷255
WORD	2	0÷65535
INTEGER	2	-32768 ÷32767
LONGINT	4	- 2147483648 ÷2147483647
REAL	6	2.9E-39 ÷1.7E38
SINGLE	4	1.5E-45 ÷3.4E38
DOUBLE	8	5.0E-324 ÷1.7E308
EXTENDED	8	3.4E-4932 ÷1.1E4932

В программе для одного и того же значения можно использовать переменные разного типа (при этом они будут занимать разный объем памяти). Например, если в программе используется переменная **Day** (число месяца), то для нее можно задать типы **byte, integer** или **longint**. В первом случае будет занят один байт памяти, во втором – два. В третьем – четыре. Однако реально будет использоваться только один байт, а остальные – просто заняты. Поэтому следует подбирать наиболее подходящий тип для каждой переменной. Особое внимание нужно обращать на описание строковых переменных и массивов.

Выделяя память для строковых переменных, помните, что если не указана предельная длина строки, то переменной выделяется 256 байтов. При объявлении переменной, предназначенной, например, для хранения имени человека, следует писать name: string[30], а не name: string.

Каждому массиву программы выделяется память, объем которой определяется как типом элементов массива, так и их количеством. Например, для хранения двухмерного массива вещественных чисел (например, 20x20) нужно более трех килобайт памяти (20x20x8=3200).

1.1. Позиционные системы счисления

Системы счисления могут быть как позиционные, в которых значение зависит от позиции цифр, так и непозиционные, где такая зависимость отсутствует вообще или используется не всегда.

Во всех вычислительных машинах применяется позиционная система счисления.

В позиционной системе счисления каждое число представляется последовательностью цифр, причем каждой цифре x_i присваивается определенный вес b^i , где b- основание системы:

$$D=x_n * b^n + x_{n-1} * b^{n-1} + \ldots + x_0 * b^0 = x_n\, x_{n-1} \ldots x_0$$

Например, число 2003 представляется в системе с десятичным основанием (в десятичной системе), как

$$2003 = 2.003 * 10^3,$$

в двоичной системе как

$$2003=1*2^{10}+1*2^9+1*2^8+1*2^7+1*2^6+0*2^5+1*2^4+0*2^3+0*2^2+1*2^{1+}1*2^0 = 11111010011_2$$

в восьмеричной системе как

$$2003=3*8^3+7*8^2+2*8^1+3*8^0 =3723_8$$

в шестнадцатеричной системе как

$$2003=7*16^2+15*16^1+3*16^0 =7D3_{16}$$

Любое число в позиционной системе счисления, не считая крайних нулей, представляются единственным образом. Крайняя слева цифра называется цифрой старшего разряда, крайняя справа – цифрой младшего разряда.

Число в позиционной системе счисления представляется либо степенным рядом

$$D_b = \sum_{k=-l}^{m-1} x_k b^k$$

либо схемой Горнера:

$$D_b^{\text{int}} = \sum_{k=0}^{m-1} x_k b^k = (\ldots((x_{m-1}b+x_{m-2})\, b + x_{m-3})\, b + x_0;$$

$$D_b^{decimal} = \sum_{k=-L}^{-1} x_k b^k = b^{-1}(x_{-1}+b^{-1}(x_{-2}+\ldots+b^{-1}(x_{-L+1}+x_{-L}b^{-1}))),$$

где x_i – любая цифра из алфавита системы с основанием b;

m, L – число разрядов соответственно для целой и дробной части числа.

В современных компьютерах используются позиционные системы счисления с основаниями 2, 8, 10 и 16. В таб. 1.2 приведены возможные способы изображения первых 16 чисел во всех четырех системах счисления.

Для того, чтобы без затруднения представить числа по степенному ряду, необходимо выучить степени числа 2, 8, 16 (см. таб. 1.3.)

Сложение и вычитание положительных чисел в различных системах счисления очень похожи на десятичную.

Таблица 1.2.

Способы представления чисел

Двоичные числа D_2	Восьмеричные числа D_8	Десятичные числа D_{10}	Шестнадцатеричные числа D_{16}
0	0	0	0
1	1	1	1
10	2	2	2
11	3	3	3
100	4	4	4
101	5	5	5
110	6	6	6
111	7	7	7
1000	10	8	8
1001	11	9	9
1010	12	10	A
1011	13	11	B
1100	14	12	C
1101	15	13	D
1110	16	14	E
1111	17	15	F

Задачи

1. Представить в двоичной форме числа 48, 57, 511, 121.

Разложим числа по степенному ряду двойки:

$48 = 32+16 = 1*2^5+1*2^4 = 110000_2$;

$57 = 32+16+8+1 = 1*2^5+1*2^4+1*2^3+1*2 = 111001_2$;

$511 = 256+128+32+16+8+4+2+1 = 1*2^8+1*2^7+1*2^6+1*2^5+1*2^4++1*2^3+1*2^2+1*2^1+1*2^0 = 11111111_2$;

$121 = 64+32+8+4+2+1 = 1*2^6+1*2^5+0*2^4+1*2^3+1*2^2+1*2^1+$
$+1*2^0 =1101111_2$.

2. Представить в восьмеричной форме числа 48, 57, 511, 121.

Разложим числа по степенному ряду восьмерки:

$48 = 6*8 = 6*8^1+0*8^0 = 60_8$;

$57 = 56+1 = 7*8^1+1*8^0= 71_8$;

$511 = 7*64+7*8+7*1 = 7*8^2+7*8^1+7*8^0 = 777_8$;

$121 = 64+ 7*8+1 = 1*8^2+7*8^1 +1*8^0 = 171_8$.

3. Представить в шестнадцатеричной форме числа 48, 57, 511, 121.

Разложим числа по степенному ряду шестнадцати:

$48 = 3*16 = 3*16^1+0*16^0 = 30_{16}$;

$57 = 3*16+9 = 3*16^1+9*16^0 = 39_{16}$;

$511 = 256+15*16+15*1 =16^2+15*16^1+15*16^0 = 1FF_{16}$;

$121 = 7*16+9 =7*16^1+9*16^0 = 79_{16}$.

Таблица 1.3.

Степенные ряды

Степень двойки	Степень восьмерки	Степень шестнадцати
$2^0=1$	$8^0=1$	$16^0=1$
$2^1=2$	$8^1=8$	$16^1=16$
$2^2=4$	$8^2=64$	$16^2=256$
$2^3=8$	$8^3=512$	$16^3=4096$
$2^4=16$	$8^4=4096$	
$2^5=32$		
$2^6=64$		
$2^7=128$		
$2^8=256$		
$2^9=512$		
$2^{10}=1024$		

1.2. Перевод целого числа из десятичного счисления в другое

Предположим, что у нас есть целое число A, представленное с помощью основания m: A_m. Тогда в соответствии со схемой Горнера, его можно представить с помощью нового основания n как:

$A_m = A_n = (\ldots((a_{i-1}*n+a_{i-2})*n + a_{i-3})*n + \ldots a_1)*n + a_0$.

При переводе необходимо найти такие a_i в новой системе счисления.

Для того, чтобы перевести число A_m в новое счисление n, разделим его на n:

$A_m/n = A_n/n = (\ldots((a_{i-1}*n+a_{i-2})*n + a_{i-3})*n + \ldots a_2)*n + a_1 + a_0/n$.

Получаем:

$A_m = A_n = Am^1 * n + a_0$,

где Am^1 – целая часть от деления, а a_0 – остаток. Это остаток является первой младшей цифрой в новой системе счисления.

При последующих делениях получим новую целую часть и новый остаток

$Am^1 = Am^2 * n + a_i$,

где a_i – следующая младшая цифра.

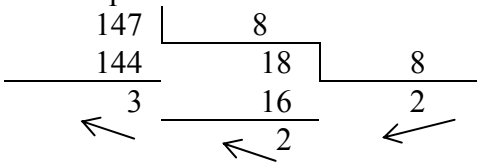
Деление проводится до тех пор, пока целая часть от деления не станет меньше n, в этом случае последняя целая часть дает цифру старшего разряда числа.

Правила перевода целых чисел:

1. Последовательно делить заданное число и получаемые целые числа на новое основание счисления до тех пор, пока целая часть не станет меньше нового основания счисления.
2. Полученные остатки от деления, представленные цифрами из нового счисления, записать в виде числа начиная с последней целой части.

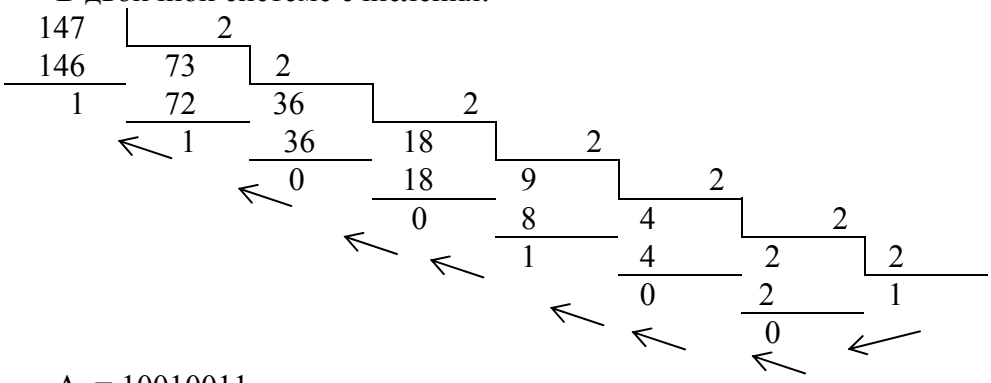
Пример. Представить число 147 в разных счислениях.

В восьмеричной системе счисления:



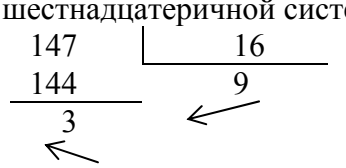
$A_8 = 223_8$.

В двоичной системе счисления:



$A_2 = 10010011_2$

В шестнадцатеричной системе счисления:



$A_{16} = 93_{16}$

1.3. Перевод дробного числа из десятичного счисления в другое

Предположим, что есть дробно число A, представленное с помощью основания m: A_m. Тогда в соответствии со схемой Горнера, его можно представить с помощью нового основания n как:

$A_m = A_n = n^{-1} (a_{i-1} + n^{-1} * (a_{i-2} + n^{-1} * (a_{i-3}) + \ldots + a_{-i} * n^{-1})))$.

При переводе на другую систему счисления задача – нахождение таких a_{-i} в новой системе счисления.

Для того, чтобы перевести это число A_m в новое счисление n, умножим его на n:

$A_m * n = A_n * n = a_{-1} + n^{-1} * (a_{-2} + n^{-1} (a_{-3} + \ldots + a_{-i} * n^{-1}))$.

Иначе говоря,

$A_m = A_n = Am^1 + \mathbf{a_1}$,

где Am^1 - дробная часть при умножении, а $\mathbf{a_1}$– целая часть.

Это целая часть является первой старшей цифрой в новой системе счисления.

При последующих умножениях получаем новую целую часть и новую дробную часть

$Am^2 = Am^1 + \mathbf{a_2}$,

где $\mathbf{a_2}$ – следующая старшая цифра.

Умножение проводим до тех пор, пока дробная часть не станет нулевой.

Правила перевода дробных чисел:

1. Последовательно умножать данное число и получаемые дробные части произведений на основание новой системы до тех пор, пока дробная часть произведений не станет равна нулю или не будет получено требуемое по условию количество разрядов.

2. Полученные целые части являются разрядами числа в новой системе счисления.

3. Составить дробную часть числа в новой системе, начиная с целой части первого произведения.

Пример.

Представить $A_{10} = 0.125$ в разных счислениях:

0	125
	* 2
0	250
	* 2
0	500
	* 2
1	000

$A_2 = 0.001$

0	125
	* 8
1	000

$A_8 = 0.1$

0	125
	* 16
2	000

$A_{16} = 0.2$

Необходимо отметить, что не каждое число может быть точно выражено в новой системе счисления; поэтому иногда вычисляют только требуемое количество разрядов в дробной части, округляя последний разряд.

1.4. Перевод чисел в десятичную систему счисления

Перевод чисел в десятичную систему счисления можно сделать: по формулам степенного ряда, по схеме Горнера или по правилам перевода.

Пример. Дано $A_1 = 100100.1001_2$, $A_2 = 234.5_8$, $A_3 = ABC.E_{16}$ и переведем их разными способами в десятичный код.

Перевод по степенному ряду:

$A_1 = 100100.1001_2 = 2^5 + 2^2 + 2^{-1} + 2^{-4} = 36 + 1/2 + 1/16 = 35.5625_{10}$

$A_2 = 234.5_8 = 2*64 + 3*8 + 4*1 + 5*1/8 = 156.625_{10}$

$A_3 = ABC.E_{16} = 10*256 + 11*16 + 12*1 + 14* 1/16 = 2748.875_{10}$

Перевод по схеме Горнера:

$A_1 = 100100.1001_2 = (((1*2+0)*2+0)*2+1)*2+0)*2+0+$
$1/2*(1+1/2*(0+1/2*(0+1*1/2))) = 36.5625_{10}$

$A_2 = 234.5_8 = (2*8+3)*8+4+5*1/8 = 156.625_{10}$

$A_3 = ABC.E_{16} = (10*16+11)*16+12+14*1/16 = 2748.875_{10}$

Использование правил перевода:

для формирования целой части:

100100	1010		234	12	
1010	11		12	17	12
10000			112	12	1
1010			106	5	
110			6		

$A_1^{ц} = 36$ $A_2^{ц} = 156$

ABC	A			
A	112	A		
	A			
B	72	1B	A	
A	6E	14		
1C	4	7	2	
A				
8				

$A_3^{ц} = 2748$

Главное неудобство использования правила перевода заключается в том, что необходимо знать правила деления и умножения в различных системах счисления. При преобразованиях по степенному ряду и по схеме Горнера используются только десятичные арифметические операции.

1.5. Форматы данных и машинные коды чисел

Числа в двоичных кодах представляются с фиксированной точкой или запятой (естественной формы числа) и с плавающей точкой и запятой (нормальной формы числа).

0	1001
	*1010
101	1010
	*1010
110	0100
	*1010
010	1000
	*1010
101	0000

	5
	* 12
	2
	* 12
	4
	* 12
	0

0	E
	* A
8	C
	* A
7	8
	* A
5	0

$A_1^{д} = 0.5625$ $A_2^{д} = 0.625$ $A_3^{д} = 0.875$

Для чисел в естественной форме положение точки жестко фиксируется:

- для целых чисел точка располагается справа от младшего разряда:

$0000000000000000_2 = 0_{10}$

$0111111111111111_2 = 32767_{10}$

- для правильных дробей – перед старшим разрядом:

$0.000000000000000_2 = 0_{10}$

$0.000000000000001_2 = 0.000\ 030\ 517\ 578 = 125_{10}$

- для смешанных дробей – в определенном месте, определяющем целую часть числа от дробной:

$000000.0000000000_2 = 0.0_{10}$

$000001.0000000001_2 = 1.000\ 976\ 563_{10}$

Наиболее часто такая форма используется для целых числа и целых чисел без знака. Количество разрядов может быть либо 16 (вид H), либо 32 (вид F).

Во всех форматах знак числа помещается в старший разряд и кодируется как 0 – знак положительного числа, либо как 1 – знак отрицательного числа. Знак определяется от самого числа воображаемой точкой (рис. 1.1.).

Фиксированная точка позволяет задавать число только в строго определенном диапазоне.

В формате H числа можно задавать

от $1111\ 1111\ 1111\ 1111_2$ до $0111\ 1111\ 1111\ 1111_2$

т.е от -32767_{10} до 32767_{10}, или от $-2^{15}-1$ до $2^{15}-1$

В формате F числа находятся в интервале

от $1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_2$

до $0111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_2$

т.е. от $-7FFFFFF_{16}$ до $7FFFFFFF_{16}$

Представление в шестнадцатеричной системе для формы F предпочтительнее двоичной системы.

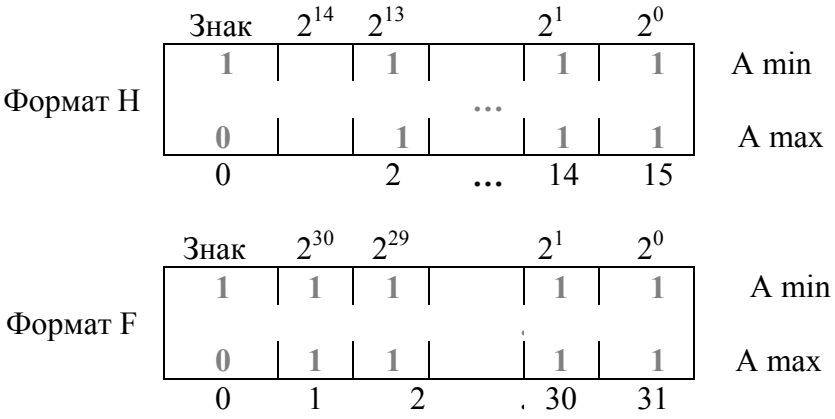


Рис. 1.1. Форматы чисел с фиксированной точкой

Пример. Определить, какие из следующих шестнадцатеричных чисел положительные, а какие отрицательные: 9754, 157, ADF, 7654AD, DFEA.

Знак числа определяется по первой цифре: если оно меньше 8 (< 1000), то число положительное, если значение от 8 до F, то отрицательное. Таким образом, получаем 9754<0, 157>0, ADF<0, 7654AD>0, DFEA<0.

1.6. Нормальная форма числа, или представление чисел в форме с плавающей точкой

Для расширения диапазона рассматриваемых чисел по сравнению с естественной формой чисел используется формат с плавающей точкой или нормальная форма. Любое число в этом формате представляется, как

$$A = \pm m_a E^{\pm P_a},$$

где m_a – мантисса числа A;

E – основание системы счисления;

$\pm$ Pa – порядок.

Все эти величины – двоичные числа без знака.

На рис.1.2. приведен формат числа в нормальной форме. Старший разряд (нулевой) содержит знак мантиссы, первый разряд – знак порядка, 6 разрядов, со второго по седьмой, определяют значение порядка, а остальные – мантиссу. Нормальная форма может быть представлена коротким форматом E (4 байта), длинным форматом D (8 байт) и повышенной точностью (16 байт). Во всех этих формах представления первый байт остается постоянным, изменяется только область, отведенная под мантиссу.

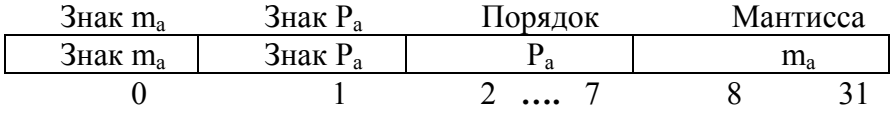


Рис.1.2. Нормальная форма числа.

При таком представлении число 0 может быть записан 64 разными способами, т.к. для этого подходят любые значения порядков $0 * 2^0 = 0 * 2^1 = \ldots = 0 * 2^{63}$. А другие числа могут иметь много различных форм записи.

Например, $1536_{10} = 3*2^9 = 6*2^8 = \ldots = 768*2^1$.

Для однозначного представления чисел мантиссу нормализуют, т.е накладывают ограничение

$$1/E \le m < 1.$$

Это ограничение означает, что мантисса представляет собой правильную дробь и содержит хотя бы одну значащую цифру после запятой, отличную от нуля. Нормализованным представлением нуля является такое представление, при котором во всех разрядах находятся нули.

При использовании нормальной формы для части компьютеров характерно смещение оси порядков в область положительных значений. В этом случае арифметические действия производятся над порядками, не имеющими знака. В нормальной форме под значение порядка отводится 7 разрядов, один из них знаковый. Таким образом, значение порядка может лежать в интервале $2^6 \le P \le 2^6 - 1$, т.е от –64 до 63.

Сместив порядок на $2^6 = 64 = 40_{16}$, получается интервал возможных значений $0 \le P \le 2^7 - 1 = 127$. Смещенный порядок на 40_{16} называется характеристикой и вычисляется как $P_x = P + 40$.

Если характеристика равна 40, то порядок равен нулю; если характеристика меньше 40, то порядок отрицателен; если больше – то положителен.

2. ОСНОВНЫЕ ПОНЯТИЯ О ЯЗЫКЕ ПАСКАЛЬ

Алфавит стандартного языка ПАСКАЛЬ содержит следующие символы:

- 26 букв латинского алфавита:
  A,B,C,D,E,F,G,H,I,J,K,M,L,N,O,P,R,S,T,U,V,W,X,Y,Z;
- арабские цифры: 0,1,2,3,4,5,6,7,8,9.
- 32 буквы русского алфавита;
- специальные символы: + - * / = > < : ; . , ^ # $ { } [ ] () @ _
- служебные слова:

absolute, and, array, begin, case, const, div, do, downto, else, end, external, file, for, forward, function, goto, mod, nil, not, of, or, procedure, record, repeat, set, shl, shr, string, then, if, implementation, inline , interface, interrupt, in, label, to, type, until, uses, var, while, with;

Знаки операций:

арифметических:

- + (сложение), - (вычитание),
- (умножение),
- / (деление), div (деление нацело) с отбрасыванием остатка,
- mod (нахождение остатка от деления нацело);
- *отношения:* > (больше), < (меньше), >= (больше или равно), <= (меньше или равно), = (равно), <> (не равно);
- *логических :* not (отрицание), or (логическое сложение), and (логическое умножение).

Старшинство операций определяется:

- NOT - высший порядок;
- * , / , DIV, MOD, AND - второй приоритет;
- + , - , OR - третий приоритет;
- = , <>, >, <, <=, >= - четвертый приоритет;

Числа имеют запись, близкую к обычной математической, и могут быть *целого(integer)* или вещественного (real) типа. Положительный знак числа опускается.

Переменные целого типа записываются последовательностью цифр (например, + 4, 4, - 100, 1999, - 1001).

Переменные вещественного типа имеют две формы записи: с фиксированной и плавающей точкой. Числа с фиксированной точкой записываются в виде целой и дробной частей, разделенных десятичной точкой. Например, 2.65, - 11.862, 0.5, -0.65. Запись числа не может начинаться или заканчиваться десятичной точкой. Числа с плавающей точкой используются для записи чисел в широком диапазоне значений (от очень маленьких до очень больших). Десятичный порядок числа записывается буквой E.

Целое число занимает 2 байта, вещественное - 4 байта, символьное - 2 байта.

Например:

Запись на языке ПАСКАЛЬ и обычная математическая запись

2,56 E6 $2.56*10^6$ или 2560000

0.25 E-4 $0.25*10^{-4}$ или - 0.000025

- 1.8 E5 $-1.8*10^5$ или - 180000

Имя (или идентификатор) служит для обозначения каких-либо объектов, чаще всего для записи констант, типов и выражений. В языке ПАСКАЛЬ различают два вида имен: *стандартные и даваемые пользователем ЭВМ*. Имя должно начинаться с буквы и содержать последовательность букв, цифр или символов подчеркивания.

Например, x, summa, de11.

Прописные буквы (A, ..., Z) воспринимаются тождественно строчным (a, ...,z). Следовательно, имена primer или PRIMER воспринимаются одинаково. Имена могут иметь любую длину, однако только первые 63 символа являются значащими.

Стандартные имена заложены в языке для обозначения стандартных объектов (например, стандартных программ, функций и т.д.).

Стандартными именами являются ABS, ARCTAN, BOOLEAN, CHAR, CHR, COS, CLOSE, DISPOSE, EOF, EOLN, EXP, FALSE, FORWARD, GET, INPUT, INTEGER, LN, MAXINT, NEW, ODD, ORD, OUTPUT, PACK, PAGE, PRED, PUT, READ, READLN, REAL, RESET, REWRITE, ROUND, SIN, SQR, SQRT, SUCC, TEXT, TRUE, TRUNC, UNPACK, WRITE, WRITELN.

Пользовательское имя - имя даваемое пользователем. Следует отметить, что в качестве имени нельзя давать служебные слова и стандартные имена. Имя состоит из букв и цифр и обязательно начинается с буквы, латинской или русской, входящей в состав языка. Имя может иметь произвольную длину (в некоторых версиях языка длина имени может быть ограничена). Лучше всего использовать имена, состоящие из одного слова и не более 63 символов. Например: СКОРОСТЬ, SUMMA, WW1, srep1a.

К основным элементам данных относятся константы и переменные.

Переменные используются для записи значений изменяющихся в программе и могут изменять свое значение в ходе выполнения программы.

По структуре различают простые переменные и переменные с индексом.

Простые переменные записываются своими именами. Они могут принадлежать к различным типам переменных.

Пример записи: X1, X1A22, PRIM!, SUMMA, REZULTAT.

Переменные с индексом являются элементами массива, состоящего из упорядоченного набора значений одного типа, имеющих общее имя, например $\mathbf{a_{11},a_{12},...,a_{1n}}$.

Программа, написанная на языке ПАСКАЛЬ, оперирует некоторыми объектами, называемыми данными. Тип данных определяет возможные значения констант, переменных, функций, выражений, принадлежащих к этому типу, форму представления в ЭВМ и операции, которые могут выполняться над ним. Все типы данных можно разделить на *простые* и *сложные*.

К *простым* относятся стандартные типы. Стандартными являются целый INTEGER, вещественный REAL, логический BOOLEAN, символьный CHAR.

Сложные типы данных представляют собой различные комбинации простых типов (массивы, множества, записи, файлы).

Типы переменных делятся на **целые**, **вещественные**, **логические**, **символьные** и т.д.

В языке ПАСКАЛЬ целый тип обеспечивает задание целых чисел. В зависимости от возможности ЭВМ существуют ограничения на диапазон целых чисел. Определено некоторое целое число MAXINT такое, что значение любой целой переменной или константы должно находиться в диапазоне от - MAXINT до + MAXINT. В противном случае будет зафиксирована ошибка. Над переменными целого типа определены следующие *арифметиче-*

ские операции: **сложение, вычитание, умножение, деление, div, mod.** Все эти операции вырабатывают результат целого типа, кроме операции деления, результат которой всегда будет вещественного типа. Определены также *операции отношения*: **равно, не равно, меньше, больше, меньше или равно, больше или равно.**

С аргументами целого типа могут использоваться следующие стандартные функции, приведенные в таблице 2.1.

Определены также функции:

ODD(X) - вырабатывает результат булевского типа: для четного аргумента - FALSE, для нечетного аргумента - TRUE;

SUCC(X) - вырабатывает следующее целое число (т.е. число, на единицу больше X);

PRED(X) - вырабатывает предыдущее целое число (т.е. число, на единицу меньше X);

Переменные и константы типа в ПАСКАЛЕ употребляются в том же смысле, что и в математике действительные числа.

Вещественные переменные обладают двумя важными характеристиками - диапазон значений и точность, которые определяются структурными особенностями конкретной ЭВМ.

Над переменными типа *определены* следующие *математические операции*: **сложение, вычитание, умножение, деление, div, mod.**

Определены также *операции отношения*: **равно, не равно, меньше, больше, меньше или равно, больше или равно.**

При обращении к стандартным функциям необходимо записать имя функции, а в скобках указать аргумент. Перечень стандартных функций, типы аргументов и функций приведены в таблице 2.1.

Таблица .2.1

Стандартные функции в ПАСКАЛЕ

МАТЕМАТИЧЕСКАЯ ЗАПИСЬ	ЗАПИСЬ ФУНКЦИИ НА РАСКАЛЕ	ТИП АРГУМЕНТА	ТИП РЕЗУЛЬТАТА
sin x	SIN(X)	REAL	REAL
cos x	COS (X)	REAL	REAL
ln x	LN(X)	REAL	REAL
$\sqrt{x}$	SQRT(X)	REAL	REAL
arctg x	ARCTAN(X)	REAL	REAL
e^x	EXP(X)	REAL	REAL
x^2	SQR(X)	REAL	REAL
|x|	ABS(X)	INTEGER, REAL	INTEGER, REAL
	TRUNC(X)	INTEGER, REAL	INTEGER

Определены стандартные функции преобразования значений типа в значения целого типа:

TRUNC(X) - вырабатывает целый результат путем отбрасывания дробной части аргумента;

ROUND(X) - вырабатывает целый результат путем округления до ближайшего целого.

Пример, если X = 21.53, то TRUNC(X) = 21, ROUND(X) = 22;

если X = -2.7, то TRUNC(X) = - 2, ROUND(X) = -3.

$$\text{round}(x) = \begin{cases} \text{trunc}(x + 0.5), & \text{при } x \geq 0, \\ \text{trunc}(x - 0.5), & \text{при } x < 0. \end{cases}$$

Значениями "с*имвольного типа*" являются элементы конечного и упорядоченного набора знаков. Существуют две стандартные, обратные по отношению друг к другу функции, называемые функциями преобразования:

ORD(X) - дает порядковый номер символа X в упорядоченном множестве символов.

Порядковые номера следуют по возрастанию без пропусков. При конкретной реализации языка использование функции позволяет получить, например:

ORD(‘ : ‘) = 58,
ORD(‘ 5 ‘) = 53,
ORD(0) = 240,
ORD(9) = 249,

Функция CHR(I) - дает символ, стоящий под номером I в упорядоченном множестве символов.

Например: CHR(66) = ‘ B ‘,CHR(57) = ‘ 9 ‘.

Очевидно, что CHR(ORD(S)) = S и ORD(CHR(I)) = I.

Над переменными символьного типа определены *операции отношения*. Пусть C1 и C2 - переменные символьного типа. Отношение C1<C2 истинно тогда и только тогда, когда ORD(C1) < ORD(C2).

К аргументам символьного типа применимы также *стандартные функции*: **PRED(S)** и **SUCC(S)**. Функция PRED(S) дает предыдущий символ, а функция SUCC(S) - следующий.

Например: **PRED(‘ B ‘) = ‘ A ‘ , а SUCC(‘ 9 ‘) = ‘ : ‘.**

Справедливы равенства:

PRED(S) = CHR(ORD(S) - 1) и SUCC(S) = CHR(ORD(S) + 1).

Переменные "*булевского типа*" могут принимать одно из двух значений: **TRUE** (истина) или **FALSE** (ложь). Над ними определены три *логические операции*:

NOT (отрицание - **НЕ**),
AND (конъюнкция - **И**),
OR (дизъюнкция - **ИЛИ**).

Логические операции OR или AND выполняются над двумя величинами, а операция NOT - над одной.

Логическое сложение дает истинный результат, если хотя бы одна из логических величин (A или B) имеет истинное значение. Если обе величины (A или B) имеют ложное значение, то и результат операции будет ложным.

Логическое умножение дает истинный результат только в том случае, если обе величины (A или B) истинны. Если хотя бы одна величина (A или B) ложна, то результат также будет ложным.

Логическое отрицание дает ложный результат, если величина имеет истинное значение, и наоборот.

Результаты операций над логическими данными сведены в таблицу 2.3.

Константы символьного типа записываются литерами, заключенными в кавычки (апострофы). Символ апострофа записывается двойными кавычками.

Пример записи констант: ‘ B ‘ ‘D ‘ ‘6 ‘

Логические данные и операции над ними имеют важное значение в информатике, так как позволяют внести в расчеты элементы человеческой логики. В информатике принято

1 - истина, 0 - ложь.

Значение FALSE и TRUE можно рассматривать как упорядоченное множество, состоящее из двух элементов. Целый, символьный, булевский типы называются так же дискретными типами, поэтому над ними разрешены операции отношений:

ORD(FALSE) = 0;
ORD(TRUE) = 1;
SUCC(FALSE)= TRUE;
PRED(TRUE) = FALSE.

Константы не изменяют своего значения в процессе выполнения программы. Она может быть задана явно своим значением или обозначена именем.

Константы бывают целые, действительные, логические, символьные, строковые.

Таблица 2.2

Типы аргументов и функций стандартных функций в ПАСКАЛЕ

Обращение	Выполняемые функции	Тип	
		аргумента	функции
ABS(X) SQR(X)	|x| x^2	REAL или INTEGER	REAL или INTEGER
SIN(X) COS(X) EXP(X) LN(X) SQRT(X) ARCTAN(X)	sin(x) cos(x) e^x Ln(x) $\sqrt{x}$ arctg x	REAL или INTEGER	REAL
TRUNC(X) ROUND(X)	Выделение целой части числа Округление числа	REAL	INTEGER
PRED(X) SUCC(X)	Нахождение: предыдущего элемента последующего элемента	INTEGER или CHAR или BOOLEAN	INTEGER или CHAR или BOOLEAN
ORD(X)	Определение порядкового номера символа X в наборе символов	CHAR или BOOLEAN	INTEGER
CHR(I)	Определение символа из набора символов по порядковому номеру i	INTEGER	CHAR
ODD(X)	Определение нечетности числа	INTEGER	BOOLEAN

Таблица 2.3

Таблица истинности

A	**B**	NOT **A**	**A** OR **B**	**A** AND **B**
TRUE	TRUE	FALSE	TRUE	TRUE
TRUE	FALSE	FALSE	TRUE	FALSE
FALSE	TRUE	TRUE	TRUE	FALSE
FALSE	FALSE	TRUE	FALSE	FALSE

Константы целого и действительного типов являются числами. Они могут иметь положительный или отрицательный знак и записываются числами или именами соответствующего типа.

Константы логического типа имеют одно из двух значений - TRUE (истинно) или FALSE (ложно),

Текстовые к*онстанты* (строки) записываются последовательностью символов (текстом), заключенной в кавычки. Имеющиеся в тексте кавычки дублируются. Разрешается использование последовательности символов, заключенных в апострофы, длиной не более 256 знаков (в зависимости от версии языка ПАСКАЛЬ).

Комментарий служит для пояснения программы или отдельных частей. Наличие комментариев делает программу более понятной и удобной для чтения. Комментарий языка

ПАСКАЛЬ - это последовательность символов, ограниченных слева парой символов **(*** и справа парой символов ***)**. Вместо круглых скобок допускается {..}.

(* .....*) - для комментария в несколько строк;

{ .....} - для комментария, записанного сразу после оператора.

Например. (* Определение площади фигуры *)

или { Координаты точки }.

Конструкция языка, задающая порядок выполнения действий над элементами данных называется выражением.

Выражение состоит из операндов - величин и выражений, над которыми производится операция.

Операции определяют действия, которые надо выполнить над операндами.

Операции бывают унарными и бинарными. В первом случае операции относятся к одному операнду и всегда записываются перед ними, во втором - операция выражает отношения между двумя операндами и записывается между ними.

Например:

- A - унарная, A + B - бинарная.

В качестве операндов могут быть константы, переменные, функции. Порядок выполнения операций определяется скобками, а при их отсутствии - согласно старшинству (приоритету) операций. Скобки заменяют приоритет.

Операции одного приоритета выполняются последовательно слева направо.

Примеры записи выражений:

Запись на языке ПАСКАЛЬ и Математическая запись

0.5+A	a+0,5
P-T/A-B	$\frac{p-T}{a}-в$
(1,25*SIN(X)+P)/(SQRT(A+B)/X)	$\frac{(1{,}25\sin(x)+p}{\frac{\sqrt{a+b}}{x}}$

С учетом старшинства операций и скобок при одних и тех же значениях операндов результаты выражения получаются различными.

Выражение 7-2*3 имеет значение 1, а выражение (7-2)*3 - значение 15. Выражение 16/4*2 имеет значение 8, а не 2 так как операции деления и умножения - одного приоритета и выполняются по порядку следования слева направо.

Типы результатов выражений в зависимости от типов операндов и выполняемых операций представлены в таблице 2.4.

Функция ORD(x) -функция для дискретных типов, следовательно для нее определены операции отношений. Так как все символы упорядочены, то для символов x_1 и x_2 справедливо выражение отношения ORD (x_1)< ORD (x_2).

Отношение означает, что порядковый номер символа x_1 меньше, чем x_2. Поэтому результат выполнения операции отношение имеет тип BOOLEAN.

Отношение ORD (‘ 3 ‘) > ORD (‘6 ‘) имеет результат FALSE, так как цифра 3 расположена раньше цифры 6, а следовательно, порядковые номера в отношении записываются как 243>246.

С помощью оператора описываются действия над данными, выполняемые для реализации алгоритма решения задачи. По функциональному назначению *операторы языка* можно подразделить на группы: присваивания, ввода - вывода, управления, определения функций и процедур (подпрограмм).

Оператор присваивания предназначен для вычисления нового значения некоторой переменной, а также для определения значения, возвращаемого функцией. При выполнения

оператора присваивания вычисляется выражение стоящее в правой части, и его значение присваивается переменной в левой части.

Таблица 2.4

Результаты операций

Операция	Знак операции	Тип	
		операндов	результат
Сложение	+	REAL или INTEGER	INTEGER, если оба операнда INTEGER, и REAL в противном случае
Умножение	*		
Вычитание	-		
Деление	/	REAL или INTEGER	REAL
Деление с отбрасыванием остатка	DIV	INTEGER	INTEGER
Вычитание остатка при делении чисел	MOD		
Отрицание	NOT	BOOLEAN	BOOLEAN
Дизъюнкция	OR		
Конъюнкция	AND		
Сравнение: на равенство на неравенство	= <>	Любой тип переменных и констант	BOOLEAN
Сравнение: больше меньше больше или равно меньше или равно	> < >= <=	Любой скалярный тип	BOOLEAN

При этом тип выражения должен соответствовать типу переменной. Для стандартных типов это означает, что типы должны совпадать. Кроме того, допускается присваивание переменной вещественного типа значение выражения целого типа.

Общая форма записи оператора

V:=A,

где V - имя переменной;

:= -знак присвоения;

A - выражение.

Например: T:= 567.78

Присвоение же переменной целого типа выражению вещественного типа запрещено.

Для преобразования значений типа в значение целого типа предназначены функции TRUNC(X) и ROUND(X).

Операторы ввода - вывода позволяют вводить в основную память исходные данные и на устройствах вывода информации получать результаты вычислений.

Операторы управления организуют управление последовательностью выполнения операторов программы.

Операторы вызова функций и процедур дают возможность разбивать программу на части, определять и именовать их.

По своему составу различают операторы двух типов: простые и структурные.

Простым считается оператор, который не содержит других операторов. К простым операторам относятся операторы присваивания, перехода, процедуры.

Структурным считается оператор, содержащий в качестве компоненты один или несколько операторов. К структурным операторам относятся операторы условные, выбора, цикла, составные.

Составной оператор представляет собой последовательность операторов, заключенную в ключевые слова BEGIN и END. В этой конструкции служебные слова BEGIN и END называются *операторными скобками.* Слово BEGIN выполняет роль открывающейся скобки, слово END - роль закрывающейся скобки. Составной оператор представляется как единый оператор. Его можно вставить в любое место программы, где допускается один оператор.

Любой из операторов составного оператора, в свою очередь, также может быть составным.

Следует обратить внимание, что после BEGIN и END **не ставится точка с запятой**; она ставится между операторами.

Форма записи данного оператора:

```
BEGIN
    оператор 1;
    оператор 2;
    .....
    оператор n-1;
    оператор n
END;
```

Пустой оператор - это оператор, не выполняющий никакого действия. Пустому оператору соответствует отсутствие записи на том месте, где по правилам должен быть какой-нибудь оператор. После него можно ставить символ точки с запятой, например:

```
A:= B;
R:=2;
;
K:=7.2;
```

Здесь третий оператор является пустым.

Составной и пустой операторы нередко применяются в условных операторах.

Программа на Паскале состоит из трех частей: *заголовок программы, раздел описаний и раздел операторов.* Структура программы представлена на рис. 2.1.

```
PROGRAM < имя>;

        раздел
      описаний

BEGIN
        раздел
     операторов
END.
```

Рис. 2.1. Структура программы на языке Паскаль

Заголовок содержит служебное слово PROGRAM, имя программы, задаваемое пользователем-программистом, и при необходимости, в круглых скобках имена стандартных процедур для связи программы с внешними устройствами ввода-вывода ЭВМ. Заканчивается заголовок символом “**;**“ (точка с запятой).

Раздел описаний предназначен для объявления всех встречающихся в программе данных и их характеристик (имена данных, их тип, возможные значения и др.). Этот раздел состоит из следующих разделов:

- объявление меток;
- объявление констант;
- объявление типов;
- объявление переменных;
- объявление процедур и функций.

Разделы **должны располагаться в строго названном порядке.** Объявление процедур и функций является одним разделом. Следует заметить, что не все перечисленные разделы обязательны в каждой программе. После каждого описания ставится символ “ **;** “ .

Структура программы

```
PROGRAM <  имя>;
LABEL (раздел меток);
CONST (раздел констант);
TYPE (раздел типов);
VAR (раздел переменных);
PROCEDURE, FUNCTION (раздел процедур и функций);
BEGIN
   оператор 1;
   оператор 2;
   оператор 3;
   .....
   оператор n-1;
   оператор n
END.
```

Раздел операторов заключается в операторные скобки вида BEGIN и END, при этом после END ставится точка. В разделе операторов записывается последовательность исполняемых операторов. Каждый оператор выражает действие, которое необходимо выполнить. Исполняемые операторы отделяются друг от друга символом “**;**“. Сама же программа записывается в свободной форме, операторы не привязаны к определенной позиции строки в отличие от многих других языков программирования.

В одной строке можно указывать несколько описаний или операторов. Допускается перенос с одной строки на другую частей описаний или операторов (но не разрешается разделять слова, константы и составные символы). В то же время рекомендуется программу записывать в такой форме, чтобы ее можно было легко читает и понимать. Для этого широко используются пробелы, пустые строки и комментарии. Рекомендуется смысловые части выделять одинаковыми отступами от начала строки. Так, для выделения заголовка программы, раздела описаний и раздела операторов удобно записывать слова PROGPAM, BEGIN, END с одной позиции строки. По отношению к ним соответствующие описания или операторы сдвигаются вправо. Желательно сдвиг делать на одинаковое число позиций от края или по отношению к предыдущему сдвигу.

Все операции ввода/вывода основаны на работе с последовательными файлами. Мы рассмотрим четыре процедуры ввода/вывода: READ, READLN, WRITE, WRITELN, использующие стандартные файлы ввода/вывода. Стандартный файл ввода имеет имя INPUT, вывода - имя OUTPUT. Эти имена передаются в виде параметров в заголовке программы. Как правило, программа без вывода не бывает, поэтому простейшая конструкция заголовка программы имеет вид

PROGRAM <имя> ;

Ввод в языке ПАСКАЛЬ может быть только бесформатный. Можно вводить данные только целого, и символьного типов. Ввод переменных логического типа не допускается.

Типы вводимых переменных должны соответствовать типам вводимых значений. Пробел воспринимается как литера. Поэтому после считывания последней литеры в строке следующая литера является недоступной. Попытка прочитать литеру приводит к чтению пробела.

Для задания переменным их числовых значений можно использовать оператор присваивания, например:

A:=9; B:= -6.789;

Однако в данном случае программа становится не универсальной, так как выполняется только при этих значениях переменных. Поэтому в разделе переменных описываются переменные A, B, C, а в разделе операторов им присваиваются соответствующие значения. Для выполнения программы при различных значениях переменной предназначен оператор ввода READ.

Как только во время выполнения программы встречается оператор READ, ЭВМ останавливается и ожидает ввода. Когда значения переменных введены, процесс выполнения программы продолжается. Оператор ввода имеет вид:

READ (a_1, a_2,...., a_n);

где a_1, a_2,...., a_n - переменные, которым последовательно присваиваются вводимые значения.

Числовые значения водятся **через пробел,** признаком окончания ввода является нажатие клавиши **Enter**. Обратите внимание: *числовые значения вводятся после набора на экране дисплея всей программы и после запуска ее на выполнение.*

Пусть переменным A, B, C необходимо присвоить значения в процессе выполнения программы: A=4, B=14, C=24.6. Оператор ввода примет вид

READ(A, B, C);

а числовые значения можно ввести следующим образом:

4 14 24,6 Enter.

Оператор READ (a_1, a_2,...., a_n); обеспечивает выбор данных в результате которой имена переменных a_1, a_2,...., a_n получают соответствующее значение.

Если вновь повторить запуск программы, то можно ввести любые другие значения, например: 15 -1.45 -2,6 Enter.

Переменные получат значение A=15, B=-1.45, C=-2.6 , при которых будет выполняться программа. Ни один оператор программы в этом случае не изменяется.

Если переменная описана как действительная (REAL), а ее значение является целым числом, то можно вводить число как целое и как действительное. *Машина сама преобразует целое число в действительное.*

Пробелы перед числом и между числами игнорируются, поэтому их можно ставить в любом количестве.

Допускается использование оператора ввода без параметров READLN.

Оператор READLN (a_1, a_2,...., a_n); обеспечивает выборку данных из стандартного файла INPUT, что после окончания выборки последней переменной осуществляет переход к началу новой строки файла.

При вводе значений переменного целого и действительного типов READ и READLN пропускает пробелы между значениями, а оператор READLN обеспечивает пропуск одной строки в стандартном файле INPUT и переход к началу новой строки.

Различают различные методы организации ввода чисел:

- с использованием одного оператора READ

READ(A,B,C);

числа вводятся через пробел;

- с использованием нескольких оператора READ

READ(A);

READ(B);
READ(C);
каждое число вводится индивидуально;
- комбинированно
READ(A,B);
READ(C);
READLN(A,B);
READLN(C);

Ввод чисел целых и вещественных типов осуществляется одинаково.

Можно выводить данные всех четырех стандартных типов (для булевого типа выводится константа TRUE или FALSE).

Для вывода данных из памяти ЭВМ на экран дисплея предназначен оператор вывода WRITE. Форма записи оператора

WRITE (b_1, b_2,.., b_n);

где - b_1, b_2,.., b_n - имена переменных, подлежащие выводу могут быть либо переменными, либо строкой символов, заключенной в апострофы.

Например, оператор

WRITE (‘ Значение B = ‘, B);

на экран дисплея выводит строку

Значение B =

а затем значение переменной B.

Если поясняющий текст не давать, то пользователь зачастую забывает, значения каких переменных нужно вводить или выводить. Особенно это сказывается при выполнении больших программ с большим количеством операторов ввода и вывода.

Оператор WRITE(b_1, b_2,.., b_n); выполняет вывод значений, соответствующие именам b_1, b_2,.., b_n в стандартный выходной файл. Выводимые значения размещаются на одной строке.

Часто используется оператор вывода без параметров

WRITELN

осуществляющий переход на новую строку экрана дисплея или принтера.

Последующий оператор вывода с параметрами будет выводить данные на новую строку экрана.

Оператор WRITELN(b_1, b_2,.., b_n); выполняет вывод значений, соответствующий, b_1, b_2,.., b_n в стандартный файл и после вывода последнего осуществляет переход к новой строке файла.

Оператор WRITELN; обеспечивает пропуск строки в файле и переход к началу следующей строки.

Имена переменных, записанные в операторе вывода, могут принадлежать целому, действительному, символьному или логическому типу.

Форма представления выводимых переменных определяется типом переменных:

значение величин целого типа выводится в обычной форме;

значение величин действительного типа - в виде нормализованного числа действительного типа с порядком;

значение логического типа - в виде логических значений TRUE или FALSE;

значение символьных переменных - в виде соответствующих символов.

Если в операторе вывода указывается общее число позиций (w) и не указывается количество позиций после запятой (d), то число выводится в экспоненциальной форме с шириной поля (w).

Пример:

WRITE(A); Выводится число 6.0E+02.

WRITE(A:8); Выводится число 600.

WRITELN(A:m:n); Выводится число 7.12.

где m - поля, отводимые под запись значения, включая десятичную точку и знак числа;

n - часть поля, отводимое под дробную часть числа.

Если *при выводе символьного типа* в явном виде не указывается количество позиций, под каждый символ отводится одна позиция.

Пример:

```
WRITELN(' s1= ',s1:5,' s2 = ',s2);
```

При выводе числа с фиксированной точкой указывается ширина поля, отводимая под все значения и дробную часть числа.

При выводе нескольких значений на каждой строке для наглядности задаются необходимым количеством пробелов. Для этого записывают оператор в виде:

```
WRITE(' ':g);
```

где g - константа целого типа, указывающая число пробелов.

При выводе значений булевого типа выводится TRUE или FALSE.

Пример:

WRITE(A < B:7); Напечатает в отведенных семи позициях слово TRUE, если A<B, или FALSE - в противном случае.

Для ввода в разделе CONST задаются соответствующие значения. При этом тип констант автоматически определяется по содержанию правой части.

Например, фрагмент программы:

```
CONST  A=35;  X= 9,79; F= - 8.003; P=234;
BEGIN
Y:= A*X+(F*A)/P;
.....
END;
```

Именованным константам A, X, F присваивает вещественный тип, а константе с именем P - целый.

Однако такой способ заданий значений исходных данных не всегда удобен, так как фактически позволяет производить вычисления только для одного набора параметров.

Изменять значения констант в программе нельзя.

Если в разделе переменных описываются переменные A, X, F, P, а в разделе операторов им присваиваются соответствующие значения:

```
var
  A,X,F,P,Y: real;
    begin
      A:=2.5; X:=7.3;  F:=-17.5; P:=544.8;
      Y:=A*X*X+F/P-X;
      ......
    end.
```

В этом случае возможности варьирования значениями параметров расширяются, так как в программе можно организовать их изменение, однако набор параметров будет статическим.

Ввод может быть только бесформатным. Можно вводить данные только вещественного целого, и символьного типов. Данные набираются на дисплее, при этом разделителем между числами служит пробел или ENTER. Разделитель между символами, между числом и символом не нужен.

2.1. Алгоритмизация задач

Процесс подготовки и решения задач на ЭВМ является пока достаточно сложным и трудоемким, требующим выполнения целого ряда этапов.

Такими этапами являются:

1. *Постановка задачи.*
2. *Математическая формулировка задачи.*
3. *Выбор численного метода решения.*
4. *Разработка алгоритма решения задачи.*
5. *Написание программы.*
6. *Ввод и отладка программы.*
7. *Ввод исходных данных (решение контрольного примера).*
8. *Решение однотипных задач.*

Данная последовательность характерна для решения каждой задачи. Однако в процессе подготовки задачи каждый этап может иметь более или менее выраженный характер. Выполнение этапов в процессе подготовки задачи носит характер последовательного приближения, так как уточнение задачи на последующем этапе приводит к необходимости возврата к предыдущему и повторному выполнению последующих этапов.

Рассмотрим подробнее выполнение работ на каждом этапе в процессе подготовки задачи к решению.

Постановка задачи. Определяет цель решения задачи, раскрывает ее содержание. Задача формулируется на уровне профессиональных понятий, должна быть корректной и понятной исполнителю (пользователю). Ошибка в постановке задачи, обнаруженная на последующих этапах, приведет к тому, что работа по подготовке задачи к решению должна начаться с самого начала.

Математическая формулировка задачи. Осуществляет формализацию задачи путем описания ее с помощью формул, определяет перечень исходных данных и получаемых результатов, начальные условия, точность вычисления. По существу разрабатывается математическая модель решаемой задачи.

Выбор численного метода решения. В ряде случаев одна и та же задача может быть решена с помощью различных численных методов. Выбор метода должен определяться многими факторами, основными из которых являются точность результатов решения, время решения на ЭВМ и объем оперативной памяти. В каждом конкретном случае в качестве критерия для выбора численного метода принимают какой-либо из указанных критериев или некоторый интегральный критерий.

В простых задачах данный этап может отсутствовать, так как сам численный метод определен математической формулировкой задачи. Например, вычисление площади треугольника по формуле Герона, корней квадратного уравнения и т.д.

Разработка алгоритма решения задачи. На данном этапе устанавливается необходимая логическая последовательность вычислений с учетом выбранного численного метода решения и других действий, с помощью которых будут получены результаты. *Алгоритм* - некоторая конечная последовательность предписаний (правил), определяющая процесс преобразования исходных и промежуточных данных в результат решения задачи. Схема алгоритма представляет собой последовательность блоков, предписывающих выполнение определенных действия, и связи между ними.

2.2. Виды и свойства алгоритма

При разработке алгоритма решения задачи математическая формулировка задачи является основой для определения последовательности действий, приводящих к получению искомого результата.

Разрабатываемый алгоритм должен обладать следующими свойствами:

массовостью, позволяющей решать не одну задачу, а целый класс задач;

детерминированностью, однозначно определяющей выполняемые действия (промежуточные и окончательные результаты разных пользователей будут одинаковыми при одинаковых исходных данных);

результативностью, позволяющей получить результат после конечного числа шагов.

Различают следующие типы алгоритмов:

- линейные;

- разветвляющиеся;

- циклические.

Алгоритм линейной структуры - алгоритм, в котором все действия выполняются последовательно друг за другом. Такой порядок выполнения действия называется естественным. На практике редко удается представить схему алгоритма решения задачи в виде линейной структуры, так как задачи содержат различные условия или требуют многократного повторения вычислений.

Алгоритм разветвляющейся структуры - алгоритм, в котором в зависимости от выполнения некоторого логического условия вычислительный процесс должен идти по одной или другой ветви. В общем случае количество ветвей в алгоритме разветвляющейся структуры может быть больше двух.

Алгоритм циклической структуры - алгоритм, содержащий многократно выполняемые участки вычислительного процесса, называемые циклами. Использование циклов позволяет существенно сократить схему алгоритма. Различают циклы с заданным и неизвестным числом повторений, характеризующиеся последовательным приближением к исходному значению с заданной точностью.

Алгоритм со структурой вложенных циклов - алгоритм, содержащий цикл, внутри которого размещены один или несколько других циклов.

Цикл, охватывающий внутренние циклы, называется внешним. Правила организации как внешнего, так и внутренних циклов те же, как и для обычного цикла. Параметры этих циклов изменяются не одновременно, т.е. при одном значении параметра внешнего цикла параметр внутреннего цикла принимает по очереди все значения.

Существует 3 способа представления алгоритмов:

а) словесный;

б) операторный;

в) графический.

Графическая запись алгоритма должна выполняться в соответствии со стандартами.

Схема алгоритма представляет собой последовательность блоков, предписывающих выполнение определенных действий, и связи между ними.

Выделение составных частей алгоритма должно определяться внутренней логикой процесса вычислений.

Схема алгоритма может выполняться с разной степенью детализации. Схема, в которой определены ввод и вывод информации и учитываются особенности языка программирования, называется *схемой программы*.

Написание программы осуществляется по разработанному алгоритму с помощью языка программирования. Программа представляет собой последовательность операторов языка, записанную в соответствии со схемой программы.

Ввод программы и исходных данных выполняется с помощью клавиатуры.

Отладка программ. представляет собой процесс обнаружения и устранения синтаксических и логических ошибок.

Синтаксические ошибки, связанные с неправильной записью конструкции языка, выявляются самой ЭВМ. *Логические ошибки* появляются в результате нарушения последовательности вычислений и отсутствия необходимых данных для ЭВМ. Для выявления логических ошибок используют контрольные вычисления, выполненные другими средствами и методами.

После отладки программы и проверки ее на тестовых данных дополнительные операторы, введенные для отладки, исключаются из программы.

Рассмотрим пример составления алгоритма.

Задача. Определить площадь треугольника по длине сторон.

1. *Математическая формулировка задачи*. Для определения площади треугольника воспользуемся формулой Герона

$$s = \sqrt{p*(p-a)*(p-b)*(p-c)}\,,$$

где a, b, c - длины сторон;
p=(a+b+c)/2 - полупериметр треугольника,

2. *Алгоритм решения задачи.*

- Ввод исходных данных a ,b, c.
- Вычисление p.
- Вычисление s.
- Вывод результата s.

3.*Блок-схема программы.*

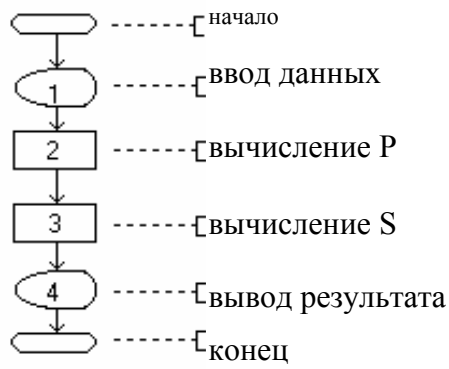


Рис. 2.2. Блок-схема примера.

В приведенной программе сначала вычисляются значения полупериметра p, а затем значение s, при вычислении которого используется p. Это позволяет избежать повторения вычисления одной и той же величины, а следовательно, уменьшить время вычисления задачи. На блок-схеме блоки расположены в той последовательности, в которой они должны выполняться. Любая перестановка блоков приведет к невозможности решения задачи.

4. Определение переменных участвующих в решении задачи.
Исходные данные: a, b, c - стороны треугольника, см (REAL);
Промежуточные данные:
p- полупериметр треугольника (REAL);
Выходные данные:
s - площадь треугольника (REAL).

5. Листинг программы:

```
PROGRAM PR1;
  VAR
    a, b, c, p, s: REAL;
  BEGIN
    WRITE('Введите  a,b,c в см');
    READLN(a,b,c);
    p:=(a+b+c)/2;
    s:=SQRT(p*(p-a)*(p-b)*(p-c));
    WRITELN (' Площадь  S= ', s:8:3,'  кв. см');
  END.
```

2.3. Стандартные функции

При составлении программы часто используются стандартные функции. Рассмотрим их использование.

Пример1. Даны числа **x** и **a**. Определить стандартные функции.

```
PROGRAM PR2;
  VAR
    X,A,Y1,Y2,Y3,Y4,Y5,Y6:REAL;
BEGIN
     WRITE(‘ Введите X и A’):
     READLN(X,A);
     Y1:=COS(X);
     Y2:=SIN(X);
     Y3:=Y2/Y1;
     Y4:=EXP(X);
     Y5:=LN(X);
     Y6:=EXP(X*LN(A));
WRITELN(‘ Ответ cos(x) =  ‘,Y1:8:3,’  радиан’);
WRITELN(‘ Ответ sin(x) =  ‘,Y2:8:3,’  радиан’);
WRITELN(‘ Ответ tg(x) =  ‘,Y3:8:3,’  радиан’);
WRITELN(‘ Ответ e^x =  ‘,Y4:8:3);
WRITELN(‘ Ответ Ln(x) =  ‘, Y5:8:3);
WRITELN(‘ Ответ a^x  =  ‘,Y6:8:3);
END.
```

Пример2. Дано: **a** и **b -** два неотрицательных числа и **b≠0.** Определить частное и остаток, возникающий при делении **a** на **b** с остатком.

17 div 3 = 5,
8 div 2 = 4,
1 div 5 = 0,
17 mod 3 = 2,
8 mod 2 = 0,
1 mod 5 = 1.

Пример 3. Дано действительное число A. Определить - число A четное или нечетное.

Воспользуемся стандартной функцией ODD, возвращающей значение логического типа TRUE или FALSE.

```
PROGRAM PR16;
  VAR
   a: INTEGER;
   c:boolean;
     BEGIN
      WRITE(‘Введите  число A ‘);
       READLN(a);
         c:=odd(a);
           WRITELN(c);
     END.
```

2.4. Трассировка программы

Отображение (на бумаге или на экране) того, что происходит при выполнении каждого оператора, называется трассировкой программы.

Трассировку программы удобно выполнять в таблице трассировки.

Структура таблицы:

- в первой колонке записываются номера строк, соответствующие операторам, и именно в том порядке , в каком они выполняются;
- во второй колонке записываются входы и выходы в различные части программы, пометки о выполняемых операторах. Эти сведения называются информацией об управлении.
- каждая их остальных колонок соответствует переменной программы и используется для прослеживания значений этой переменной.

Программа на Паскале, которая вычисляет сумму двух целых чисел и выводит ее на экран.

```
1. Program summa;
2.  var a,b,c: integer;
3.   begin
4.    read (a,b);
5.     c:=a+b;
6.    write (c );
7.   end.
```

Нумерация строк программы приведена только для трассировки.

Таблица трассировки программы summa.
Входные данные a=4, b=17.

Строка	Ход выполнения	a	b	c
3	вход в summa	?	?	?
4		4	17	
5				21
6	вывод: 21			
7	выход из summa			

При входе в программу - строка 3 (begin) переменные уже созданы, но еще не приобрели своих значений - еще *не определены*, этот факт отражается в таблице трассировки вопросительными знаками в соответствующих колонках. Выполнение оператора ввода read (a,b) в строке 4 привело к тому, что переменные a и b приобрели соответственно значения 4 и 17, а выполнение оператора присваивания в строке 5 - к присваиванию переменной c значения 21. Оператор в строке 6 приводит к выводу на экран числа 21, а достижение строки 7 означает выход из программы.

2.5. Построение (разработка) программ

Для того чтобы начать писать программу, необходимо знать:

- назначение программы - точная, подробная формулировка задачи;
- какую информацию программа будет получать в качестве входных данных;
- какую информацию программа должна вырабатывать в качестве выходных данных (результата).

Последние два пункта описывают и приводят примеры входных и выходных данных.

Процесс разработки программы состоит в преобразовании спецификации в программу путем пошаговой детализации, на каждом этапе которой определенная часть программы приобретает все более развернутый вид, и называется методом пошагового уточнения (нисходящее проектирование, метод разработки сверху-вниз).

В общем, процесс начинается с выделения наиболее общих шагов (составление общей схемы программы). В частности, общую схему любой программы можно представить в виде трех последовательно выполняемых шагов (задач):

1. Ввод входных данных.
2. Решение поставленной задачи.
3. Вывод результатов.

Далее достаточно сложные шаги, решение которых не очевидно, детализируются - разбиваются на более мелкие шаги. В ходе детализации по мере необходимости вводятся в употребление новые переменные. Процесс продолжается до тех пор, пока каждый из выделенных шагов (блоков) программы не окажется настолько простым, что его реализация на языке программирования уже не вызывает трудностей.

Процесс разработки программы методом пошаговой детализации оформляем в виде таблицы, учитывая следующие правила:

- каждый раздел таблицы соответствует одному из этапов детализации;
- если какое-либо предложение или выражение можно сразу записать на Паскале, оно так и записывается, без предварительной формулировки;
- если вводятся в употребление новые переменные, они перечисляются в графе “Примечания”.

Пример разработки программы.

Задача: Дан радиус окружности. Найти ее длину.

Алгоритм решения

1. Дан радиус окружности. Составить программу для вычисления длины окружности по формуле $L=2\pi r$.
2. Входные данные: вещественное число, радиус окружности.
3. Выходные данные: вещественное число, длина окружности.

Таблица разработки.

Шаги разработки	Примечания
work1 begin ввод входных данных вычисление значения выражения вывод результата end	
ввод входных данных read (r)	переменная r (real)
вычисление значения выражения L:=2*3.1415*r;	переменная L (real)
вывод результата write (L)	

Текст программы

```
1. Program work1;
2.  var r,L: real;
3.   begin
4.    read ( r );
5.     L:=2*3.1415*r;
6.    write (L)
7.   end.
```

Таблица трассировки

Входные данные: 5

Строка	Ход выполнения	r	L
3	вход в work1	?	?
4		5	
5			31.415

6	вывод: 31.415		
7	выход из work1		

Результат: 31,415

2.6. Управляющие конструкции языка. Условный оператор

Условный оператор используется в тех случаях, когда вычисления могут пойти по различным путям, в зависимости от выполнения или невыполнения определенных условий, причем выбор делается во время выполнения программы.

В Паскале это реализуется специальными управляющими операторами или операторными структурами, которые называются операторами перехода.

Схему алгоритма разветвленной структуры характеризует наличие блока «решение», который имеет два выхода, помеченные словами «да» и «нет». Этот блок называется также логическим блоком. В этом блоке осуществляется проверка выполнения некоторого логического условия. Если условие «истинно», то вычислительный процесс идет по выходу «да», в противном случае - по выходу «нет».

Различаются три типа разветвляющихся алгоритмов, блок-схемы которых приведены на рис.2. 3.

Ветвление, представленное на рис 2.3.а, называется обходом, так как оператор P1, записанный в арифметическом блоке, не выполняется, если условие S1 ложно. При реализации вычислительного процесса арифметический блок будет обойден и направление вычислений пойдет по ветви «нет».

Выбор из двух возможностей, или альтернатива, представлен на рис. 2.3.б. Если проверяемое условие S1 будет истинным, выполнится оператор P1, в противном случае выполнится оператор P2. Отметим, что алгоритм обхода является частным случаем альтернативы.

Выбор из множества возможностей представлен на рис.2.3.в. Здесь SN представляют собой условия выбора. По значению которых выполняется один из соответствующих операторов P1, P2, ...PN.

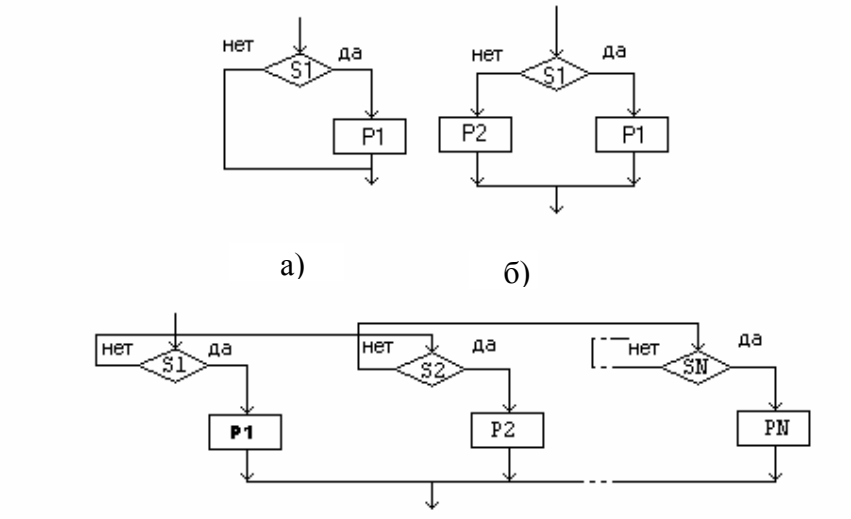


Рис.2. 3. Блок-схемы разветвляющихся процессов

Альтернатива или полная форма условного оператора имеет вид

IF < логическое выражение > THEN оператор P1

ELSE оператор P2;

Если логическое выражение истинно (true), тогда выполняется оператор P1, иначе (если логическое выражение ложно (false) - оператор P2. В качестве операторов P1 и P2 могут быть отдельные операторы. Это может быть: - *комментарий*, *вычисление, присваивание*. Если не-

обходимо использовать группу операторов, то они объединяются при помощи операторных скобок **BEGIN** и **END.**

```
IF < логическое выражение >
 THEN
    BEGIN
    оператор P11;
    оператор P12;
    .....................
    оператор P1i
    END
ELSE
    BEGIN
    оператор P21;
    оператор P21;
    .....................
    оператор P2i
    END
```

Условие, управляющее разветвлением вычислений, не обязательно должно иметь форму операции отношения. Оно может принимать вид любого логического выражения, в частности, логической переменной.

Правила написания программы позволяют записывать ее в свободной форме. Однако для удобства восприятия программы, особенно большой и сильно разветвленной, рекомендуется слово ELSE писать под тем словом IF, которому оно относится.

Обход или краткая форма условного оператора имеет вид

IF < логическое выражение > THEN оператор P1;

Если логическое выражение истинно (true), тогда выполняется оператор P1, иначе (если логическое выражение ложно (false) - выполняется оператор, расположенный в программе после условного оператора IF.

Краткой формой условного оператора нужно пользоваться осторожно, так как может нарушиться вся структура при вложенных условных операторах. Вместо краткой формы рекомендуется использовать всегда полную форму, но после слова ELSE ничего не ставить (т.е. пустой оператор).

Пример 4. Определить, число 35 кратно ли 5?

Рассмотрим первый случай. Число 35 кратно 5, если остаток от деления нацело равен 0.

Фрагмент программы будет выглядеть следующим образом:

```
. . .
   Write(‘ Введите число A ‘);
   Readln(A);
   Write(‘ Введите число B ‘);
   Readln(B);
      Y:= A mod B;
      IF Y=0  THEN Writeln(‘ Число ‘, A,’  кратно B ‘)
                ELSE  Writeln(‘ Число ‘, A,’ не кратно B ‘);
```

Рассмотрим второй способ:

```
   . . .
   Write(‘ Введите число A ‘);
   Readln(A);
   Write(‘ Введите число B ‘);
   Readln(B);
     Y:= A - (a div b) * b;
```

```
IF Y=0 THEN Writeln(‘ Число ‘, A,’  кратно B ‘)
        ELSE  Writeln(‘ Число ‘, A,’ не кратно B ‘);
. . .
```

Пример 5. Определить, число A кратно ли B и C?.

Фрагмент программы будет такой:

```
. . .
Write(‘ Введите число A ‘); Readln(A);
Write(‘ Введите число B ‘); Readln(B);
Write(‘ Введите число C ‘); Readln(C);
  Y1:= (a mod b);
  Y2:= (a mod c);
      IF (Y1=0) and (Y2=0) THEN Writeln(‘Число ‘, A,’ кратно’, B, ‘ и кратно ‘,C)
                           ELSE Writeln(‘Число ‘, A,’ не кратно’, B,’ и ‘,C);
. . .
```

Пример 6. Даны два числа **A** и **B.** Определить максимальное число.

```
PROGRAM PR6;
 VAR
  A,B: REAL;
   BEGIN
    WRITE(‘Введите число A’);
    READLN(A);
    WRITE(‘Введите число B’);
    READLN(B);
IF A>B THEN WRITELN (‘ Число A максимальное’);
  ELSE WRITELN (‘ Число B максимальное’);
     END.
```

Пример 7. Дано действительное число A. Определить - число A четное или нечетное.

Воспользуемся стандартной функцией ODD, возвращающей значение логического типа TRUE или FALSE.

```
PROGRAM PR7;
  VAR
  X: INTEGER;
  c:boolean;
    BEGIN
     WRITE(‘Введите  число A ‘);
      READLN(a);
        c:=odd(x);
          WRITELN(C);
    END.
```

Пример 8.

Определить, попадает ли точка A с координатами X и Y внутрь кольца, определенного радиусами R и R2. Центр круга совпадает с началом координат (рис. 2.4).

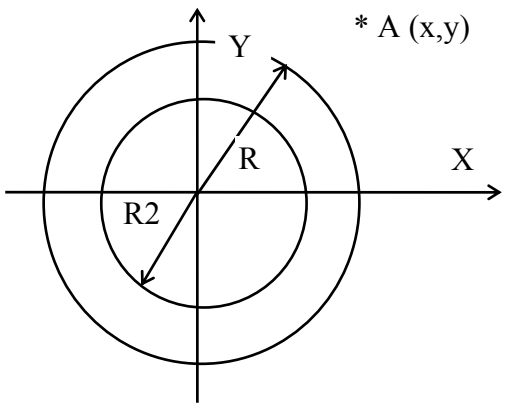


Рис. 2.4

Решение. Определим расстояние от точки А до начала координат

d=$\sqrt{x^2 + y^2}$.

Очевидно, что точка А будет находится внутри кольца, если d будет меньше R2 и больше R.

R<= d < =R2

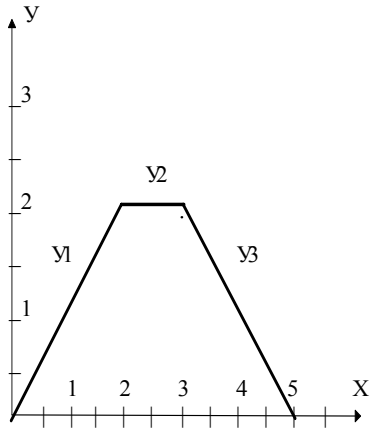
```
PROGRAM PR8;
 VAR
  X,Y,R1,R2: REAL;
BEGIN
  WRITE('Введите значения R и R2');
  READLN(R,R2);
  WRITE('Введите значения X и Y');
  READLN(X,Y);
    d:=sqrt(x*x+y*y);
     IF (d>= R) and (d<= R2)
      THEN WRITELN(' Точка попадает ')
       ELSE WRITELN(' Точка не попадает ');
  END.
```

Пример 9.

Вычислить функцию y=f (x). Представленную на рис. 2.5.

Комментарий: если удовлетворено условие X<2, то Y получит значение, равное X, и это значение затем будет выведено. Если условие X<2 не удовлетворяется, то значение Y будет определено выполнением условного оператора

```
if x<3 then y:=2
        else y:= -x+5
```



Для создания программы определим уравнения для У1,У2.У3.

У1=X;
У2=2;
У3= - X+5.

Рис. 2.5

```
PROGRAM PR9;
 VAR
  X,Y: REAL;
BEGIN
 WRITE(‘ Введите X = ‘);
 READLN(X);
  IF X< 2 THEN Y:=X
          ELSE
  IF X<3  THEN Y:=2
          ELSE  Y:= -X+5;
 WRITE(‘ Ответ Y = ‘,Y:8:3)
END.
```

Пример 10.

Определить, принадлежит ли точка с координатами X,Y прямоугольнику с координатами X_1,X_2,X_3,X_5 и Y_1,Y_2,Y_3,Y_5. (рис. 2.6).

Решение. Точка принадлежит прямоугольнику, если координаты имеют значение ($X >= X_1$ и $X<=X_5$) и ($Y >= Y_1$ и $Y<= Y_5$)

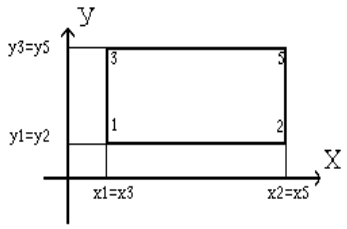


Рис. 2.6

Так как при проекции на ось X координат прямоугольника имеем $\mathbf{X_1 = X_3}$, $\mathbf{X_2 = X_5}$ и при проекции на ось Y имеем $\mathbf{Y_1 = Y_2}$ **и** $\mathbf{Y_3 = Y_5}$. Поэтому достаточно ввести координаты $\mathbf{X_1}$ и $\mathbf{X_5}$, $\mathbf{Y_1}$ и $\mathbf{Y_5}$

```
 PROGRAM PR10;
VAR
  X,Y,X1,X5,Y1,Y5:REAL;
 BEGIN
  WRITE(‘ Координаты точки A  (X,Y) ‘);
  READLN(X,Y);
  WRITE(‘Координаты прямоугольника X1,X5,Y1,Y5’);
   READLN(X1,X5,Y1,Y5);
   IF (X>=X1) AND (X<=X5) AND (Y>=Y1) AND (Y<=Y5)
     THEN WRITELN(‘ Точка принадлежит прямоугольнику’)
     ELSE WRITELN(‘ Точка не принадлежит прямоугольнику’);
 END.
```

Пример 11.

Пусть значение Y зависит от значения X, график зависимости приведен на рис. 2.7.

Решение.

Для того чтобы определить значения функции Y(x) необходимо разделить ее на части Y1, Y2, Y3. Для этого воспользуемся известными формулами, которые описывают уравнение данного типа.

Формула для определения уравнений типа Y1 и Y3 выглядит следующим образом:

$$\frac{Y - Y1}{X - X1} = \frac{Y2 - Y1}{X2 - X1},$$

где X1, X2, Y1, Y2 - координаты начала и конца отрезка.
Формула для определения уравнения типа Y2 выглядит следующим образом:

$Y=(x \pm a)^2 \pm в,$

где **а** и **в** координаты экстремума данной кривой.
После соответствующих преобразований имеем:
Y1 = 2.5 X+ 15; Y2 = $(X+0.5)^2$ + 1.75; Y3 = 7 - X.

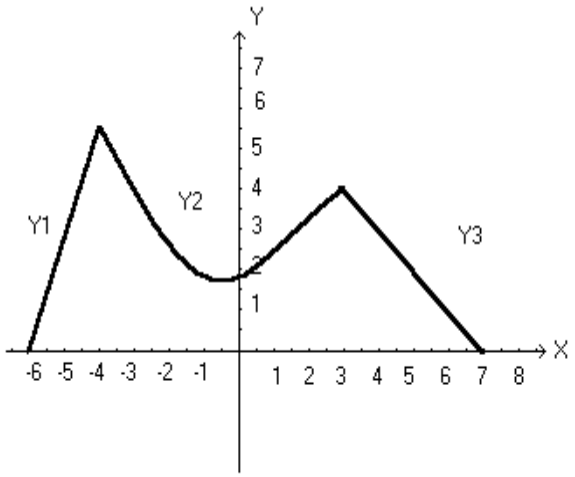


Рис. 2.7.

программа (вариант а):

```
PROGRAM PR11a;
   VAR
     X,Y,Y1,Y2,Y3:REAL;
       BEGIN
         WRITE (‘Введите X = ‘);
         READLN(X);
           Y1:=0.4X+15;
           Y2:=SQR(X+0.5)+1/75;
           Y3:= 7-X;
             IF (X>=-6) AND (X<=-4) THEN Y:=Y1;
              IF (X>-4) AND (X<=3) THEN Y:=Y2;
               IF (X>3) AND (X<=7) THEN Y:=Y3;
         WRITELN(‘ Ответ Y = ‘, Y:8:3);
       END.
```

Программа (вариант б):

```
  PROGRAM PR11b;
    VAR
      X,Y,Y1,Y2,Y3:REAL;
        BEGIN
          WRITE (‘Введите X = ‘);
          READLN(X);
            Y1:=2/5*X+15;
            Y2:=SQR(X+0.5)+1/75;
```

```
    Y3:= 7-X;
     IF (X>=-6) AND (X>=-4) THEN Y:=Y1
                                ELSE
        IF (X>-4) AND (X<=3) THEN Y:=Y2
                                ELSE
            IF (X>3) AND (X<=7) THEN Y:=Y3;
  WRITELN(‘ Ответ Y = ‘, Y:8:3);
END.
```

2.7. Оператор выбора CASE

Операторная запись алгоритма выбора используется в тех случаях, когда в зависимости от значения какого-либо выражения необходимо выполнить один или несколько последовательных операторов. Оператор выбора относится к сложным и имеет следующую форму записи:

CASE <выражение> OF
константа 1: оператор 1;
константа 2: оператор 2;
. . . . .
константа n: оператор n;
END

Здесь CASE (в случае), OF (из), END (конец) - служебные слова.

Оператор выбора действует следующим образом. Если значение выражения равно одной из констант, то выполняется соответствующий ей оператор. Затем управление передается за пределы оператора выбора.

Если значение выражения не совпадает ни с одной константой, то управление передается за пределы группы.

Выражение может быть любым стандартным типом, кроме действительного (REAL). В соответствии с этим и константа не может быть действительного типа. Тип константы должен совпадать с типов выражения. **Описание констант не требуется.**

Пример 12. Ввести номер недели и вывести соответствующий ему день недели на русском и английском языках.

```
PROGRAM PR12;
  VAR
    N: INTEGER;
      BEGIN
        WRITE(‘Введите номер дня недели ‘);
        READLN(N);
           CASE N OF
1: WRITELN (‘ Понедельник - MONDAY ‘);
2:WRITELN (‘ Вторник - TUESDAY‘);
3:WRITELN (‘ Среда - WEDNESDAY‘);
4: WRITELN (‘ Четверг - THURSDAY‘);
5:WRITELN (‘ Пятница - FRIDAY‘);
6:WRITELN (‘ Суббота - SATURDAY‘);
7:WRITELN (‘ Воскресенье - SUNDAY‘);
           END
```

END.

Примечание: После констант может быть представлено только **одно** значение:
комментарий;
вычисление;
присваивание.

Если необходимы дополнительные действия, то необходимо использовать операторные скобки **BEGIN** и **END**.

В операторе выбора в качестве константы допускается использование списка констант, например:

```
CASE S OF
 '+', '-', '*', '/' :P:=1;
A', 'B ',    :P:=2;
          '_'      :P:=3;
END.
```

Переменная **S** должна быть определена в разделе описаний как символьная. Если значение S будет один из знаков **'+', '-', '*', '/'** , то переменная P получит значение **1**; Если значение S будет символ 'A', 'B ',то P получит значение **2**; Если значение S будет **'_'**, то переменная P получит значение **3**;

2.8. Константы

Тип константы (постоянной) определяется типом соответствующего значения константы. В определении каждому значению константы ставится соответствующее имя, которое используется в дальнейшем в программе и облегчает проведение изменений в программе. В этом случае пользователь вносит изменения только в определении констант. При этом отпадает необходимость поиска и изменения константы по всей программе, так как транслятор каждое имя изменяет на соответствующее значение. Если в программе не предполагается использование констант, то раздел определения констант не приводится.

Константы в программе могут быть заданы явно своим значением или обозначены именем. Если константа обозначена именем, то она записывается в разделе CONST. В этом случае CONST не изменяется в ходе программы. Описание начинается со слова CONST и имеет следующую форму записи:

CONST <имя константы> = значение;

Пример: CONST n=10;

Допускается описывать несколько CONST. Каждое описание заканчивается символом точки с запятой (;).

Пример:

```
CONST              типы констант
  num = 23;     { константа целого типа - integer,}
  b = 1.8E-3; { константа действительного типа - real, }
  pi = 3.14;   { константа действительного типа - real, }
  sim = ' R ';  { константа символьного типа - char, }
  L = true;   { константа логического типа - boolean, }
```

Константы целого и действительного типа являются числовыми. Они могут иметь положительный или отрицательный знак и записываются числами или именем соответствующего типа.

Константы логического типа имеют одно из двух значений:

TRUE или FALSE.

Константы символьного типа записываются литерами, заключенными в кавычки. Символы апострофа записываются двойными кавычками

' B ', ' C ', '_', ''''.

Текстовые константы (строки) записываются последовательностью символов (текстом), заключенные в кавычки:

' x= ', ' параметр ', ' сумма S =',

Пример 13. Определить объем шара V с радиусом R.

Решение. Объем шара определим как

$V= 4/3*R^3*\pi$.

```
PROGRAM PR13;
 CONST PI=3.14;
  VAR
   V,R: REAL;
    BEGIN
      WRITE('Введите R');
       READLN(R);
          V:=4*PI*R*R*R/3;
WRITELN (' Объем шара V = ', V:8:3);
    END.
```

2.9. Метки и оператор перехода

В Паскале принят естественный порядок выполнения программы - все операторы выполняются последовательно один за другим в том порядке, как они записаны. Однако в практике возникает необходимость нарушения последовательности выполнения операторов.

Для этого предназначен оператор перехода, который имеет следующую форму записи:

GOTO метка,

где - метка- целое число без знака в диапазоне 1-9999. Это число записывается перед помеченным оператором и отделяется от него двоеточием:

```
GOTO 32
          10: a:=2;
          ....
          32: y:=x/z;
```

Здесь после оператора GOTO 32 выполняется оператор с меткой 32. Следует отметить, что оператор, следующий за оператором перехода, также должен быть помечен. Иначе все операторы в программе между оператором GOTO и оператором с меткой 32 будут лишними.

Допускаются буквенные и буквенно-цифровые обозначения меток. Все метки должны быть объявлены в разделе описания LABEL.

Объявление имеет вид:

```
LABEL 32;
```

Допускается объявлять список меток:

```
LABEL  1,2,A1,FF102;
```

Пример 14. Определить корни уравнения

$AX^2 + BX + C = 0$,

где A,B,C - действительные числа.

```
PROGRAM PR14;
 LABEL 1,2,3,4;
```

```
var
 a,b,c,d,K,x1,x2:real;
  begin
   write('Введите коэффициенты A,B,C');
   readln (a,b,c);
   d:=b*b-4*a*c;
     if d<0 then goto 1;
       if d=0 then goto 2;
         x1:=(-b + sqrt(d))/2;
         x2:=(-b - sqrt(d))/2;
            goto 3;
   1: Writeln('Действительных коней нет');
            goto 4;
   2: x1:=(-b)/(2*a);
      x2:=x1;
        goto 3;
   3: writeln('Ответ: x1 = ',x1:8:3,' x2 = ',x2:8:3);
   4:
 end.
```

2.10. Организация циклических процессов

При решении многих задач вычислительный процесс имеет циклический характер. Это означает, что часть операторов многократно выполняются при различных значениях переменных.

Различают арифметические и итерационные вложенные циклы. Они делятся на два типа: циклы на достижение заданной точности (итерационные) и циклы с известным числом повторений (арифметические).

Циклический процесс называется итерационным, если заранее неизвестно количество повторений цикла, а конец вычислений определяется при достижении некоторой заранее заданной точности вычисления.

В языке ПАСКАЛЬ имеется три вида операторов цикла:
оператор цикла с предварительным условием (предусловием);
оператор цикла с последующим условием (постусловием);
оператор цикла с параметром.

Для всех операторов цикла характерна следующая особенность:
повторяющиеся вычисления записываются всего один раз;
вход в цикл возможен только через его начало;
переменные цикла должны быть определены до входа в циклическую часть;
необходимо предусмотреть выход из цикла: или по естественному его окончанию, или по оператору перехода.

Если этого не предусмотреть, то циклические вычисления будут повторяться бесконечно. В этом случае говорят, что произошло “зацикливание” выполнения программы.

2.11. Оператор цикла с параметром

Оператор цикла с параметром используется в тех случаях, когда заранее известно, сколько раз должна повторятся циклическая часть программы или можно вычислить количество шагов цикла.
Оператор цикла имеет вид:

FOR I:=m1 TO m2 DO

```
BEGIN
   ТЕЛО
   ЦИКЛА
END.
```

Или

```
FOR  I:= m1 DOWNTO m2 DO
  BEGIN
     ТЕЛО
     ЦИКЛА
  END.
```

Здесь - FOR (для), TO (до), DO (выполнить), DOWNTO (вниз до) - служебные слова;

I - параметр цикла;

m1, m2 - начальное и конечное значение параметра цикла.

Тело цикла выполняется повторно для каждого значения параметра цикла I от его начального значения m1 до конечного значения m2 включительно.

В качестве параметров цикла может быть только переменная, в качестве m1 и m2 могут быть выражения, за исключением действительного типа (REAL).

Чаще всего параметр цикла I используется как переменная целого типа, а шаг его изменения равен **+1** или **- 1**.

Если значение параметра цикла возрастает, то шаг его изменения **+1.** Если значение параметров цикла уменьшается, то шаг его изменения **- 1** и в операторе цикла FOR вместо служебного слова TO записывается служебное слово DOWNTO.

Задать шаг, отличный от 1 или -1, нельзя !

Выполнение операторов цикла начинается с проверки условия I<= m2 для цикла TO (I>=m2 для цикла DOWNTO). Если оно не справедливо, то оператор циклической части программы не выполняется, а управление передается следующему оператору. Если же условие I<= m2 истинно, то выполняется оператор циклической части программы, а затем параметру цикла присваивается следующее значение I:=SUCC(I) (для цикла TO) или предыдущее значение I:=PRED(I) (для цикла DOWNTO). Далее весь процесс повторяется. Если параметр цикла целого типа, то это означает его увеличение (соответственно уменьшение) на единицу при каждом новом выполнении расположенного в цикле оператора.

Для оператора цикла с параметром существуют некоторые ограничения:

значения параметров цикла, начального и конечного значения параметра цикла изменять внутри цикла нельзя;

войти в цикл можно только через его начало, а выйти - либо при исчерпании значений параметров цикла, либо при выполнении операторов перехода по метке, расположенной в цикле.

Итак, оператор цикла с параметром позволяет осуществить последовательный перебор значений параметра в любом их двух направлений, но с приращением, равный единице соответствующего типа данных. В том случае, если шаг просмотра отличен от единицы или заранее не известно количество повторений тела цикла, необходимо применять один из двух других видов операторов цикла.

Рассмотрим использование операторов цикла с параметрами.

Пример 15. Пусть имеется фрагмент программы с переменными целого типа

```
FOR I:=1  TO  5  DO
   BEGIN
    A:=2*I;
     B:=2*I+1;
       WRITELN(A:3,’  ‘,B:3);
```

```
END;
```

Циклическая часть программы выполняется повторно пять раз, при этом параметр цикла I изменяет свое значение от 1 до 5. В результате выполнения программы переменные получают следующие значения:

```
I . . . . . 1   2   3   4   5
A . . . .  2   4   6   8  10
B  . . . . .3   5   7   9   11
```

Фрагмент программы с убыванием значений параметра цикла от 5 до 1 имеет следующий вид:

```
FOR I:=1  DOWNTO  5  DO
 BEGIN
   A:=2*I;
    B:=2*I+1;
     WRITELN(A:3,’  ‘,B:3);
 END;
```

В процессе выполнения программы переменные принимают следующие значения:

```
I . . . . . 5    4   3   2   1
A . . . . . 10   8   6   4   2
B  . . . . .11   9   7   5   3
```

Если циклическая часть программы содержит только один оператор, то операторные скобки BEGIN - END можно не ставить. В этом случае цикл с параметром записывается в следующем виде:

```
FOR I:= m1  TO  m2  DO
  оператор
```

Параметр цикла I не должен переопределяться внутри циклической части.

Если шаг изменения параметра цикла равен +1 и m1> m2, то циклическая часть не выполняется ни разу.

После естественного завершения цикла значение параметра цикла не определено. Это означает, *что при последнем выполнении циклической части значение I=m2, а после ухода за пределы цикла значение I теряется.*

2.12. Табуляция функций

Оператор FOR следует использовать во всех случаях, когда заранее известно число повторений или его можно подсчитать. Для вычисления значений функций при изменении аргумента с постоянным шагом h в определенном интервале (от начального $x_н$ до конечного $x_к$ значения) количество повторений n определяется по формуле

$$n = \left[\frac{x_к - x_н}{h} \right] + 1$$

Квадратные скобки указывают на то, что результат округляется до целой части путем отбрасывания дробной.

Пример 16. Составить программу для вычисления и вывода значений функции

$$y = \frac{x^2 - 2x + 2}{x - 1}$$

при изменении X от -4 до +4 с шагом 0.2.

Решение. Определим число повторений n из формулы

$$n = \left[\frac{4 - (-4)}{0.2} \right] + 1 = 41$$

```
PROGRAM PR16;
 VAR
  X,Y: REAL;
  I: INTEGER;
   BEGIN
     X:=-4;
      FOR I:=1 TO 41 DO
       BEGIN
         Y:=(X*X-2.0*X+2.0)/(X-1.0);
          X:=X+0.2;
            WRITE(X,Y)
       END
   END.
```

Для данной функции эту программу можно записать в общем виде для различных значений интервалов и величин изменения шага и в наглядном виде. При этом информация выводится в удобной форме.

```
PROGRAM PR16a;
 VAR
  XN,XK,HX,X,Y: REAL;
   I,N: INTEGER;
    BEGIN
WRITE(‘ Введите начальное значение X’);
READLN(XN);
WRITE(‘ Введите конечное значение X’);
READLN(XK);
WRITE(‘ Введите шаг изменения X’);
READLN(HX);
N:=TRUNC((XK-XH)/HX)+1;
WRITELN(‘ ----------------------------’);
WRITELN(‘*        XN   *      Y     *’);’
WRITELN(‘ ----------------------------’);
  FOR I:=1 TO N DO
   BEGIN
    Y:=(XN*XN-2.0*XN+2.0)/(XN-1.0);
WRITELN(‘*  ‘,XN:8:3,‘ * ‘,Y:8:3,’  *’);
  XN:=XN+HX;
   END
WRITELN(‘ ----------------------------’);
  END.
```

Рассмотрим различные формы вывода окончательного результата расчета.

Пример 16б. Необходимо в конечной форме провести линию после каждого шага расчета.

Фрагмент программы

```
. . . . . .
WRITELN(‘ ----------------------------’);
WRITELN(‘*        XN   *      Y     *’);’
WRITELN(‘ ----------------------------’);
```

```
 FOR I:=1 TO N DO
  BEGIN
   Y:=(XN*XN-2.0*XN+2.0)/(XN-1.0);
    WRITELN('*  ',XN:8:3,' * ',Y:8:3,'  *');
    WRITELN(' ----------------------------');
   XN:=XN+HX;
  END
WRITELN(' ----------------------------');
END.
```

Пример 16в. В задаче, рассмотренной в примере 18, вывести только окончательный результат. Фрагмент программы

```
. . . . . .
WRITELN('*        XN   *      Y     *');'
  FOR I:=1 TO N DO
    BEGIN
     Y:=(XN*XN-2.0*XN+2.0)/(XN-1.0);
     XN:=XN+HX;
  END
 WRITELN('*  ',XN:8:3,' * ',Y:8:3,'  *');
END.
```

2.13. Вычисление суммы

Сумма вычисляется по рекуррентному выражению

S=S+Y,

где **S** - накапливаемая сумма;

Y - слагаемое.

По данному выражению каждое новое значение получается из предыдущего добавлением очередного слагаемого. Для первого слагаемого Y начальное значение суммы S = 0.

Пример 17. Составить программу для вычисления среднеарифметического N произвольных чисел.

Введем обозначения:

S- сумма всех чисел;

SR - среднеарифметическое N чисел;

A - значение I - го числа;

N - количество чисел.

```
PROGRAM PR17;
  VAR
    I,N:INTEGER;
      S,SR,A:REAL:
   BEGIN
     WRITE(' Введите N');
     READLN(N);
       S:=0;
          FOR I:=1 TO N DO
           BEGIN
            WRITE(' Введите ',I,' число');
            READLN(A);
          S:=S+A:
           END;
                  SR:=S/N;
```

```
            WRITELN('Ответ: SR = ',SR:8:3);
END.
```

2.14. Вычисление произведения

Произведение вычисляется по рекуррентному выражению

P=P*Y,

где **P** - промежуточные произведения;

Y - сомножители.

По данному выражению каждое новое значение получается из предыдущего умножением очередного сомножителя. Для первого сомножителя Y начальное значение произведения P = 1.

Пример 18. Составить программу для вычисления произведения N чисел.

Введем обозначения:

P - произведение всех чисел;

A - значение I - го числа;

N - количество чисел.

```
PROGRAM PR18;
  VAR
    I,N:INTEGER;
     P,A:REAL:
         BEGIN
          WRITE(' Введите N');
          READLN(N);
         P:=1;
          FOR I:=1 TO N DO
BEGIN
  WRITE(' Введите ',I,' число');
  READLN(A);
   P:=P*A:
END;
   WRITELN('Ответ: P = ',P:8:3);
 READLN;
END.
```

2.15. Определение факториала

Математически определение факториала запишем как **P=N!** При этом используется прием накопления произведения.

Рассмотрим, чему равняется 3!. P=3*2*1=6.

Для вычисления факториала воспользуемся переменной типа LONGINT вместо INTEGER.

```
PROGRAM PR19;
  VAR
   I,N:INTEGER;
   P:LONGINT;
     BEGIN
       WRITE(' Введите ',N,' число');
       READLN(N);
         P:=1;
```

```
        FOR I:=1 TO N DO
          BEGIN
            P:=P*I:
           END;
        WRITELN('Ответ: P = ',P:8:3);
        READLN;
    END.
```

Рассмотрим пример следующего типа.

Пример 20. Вычислить $y = \sum_{1}^{N} (-1)^i * a^i$.

Решение. В данном случае нам необходимо учесть знакочередование функции.

```
PROGRAM PR20;
  VAR
   I,N,P:INTEGER;
  A.Y: REAL;
   BEGIN
     WRITE(' Введите N ');
     READLN(N);
       WRITE(' Введите число  A ');
       READLN(A);
        P:=1;
        Y:=0;
          FOR I:=1 TO N DO
            BEGIN
              P:=P*(-1):
              Y:=Y+(P)*(EXP(I*LN(A)));
              END;
        WRITELN('Ответ: Y = ',Y:8:3);
    END.
```

Пример 21. Определение наименьшего значения по выражению

K^3 *sin (n+$\frac{k}{n}$), где к= 1,2, ..N.

```
PROGRAM PR21;
 VAR
  ELEM, MIN, REAL;
  K,N:INTEGER;
BEGIN
   WRITE('Введите N ');
   READLN(N);
     MIN:= SIN(N+1/N);
       FOR K:= 2 TO N DO
         BEGIN
           ELEM:=K*K*K*SIN(N+K/N);
             IF ELEM < MIN THEN MIN:=ELEM
         END;
   WRITELN('MIN = ',MIN:8:5);
    END.
```

Примечание: при выполнении этой программы наименьшее число определяется за N шагов; после выполнения K-го шага значение переменной MIN равно наименьшему числу среди первых K чисел последовательности. Первый шаг состоит в том, что переменная MIN присваивается значению первого члена последовательности. Остальные N-1 шагов выполняются с помощью операторов цикла. Каждый следующий шаг выполняется с учетом значения полученного на предыдущем шаге.

2.16. Вложенные циклы

Если телом цикла является циклическая структура, то такие циклы называются *вложенными* или *сложными*. Цикл, содержащий в себе другой цикл, называют *внешним*. Цикл, содержащийся в теле другого цикла, называют *внутренним*.

Внутренний и внешний циклы могут быть любыми из трех рассматриваемых видов: циклами с параметром, циклами с предусловием, циклами с постусловием. Правила организации как внешнего, так и внутреннего циклов такие же как и для простого цикла каждого из этих видов. Однако при построении вложенных циклов необходимо соблюдать следующее дополнительное условие: *все операторы внутреннего цикла должны полностью лежать в теле внешнего цикла*.

Сложные циклы условно разбивают на *уровни вложенности*. Параметры циклов разных уровней изменяются не одновременно. Вначале все свои изменения изменит параметр цикла наивысшего уровня вложенности при фиксированных (начальных) значениях параметров циклов с меньшим уровнем. Затем изменяется на один шаг значения параметра цикла следующего уровня и снова полностью выполняется самый внутренний цикл и т.д. до тех пор, пока параметры циклов всех уровней не примут требуемые значения.

При этом, если в сложном цикле с глубиной вложения K число повторений циклов на каждом уровне равно N_0, N_1, ...,N_k соответственно, то общее количество повторений тела самого внутреннего цикла равно $N = N_0 * N_1 * ...* N_к$.

Рассмотрим примеры.

Пример 22. Вычислить **Y=** $\sum_{i=1}^{N}\sum_{j=1}^{M} a^{\frac{i}{j}}$ **.**

```
PROGRAM PR22;
  VAR
    I,J: INTEGER;
    A,Y: REAL;
     BEGIN
    WRITE(‘ Введите N, M);
    READLN(N,M);
    WRITE(‘ Введите A);
    READLN(A);
      Y:=0;
WRITELN(‘ I    J     Y’);
 FOR I:=1 TO N DO
    FOR J:=1 TO M DO
       BEGIN
        Y:= Y + EXP((I/J)*LN(A));
          WRITELN(I,’  ‘,J,’  ‘,Y:8:5);
       END;
  END.
```

Пример 23. Составить программу вычисления значений функции **Z=sin(x) +cos(y)** для **x=x_0(h_x)x_n** и **y=y_0(h_y)y_n.** Аргументы функции **x, y -** действительные числа. (Здесь имеем **x_0, y_0** - начальные значения **x** и **y**; **hx, h_y** - шаг изменения **x** и **y**; **x_n, y_n** - конечные значения x и y).

Для определения значений функций Z для всех различных пар (x и y) необходимо процесс вычислений организовать следующим образом. Вначале при фиксированном значении одного из аргументов, например при x= x_0 вычислить все значения Z для всех заданных y: y_0, $y_0 + h_y$, ... , y_n. Затем, изменив значение x на x+ h_x, вновь перейти к полному циклу изменения переменной y. Данные действия повторить для всех заданных x: x_0, $x_0 + h_x$, ... , x_n.

При реализации данного алгоритма нам необходима структура вложенных циклов: внешнего цикла - для изменения значения переменной x и внутреннего цикла - для изменения значений переменной y. Причем в данной задаче внешний и внутренний циклы можно поменять местами, при этом изменится только очередность изменения аргументов при вычислении функции.

```
PROGRAM PR23;
  VAR
   I,J,NX,NY:INTEGER;
   X,X0,HX,XN,Y,Y0,HY,YH,Z:REAL;
     BEGIN
      WRITE(‘ Введите  X0,HX,XN ‘);
       READLN(X0,HX,XN);
     WRITE(‘ Введите Y0,HY,YH’);
      READLN(Y0,HY,YH);
          NX:= TRUNC((XN-XO)/HX)+1;
          NY:= TRUNC((YN-YO)/YX)+1;
          X:=X0;
WRITELN (‘    X       Y        Z’);
    FOR I:=1 TO NX DO
      Y:=Y0;
        BEGIN
         FOR J:=1 TO NY DO
            BEGIN
              Z:=SIN(X)+COS(Y);
             WRITELN (X:5:3,’ ‘’, Y:5:3,’ ‘,Z:8:5);
             Y:=Y+Y0;
            END;
     X:=X+X0;
        END;
  END.
```

2.17. Оператор цикла с предусловием

Оператор цикла с предусловием имеет вид

WHILE **B** DO **A,**

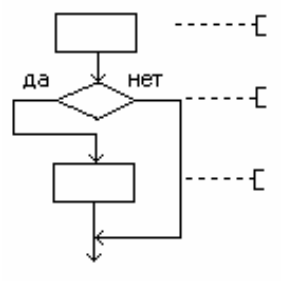
где WHILE (ПОКА), DO - служебные слова:

B - логическое выражение булевского типа;

A - простой или составной оператор.

Выполнение оператора начинается с вычисления значения B. Если оно имеет значение TRUE, то выполняется оператор A. Затем выполнение оператора цикла повторяется до тех пор, пока значения выражения B не станет равным FALSE. Как только получается значение FALSE, управление передается оператору, следующим за оператором цикла, а оператор A внутри цикла не повторяется.

Если выражение булевского типа было ложным при первом входе в цикл, то оператор А не выполняется ни разу. Очевидно, что один из операторов, находящийся внутри цикла, должен влиять на значение логического выражения, поскольку иначе цикл будет повторяться бесконечно. Функциональная схема итерационного процесса с предусловием представлена на рис. 2.8.



начальное присваивание

условие

тело цикла

Рис. 2.8. Функциональная схема итерационного процесса

Проиллюстрируем использование операторы цикла с предусловием.

Пример 24. Вывести на экран все четные числа до 10.

```
PROGRAM PR24;
  VAR
        K:INTEGER;

    BEGIN
      K:=0;
       WHILE K<=10 DO
        BEGIN
          K:=K+2;
            WRITE(K:3);
        END
    END.
```

В результате выводятся числа 2 4 6 8 10 12,

Рассмотрим этот же пример с другим выводом:

Пример 24а.

```
PROGRAM PR24;
  VAR
   K:INTEGER;
    BEGIN
      K:=0;
       WHILE K<=10 DO
          BEGIN
           WRITE(K:3);
            K:=K+2;
          END
    END.
```

В результате выводятся числа 0 2 4 6 8 10,

Пример 25. Пусть даны числа a и b (a> 1), и надо получить все члены бесконечной последовательности a, a^2, a^3,.. меньше числа b.

```
PROGRAM PR25;
  VAR
   A,B,C: REAL;
    BEGIN
```

```
    WRITE(‘ Введите A и B’);
    READLN(A,B);
      C:=A;
        WHILE C< B DO
          BEGIN
            WRITELN(‘C = ‘,C:8:5);
             C:=C*A;
          END
   END.
```

Пример 26. Вычислить $\sqrt{U}$ для данного $U \geq 0$.
Рассмотрим бесконечную последовательность x_1, x_2, x_3, ... образованную по следующему закону:

$$X_i = \frac{u+1}{2};$$

$$X_i = \frac{1}{2}\left(X_{i-1} + \frac{u}{X_{i-1}}\right), i = 2,3,\ldots$$

Оказывается, что члены этой последовательности с увеличением номера i все меньше и меньше отличаются от $\sqrt{U}$.

```
PROGRAM PR26;
 VAR
   U,X,E; REAL;
     BEGIN
      WRITE(‘ Введите U и E’);
       READLN(U,E);
         X:=(U+1)/2;
           WHILE ABS(SQR(X)-U) >=  E   DO
             X:=0.5*(X+U/X);
      WRITELN(‘X = ‘,X:5:3);
     END.
```

Пример 27. Напишем программу приближенного вычисления суммы

$$1+\frac{X}{1!}+\frac{X^2}{2!}+\cdots$$

с заданной точностью E (X<= 86).

По условию задачи считается, что нужное приближение получено, если вычислена сумма некоторых первых слагаемых, и очередное слагаемое оказалось по модулю меньше, чем данное малое положительное число E - и все последующие слагаемые уже не надо учитывать.

При анализе задачи видно, что каждое последующее слагаемое отличается от предыдущего на величину X/i.

```
PROGRAM PR27;
 VAR
  X,E,Y,S:REAL;
   I:INTEGER;
```

```
   BEGIN
    WRITE(‘ Введите X и E’);
    READLN(X,E);
  S:=0;
  Y:=1;
  I:=0;
   WHILE ABS(Y)>= E DO
    BEGIN
     S:=S+Y;
     I:=I+1;
    Y:=Y*X/I;
    END;
WRITELN(‘S = ‘,S:8:6);
   END.
```

Рассмотрим еще один пример на точность вычислений.
Пример 28. Вычислить с заданной точностью E

Y=$\sqrt{2+\sqrt{2+\sqrt{2+...}}}$ и определить N -вычислений.

В данном случае необходимо определить правила изменения приращения на каждом шаге вычисления:

на 1- ом шаге Y1=$\sqrt{2}$;

на 2- ом шаге Y2=$\sqrt{2+\sqrt{2}}$.

Разница δY = Y1-Y2 обеспечивает необходимую точность расчета.

```
PROGRAM PR28;
  VAR
   Y,Y1,Y2,Y3,X,E:REAL;
  I:INTEGER;
   BEGIN
     WRITE(' Введите точность вычисления E = ');
      READLN(E);
       Y1:=SQRT(2);
       Y2:=0;
       Y3:=Y1-Y2;
   I:=1;
WRITELN('I      Y1         Y        Y3');
  WHILE ABS(Y3)>=E DO
   BEGIN
    WRITELN(I,' ',Y1:8:7,' ',Y2:8:7,' ',Y3:8:7);
      Y1:= SQRT(2+Y1);
      Y2:= SQRT(2+Y2);
      Y3:=Y1-Y2;
        I:=1+I;
   END
   END.
```

Пример 29. Рассмотрим решение уравнения вида Y=A*X^2+B*X+C путем подставки.
Решение. A,B,C - коэффициенты уравнения;
X - начальное приближение;
E - точность вычисления;
X:=X+E/2 изменение (приращение) X.

```
PROGRAM PR29;
 VAR
  X,Y,E,A,B,C:REAL;
   I:INTEGER;
   BEGIN
    WRITE('X,E');
     READLN(X,E);
     WRITE('A');
        READLN(A);
        WRITE('B');
        READLN(B);
        WRITE('C');
        READLN(C);
         Y:=1;
          I:=1;
           WHILE I<=2  DO
            BEGIN
              WHILE ABS(Y)>= E DO
               BEGIN
                X:=X+E/2;
                Y:=A*X*X+B*X+C;
               END;
                WRITELN('Y = ',Y:8:7,' X',I:2,' = ',X:5:4);
                  I:=I+1;
                  X:=X+E/2;
                  Y:=A*X*X+B*X+C;
            END;
       END.
```

Пример 30. Составить программу вычисления значений функции

$$Y = \frac{5 - 20x}{Ln(10x^2 - 12x + 2.7)}$$

при изменении X от $X_н$ до $X_к$ с шагом dX.

При решении данной задачи возникает ситуация, когда попытка вычислить значение функции Y при X=0.3 - логарифм натуральный из отрицательного числа. В данном случае необходимо дополнительно произвести проверку, является ли аргумент логарифма положительным числом.

```
PROGRAM PR30;
 VAR
 XN,XK,DX,X,Y,R:REAL;
  BEGIN
   WRITE(' Введите XN,XK,DX ');
    READLN(XN,XK,DX);
     X:=XN;
WRITELN('   X          Y');
  WHILE   X<=XK  DO
    BEGIN
     R:=10*X*X-12*X+2.7;
      IF R>0 THEN
        BEGIN
         Y:=(5-20*X)/LN(R);
```

```
        WRITELN(X:8:5,'  ',Y:8:6);
       X:=X+DX;
      END
            ELSE
       WRITELN(' Функция не определена ');
  END
END.
```

2.18. Оператор цикла с постусловием

Оператор цикла с постусловием похож на оператор цикла с предусловием, но условие вычисляется и проверяется после выполнения операторов, составляющих тело цикла. Общий вид оператора цикла с постусловием:

```
REPEAT
 A1;
 A2;
... ;
 AN
UNTIL B
```

где A1;A2;...;AN - операторы тела цикла:
B - выражение булевского типа.

Оператор цикла с постусловием начинается с выполнения операторов внутри цикла. Затем вычисляется выражение B, и если получается истинное значение (TRUE), то осуществляется выход из цикла. Если же значение выражения ложно (FALSE), то выполнение операторов A1;A2;...;AN повторяется, а затем снова вычисляется выражение B.

В отличие от цикла с предусловием выход из цикла с постусловием осуществляется при истинности выражения B и в отличие от оператора WHILE в операторе REPEAT условие повторяется после каждой итерации; обеспечивается выполнение, по крайней мере, одного вычисления в цикле (когда значение логического выражения FALSE); тело цикла может содержать не один, а несколько операторов без записи их в составном операторе. Эта особенность записи оператора объясняется тем, что компилятор воспринимает ключевое слово REPEAT как REPEAT BEGIN, а UNTIL как UNTIL END. Функциональная схема итерационного процесса с постусловием представлена на рис. 2.9.

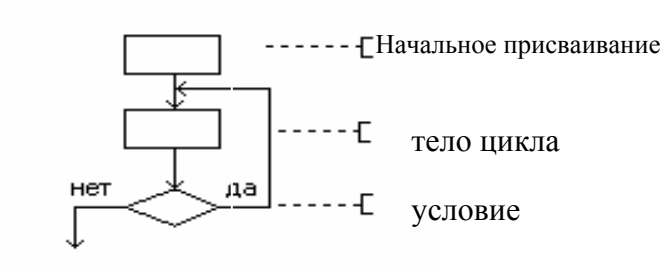


Рис. 2.9. Блок-схема итерационного процесса.

Рассмотрим пример 24 с использованием оператора цикла с постусловием:

```
PROGRAM PR34                PROGRAM PR 34A
 VAR                         VAR
  K:INTEGER                   K:INTEGER;
   BEGIN                       BEGIN
    K:=0;                       K:=0;
REPEAT                      REPEAT
```

```
 K:=K+2;                          K:=K+2;
  WRITELN(K:3);                    WRITE(K:3);
UNTIL K> 10                      UNTIL K>=10
   END.                             END.
```
Ответ будет такой Ответ будет такой
2 4 6 8 10 12 2 4 6 8 10

Пример 31. Составить программу для определения k, при котором функция x^k/k становится меньше *a*, где k=1, 2, 3, ...

Решение. Это типичный пример организации цикла с неизвестным числом повторений. Логическим выражением является выражение $x^k/k < a$. Начальное значение k=1. В теле цикла изменяется значение k оператором k+1.

```
PROGRAM PR31;
 VAR
  X,A,P:REAL;
   K:INTEGER;
    BEGIN
WRITE(‘ Введите X и A’);
READLN(X,A);
  K:=0;
  P:=1;
   REPEAT
    K:=K+1;
    P;=P*X;
   UNTIL P/K<=A;
    WRITELN(‘K= ‘,K:3);
    END.
```

Пример 32. Составить программу для вычисления средних значений величин, изменяющихся одновременно по формуле

$$X_i = \frac{(a+b+c)}{3}$$

где a - переменная, изменяющаяся от значения 0.1 с шагом 0.1;

b - переменная, изменяющаяся от b_0 с шагом h_b;

c - переменная, значения которой вводятся в цикле.

```
PROGRAM PR32;
 CONST AO=0.1: HA=0.1;
VAR
  A,B,B0,HB,XI,C:REAL;
  I,N:INTEGER;
BEGIN
  WRITE(‘ Введите B0,HB,N);
  READLN(B0,HB,N);
A:=A0;
B:=B0;
I:=0;
   WRITELN(‘XI      A      B     C    ‘);
REPEAT
 I:=I+1;
  WRIT*E*(‘Введите C’);
  READLN(C);
```

```
XI:=(A+B+C)/3;
WRITELN(XI:5:2,' ', A:4;2,' ',B:4:2,' ',C:4:2);
  A:=A+HA;
  B:=B+HB;
UNTIL I<=N
END.
```

Пример 33. Составить программу вычисления корня N-ой степени $Y=\sqrt[n]{X}$ с заданной погрешностью E по следующему рекуррентному соотношению:

$$Y_0=X; \quad Y_K= Y_{K-1} + \frac{1}{N}\left(\frac{X}{Y_{K-1}^{N-1}} - Y_{K-1}\right);$$

K=1, 2, 3, ... до $| Y_K - Y_{K-1} | <= E$

Для упрощения решения задачи введем переменную

$$d = Y_K - Y_{K-1} = \frac{1}{N}\left(\frac{X}{Y_{K-1}^{N-1}} - Y_{K-1}\right);$$

значение которой использовано для формирования условия окончания цикла с постусловием.

```
PROGRAM PR33;
 VAR
  E,D,X,Y:REAL;
   N:INTEGER;
BEGIN
  WRITE(' Введите N,E,X);
  READLN(N,E,X);
   Y:=X;
    REPEAT
      D:=(X/EXP((N-1)*LN(Y))-Y)/N;
      Y:=Y+D;
    UNTIL ABS(D) <= E;
WRITELN('Корень ',N,' степени из ',X4:2,' равен ',Y:4:3);
END.
```

2.19. Символьные переменные

Обработка символьных (иначе: знаковых, литерных) данных становится возможной благодаря привлечению значений и переменных **типа CHAR**. Значениями типа CHAR служат все те символы, которые могут быть высвечены на экране: буквы, цифры, знаки операций, скобки, пробел и т.д. Исключение составляет штрих **'** , имеющий специальное назначение.

Если в программе имеется описание

n, w : char

то возможны, например, операторы присваивания n:='a', n:=v, w:='* ' и т.д. Штрих **' -** принятая в Паскале форма кавычки - употребляется всякий раз, когда значение типа char явно указывается в программе. Выполнение операторов n:='*c*'; writeln(n) приводит к появлению на экране символа *c*.

Рассмотрим пример 34. Пусть даны символы s_1, s_2,.Известно, что символ s_1 отличен от символа **/.** Пусть s_1, ... , s_n- символы данной последовательности, предшествующие первому символу / (n заранее не известно). Подсчитать общее количество символов и количество символов a среди s_1, ... , s_n

```
PROGRAM PR 34;
 VAR
  C: CHAR;
  I,N:INTEGER;
   BEGIN
    WRITE('Введите символ ');
     READLN(C);
    N:=1;
     I:=0;
WHILE  C <> '/' DO
 BEGIN
  IF C ='a' THEN I:=I+1;
   WRITE('Введите символ ');
    READLN(C);
N:=N+1;
 END;
   WRITELN(' Количество символов в строке = ', N:3);
   WRITELN(' Количество символов  а в строке = ', I:3);
END.
```

Символы, подготовленные для ввода, не отделяются друг от друга никакими знаками. Если три символа будут вводиться с помощью read(a,b,c), то можно будет набрать на клавиатуре **ч ш ц** и тогда значениями переменных a,b,c будут соответственно символы ч, щ, ц.

При построении условий, располагающихся после IF и WHILE, можно использовать разнообразные отношения в множестве символов. Здесь возможен не только знак = и комбинация <>, но и также >, >=, <, <=. так как все множество символов считается упорядоченным.

Рассмотрим пример вывода нарастающей последовательности букв латинского алфавита.

```
PROGRAM PR 35;
 VAR
  A,D: CHAR;
   BEGIN
    FOR C:='a' TO 'z' DO
     BEGIN
      FOR D:='a' TO C  DO
       WRITE(D);
       WRITELN(' ');
     END
   END.
```

Рассмотрим еще пример, в котором использована упорядоченность значений типа CHAR. Программа, в результате выполнения которой выясняется, имеется ли хотя бы одна малая латинская буква среди символов, предшествующих первому символу / в последовательности символов.

```
PROGRAM PR 36;
 LABEL 1;
  VAR
   C: CHAR;
    BEGIN
```

```
WRITE(‘ Введите символ ‘);
READLN(C);
  WHILE C <> ‘/’DO
    IF (‘a’<=C) and (C<=‘z’) THEN
      BEGIN
       WRITELN(‘Есть ‘);
        GOTO 1;
     END
                                ELSE
  BEGIN
   WRITE(‘ Введите символ ‘);
   READLN(C);
  END;
WRITELN(‘ Нет ‘);
  1: END.
```

Пример 37. Составить программу, учитывающую число посещений врачей (хирург, окулист, лор.) в поликлинике за день. В конце выдать итоговые данные.

```
PROGRAM PR37;
  VAR
   A,B,C,K,L,M:INTEGER;
   D:CHAR;
     BEGIN
     K:=0;
      L:=0;
      M:=0;
WRITELN('Введите число, месяц, год ');
READLN(A,B,C);
  WHILE D<> '/' DO
   BEGIN
    WRITELN(' Какой врач принимает:');
    WRITELN('   врач лор. - L:');
    WRITELN('   врач окулист - O');
    WRITELN('   врач хирург - X ');
    WRITELN('   прием закончен - / ');
     READLN(D);
       IF D ='L' THEN K:=K+1;
        IF D ='O' THEN L:=L+1;
         IF D ='X' THEN M:=M+1;
   END;
WRITELN(a:2,'.',b:2,'.',c:4);
WRITELN('посетили врачей: ');
WRITELN('лор. - ', K:3);
WRITELN('окулиста  - ', L:3);
WRITELN('хирурга - ', M:3);
    END.
```

Пример 38. Дана строка символов. Определить число сочетаний группы букв ABC и ABA.

```
PROGRAM PR3338;
 VAR
```

```
  C,I,J,F,K:INTEGER;
   CH:CHAR;
BEGIN
C:=0;  I:=0;  J:=0;  F:=0;   K:=0;
   WRITELN('Введите строку символов ');
     WHILE NOT EOLN DO
BEGIN
  READ(CH);
   IF CH<> '/' THEN C:=C+1;
    IF (I=0) AND (CH='A') THEN I:=1 ELSE
     IF (I=1) AND (CH='B') THEN I:=2 ELSE
                    IF (I=2) AND (CH='A') THEN
          BEGIN
            F:=F+1;
             I:=0
           END
                                              ELSE
    IF (I=2) AND (CH='C') THEN
      BEGIN
        K:=K+1;
         I:=0
       END;
END;
 IF C=0 THEN WRITELN(' ошибка: ввод пустой строки ')
            ELSE
              BEGIN
WRITELN(' Количество сочетаний "ABA" равно ',F:3);
WRITELN(' Количество сочетаний "ABC" равно ',K:3);
               END
END.
```

Пример 39. Дана строка символов, среди которых есть двоеточие. Получить все символы, расположенные между первым и вторым двоеточиями. Если второго двоеточия нет, то получить все символы, расположенные после единственного имеющегося двоеточия.

```
PROGRAM PR39;
 VAR
  C:CHAR;
   BEGIN
    REPEAT
READ(C);
  IF C=':' THEN
    REPEAT
     READ(C);
     WRITE(C);
    UNTIL EOLN OR (C=':')
   UNTIL EOLN
END.
```

Пример 40. Удалить из введенной последовательности символьности символ *.

```
PROGRAM PR40;
 VAR
```

```
 H:CHAR;
BEGIN
  WRITE(‘Введите строку символов’);
     REPEAT
       READ(H);
        IF H <> '*' THEN WRITE(H)
      UNTIL EOLN
END.
```

2.20. Процедуры и функции

В практике программирования необходимо проводить одни и те же вычисления. Это выполняется подпрограммой (ПП). Их в Паскале различают как процедуры и функции

Процедуры используются в тех случаях, когда необходимо в подпрограмме получить **несколько результатов**.

Функции представляют последовательность операторов, в результате выполнения которых вычисляется **одно значение**, присваемое имени функции.

Подпрограммы оформляются подобно программе: в начале заголовок ПП, затем следует декларативная часть ПП и после процедурная. В декларативной части описываются все данные, область действия которых ограничена телом данной ПП. Эти данные называются *локальными.* Данные, объявленные в основной (главной) программе, называются *глобальными* и они могут использоваться в любой ПП, входящий в основную программу. В процедурной части описывается тело ПП, реализующее алгоритм решения, и которое заключается в операторные скобки BEGIN и END.

В описание процедур включают: заголовок процедуры, раздел описаний (меток, констант, типов, переменных а также дополнительных процедур, являющихся локальными по отношению к описываемой процедуре), тело процедуры.

Имя процедуры выбирается пользователем в соответствии с правилами образования имен. Описание необходимых действий в процедуре осуществляется с помощью формальных (локальных) параметров, которые используются только в теле процедуры и локальные по отношению к ней. Блок процедуры заканчивается символом - точка с запятой.

Пример оформления процедуры.

```
PROCEDURE CUMMA;
  CONST N=10;
   VAR : INTEGER;
BEGIN
  FOR I:=1 TO N DO
    BEGIN
      Y:=A+B*I;
      Z:=4*Y-I;
     END
END;
```

Здесь A и B - входные параметры, их значения передаются из основной программы в эту процедуру. Результатом процедуры (выходными параметрами) являются вычисленные значения Y и Z, которые передаются из процедуры в основную программу и там могут быть использованы.

Выполнение подпрограммы начинается с операторов основной программы. Как только возникает необходимость действия процедуры, она вызывается по ее имени и начинает выполняться. Данные из основной программы (глобальные переменные) передаются процедуре. После выполнения процедуры результаты ее (выходные параметры) передаются в основную программу в то же место, откуда был сделан вызов процедуры. Затем продолжается выполняться основная программа.

Имена, объявленные в разделе описания основной программы, действуют в разделе операторов основной программы и в любой подпрограмме (процедуре или функции). Эти имена *глобальные*. Имена, объявленные в подпрограмме, действуют только в подпрограмме и в любой объявленной в ней процедуре и функции. Это имена *локальные*. Они недоступны для операторов основной программы.

Формальные параметры представляют собой список переменных с указанием их типа. Для выполнения выходных параметров перед ним ставится слово VAR.

В общем случае - процедура содержит один или несколько входных и выходных параметров, в том числе один и, в частности, ни одного.

Вызов процедуры в основной программе имеет следующую форму записи:

имя процедуры (фактический параметр);

Рассмотрим ряд практических примеров представления программ с процедурой (примеры и трассировка представлены Кривошеиным М.Ю.).

Параметры передаются по значению

```
1 Program pro1;
2  Var  a,b,c:integer;
3   Procedure pr(x,y:integer);
4    Var  c:integer;
5     Begin
6       x:=x+2;
7       y:=y-3;
8       c:=(x+y) DIV 2;
9    end;
10 BEGIN
11  readln(a,b);
12   c:=a+b;
13    pr(a,b);
14  writeln(a,’  ‘,b,’  ‘,c);
15 END.
```

Примечание: нумерация программ представлена для удобства построения таблицы трассировки.

Таблица трассировки

№	Ход выполнения	a	b	c			
10	Вход в pro1	?	?	?			
11		2	3				
12				5			
13	Вызов ***pr***				x	y	c
5	Вход ***pr***				2	3	
6					4		
7						0	
8							2
9	Выход из ***pr***						

14	Вывод a,b,c (2 3 5)			
15	Выход из pro1			

Различные примеры с изменением параметров представлены в приложении (Трассировка).

Рассмотрим еще один пример. Определить площадь, если известны все его стороны.
Решение. Рассмотрим решение данной задачи с использованием формулы Герона.

```
PROGRAM PR41;
  VAR
    AB,DC,CD,DA,AC,S1,S,a,b,c,p: REAL;
    PROCEDURE SRT1;
      BEGIN
        P:=(A+B+C)/2;
        S:=SQRT(P*(P-A)*(P-B)*(P-C))
       END;
BEGIN
  WRITE(‘ Введите данные AB,BC,CD,DA,AC’);
  READLN(AB,BC,CD,DA,AC);
    a:=AB; b:=BC; c:=AC, SRT1; S1:=S;
    a:= DA, b:=AC, c:=CD; SRT1; S1:= S1+S;
   WRITELN(‘ Площадь равна = ’,S:8:3)
END.
```

Рассмотрим тот же пример, но переменные a,b,c,s вынесены из основной программы в процедуру.

```
PROGRAM PR42;
 VAR
  AB,DC,CD,DA,AC,S1,S: REAL;
   PROCEDURE STR2 (VAR a,b,c,s: REAL);
     VAR
       P: REAL;
         BEGIN
          P:=(A+B+C)/2;
          S:=SQRT(P*(P-A)*(P-B)*(P-C))
          END;
BEGIN
  WRITE(‘ Введите данные AB,BC,CD,DA,AC’);
  READLN(AB,BC,CD,DA,AC);
STR2(AB,BC,AC,S1);
STR2(DA,AC,CD, S2);
   WRITELN(‘ Площадь равна = ‘,(S1+S2):8:3)
END.
```

Пример. Определить значения $y = \dfrac{\prod_{i=1}^{N} a^b + \sum_{i=1}^{M} c^d}{\prod_{i=1}^{L} e^f}$

```
program pr43;
 var
  a,b,x,y1,y,p,y2,y3,y5,y6,p1,p2,s:real;
```

```
 i,n: integer;
  procedure IN1(var a,b:real;n:integer;var y:real);
   var
    p:real;
    i:integer;
      begin
       y:=1;
        for i:=1 to n do
          begin
           p:=exp(b*ln(a));
            y:=p*y;
          end;
     end;
 procedure IN2(var a, b:real; n:integer; var y1:real);
  var
   p:real;
   i:integer;
    begin
     y1:=0;
       for i:=1 to n do
        begin
        p:=exp(b*ln(a));
        y1:=y1+p;
       end;
    end;
begin
 write(' Введите a, b,n');
 readln(a,b,n);
  IN1(a,b,n,y2);
    writeln('p1 ',y2:10:2);
     writeln(' Введите c, b,m');
     read(a,b,n);
   IN2(a,b,n,y3);
   writeln('s ',y3:10:2);
   write(' Введите e, f,L');
     readln(a,b,n);
     IN1(a,b,n,y5);
     writeln('p2 ',y5:10:2);
      y6:=(y2+y3)/y5;
writeln(y6:8:3);  end.
```

2.21. Функции

В Паскале, помимо процедур, разрешены похожие на них конструкции, которые называются функциями. Обращение к функции приводит к вычислению ее значения - объекта типа *real, integer, char.* Тип фиксируется в описании функции.

Имеются две отличительные особенности описания функции в сравнении с описанием процедуры. Первая особенность связана с заголовком, он должен начинаться со служебного слова **function** (функция) и заканчиваться именем того типа, которому принадлежит значение функции.

Например:

```
 function f(a:real; var b:t):rael;
```

function g(var a,b:integer): integer;
function h(a:integer): char;

Вторая особенность: составной оператор, который располагается в описании функции после заголовка и, возможно, после описания локальных меток и переменных, должен обязательно содержать внутри себя оператор присваивания, в котором слева от := помещено имя функции, например:

```
f:=3,14
g:=a+2*b
if a mod 2=0 then h:='*' else h:='!'
```

Таких операторов присваивания может быть несколько, но при каждом конкретном обращении к функции «сработать» должен **только один** из них. Это присваивание и определит значение функции.

Относительно параметров функции имеется полная аналогия с параметрами процедур. Описание функции, как и описания процедур, располагаются в программе после совокупности описания процедур.

Расмотрим пример программы с трассировкой

```
1 Program pro6;
2  Var  a,b,c:integer;
3   function pr(x, y: integer) : integer;
4     Begin
5       x:=x+2;
6       y:=y-3;
7       pr :=(x+y) DIV 2;
8    end;
9 BEGIN
10  readln(a,b);
11   c:= pr (a,b);
12  writeln(a,'  ',b,'  ',c);
13  END.
```

Таблица трассировки

№	Ход выполнения	a	b	c	x	y	pr
9	Вход в pro6	?	?	?			
10		2	3				
11	Вызов ***pr***				x	y	pr
4	Вход ***pr***				2	3	
5					4		
6						0	
7							2
8	Выход из ***pr***						
11	Возврат в pro6			2			
12	Вывод a,b,c (2 3 2)						
13	Выход из pro6						

Пример. Рассмотрим определение площади четырехугольника по формуле Герона.

```
PROGRAM PR44;
 VAR
 AB,BC,CD,DA,AC:REAL;
  funcrion tr1(a, b, c: real): real;
    var p:real:
     begin
      p:=(a+b+c)/2;
       tr1:=sqrt(p*(p-a)*(p-b)*(p-c))
     end;
BEGIN
 WRITE(‘ Введите AB,BC,CD,DA,AC ‘);
 READLN(AB,BC,CD,DA,AC);
  WRITE(tr1(AB,BC,AC) +tr1(CD,DA,AC):8:3)
END.
```

Пример. Вычислить значение $y = \sum_{i=1}^{n} a^i + \sum_{j=1}^{m} b^j$

```
program pr45;
 var
  a,b,S:real;
  n,e:integer;
  function TR(n:integer;m:REAL):real;
   var
    k:real;       i:integer;
      begin
       S:=0;
      for i:=1 to n do
        begin
      k:=exp(i*ln(m));        S:=k+S;
      TR:=S;
        end;
      end;
begin
 write(' Введите n,a ');
 readln(n,a);
   write(' Введите m,b ');
 readln(m,b);
 S:=TR(n,a)+TR(m,b);
 writeln('S= ',S:8:3); end.
```

2.22. Сложный тип данных - массивы

Типы данных делятся на простые и сложные. К простым типам относятся стандартные, перечисляемые и ограниченные. К сложным типам - массивы, множества, записи, файлы. Элементами сложных типов могут быть простые и сложные типы.

Упорядоченный набор объектов одного типа данных (числа, таблицы и т.д.) называются массивами. Массив обозначается одним именем. Каждый элемент массива обозначается именем массива с индексом. Элементы массива упорядочены по значениям индекса.

Массивы описываются в разделе VAR или в разделе TYPE (тип).

Пример записи:

```
CONST
```

```
N=20;
 TYPE
  T=ARRAY [ 1..N ] OF REAL;
 VAR
A:T;
```

здесь: ARRAY - массив;
OF (из) - служебное слово;
[1..N] тип индекса, в качестве которого может быть любой простой тип, кроме стандартных типов REAL и INTEGER;
A - тип элемента массива.

или

```
TYPE
 T=ARRAY [ 1..20 ] OF INTEGER;
 VAR
  A:T;
```

или

```
VAR
 A:ARRAY[1..5] of real;
```

где A - имя массива, типа REAL;
Тип индекса - ограниченный от 1 до 5.
Пример описания массивов:

```
VAR
 massiv: array[1..5] of real;
 год: array[январь.. декабрь] of integer;
 L: array[строка] of BOOLEAN;
```

Если несколько массивов имеют одинаковый тип индексов и одинаковый базовый тип, то

```
var
 a,b,c:array[1..10] of real;
```

Нельзя путать понятие "ИНДЕКС" и "ТИП ИНДЕКСА". Тип индекса используется только в разделе описания массива, а индекс указывается в разделе операторов для обозначения конкретного элемента массива. При этом индекс должен быть того же типа, что и описание типа индекса.

В качестве индекса может быть выражение, частным случаем которого является константа или переменная. Элемент массива иначе называется переменная с индексом. В отличие от нее переменная без индекса называется простой переменной.

Элементы массива могут стоять как в левой части оператора присвоения, так и в выражениях. Над элементами массива можно производить те же операции, которые допустимы для данных его базового типа.

Примеры.

```
B[5]:=B[3]+1;
SUM:=SUM-C[K];
P1:=A[2*I+1];
```

Для ввода и вывода числовых значений массива используются циклы:
Например:

а) for i:=1 to 9 do
read(a[i]);

В цикле организуется **ввод** девяти значений элементов массива А: A[1], A[2],..A[9].

б) for i:=1 to 9 do
write(a[i]);

В цикле организуется **вывод** девяти значений элементов массива А: A[1], A[2],..A[9].

2.23. Тип массива

Имеется еще одна форма описания, состоящая из двух этапов:

1. Сначала в разделе описания типа TYPE указывается тип массива.
2. Затем, в разделе описания переменных VAR перечисляются массивы, относящиеся к указанному типу.

Форма объявления массива имеет вид:

```
TYPE
имя =array[1..20] of real;
 Var
  имя: имя типа;
    [1..20]  - тип индекса;
     real - базовый тип элемента массива.
```

Пример:

```
  TYPE
    s1=array[1..15] of integer;
    var x:s1;
```

Пример 45. Определить самый высокий рост спортсмена.

```
program pr45;
 const n=10;
  type mas= array[1..n] of real;
   var
   s:mas;
   max:REAL;
     i:integer;
begin
  write('Введите роста 10 спортсменов через интервал');
   for i:=1 to n do
    readln(s[i]);
  max:=s[1];
   for i:=2 to n do
    if s[i]>max then max:=s[i];
  writeln;
writeln('максимальный рост',max:8:3);
end.
```

Пример 46. Дано $a_1,a_2,..a_{20}$. Получить $a_{20},a_{10};a_{19},a_9;..;a_{10}..a_1$.

```
program pr46;
 type t = array[1..20] of integer;
  var
   x:t;
   i:integer;
begin
```

```
 write('Введите 20 чисел');
  for i:=1 to 20 do
   readln(x[i]);
    for i:=0 to 9 do
    writeln(x[20-i],'  ',x[10-i]);
end.
```

Пример 47. Определение наименьшего элемента массива и его порядкового номера.

Решение. Сначала целесообразно в качестве наименьшего значения взять значение первого элемента и сравнить его со вторым, третьим, при выполнении условия xMin:=x[1] и IMin:=1. Так как значение первого элемента может оказаться наименьшим, то перед циклом наряду с оператором xMin:=x[1] необходимо записать IMin:=1.

```
program pr47;
 const Nmax=50;
  { Xmin - минимальное значение массива,}
  { Imin - номер минимального значения }
   var
    xmin:real;
    imin,i:integer;
    x:array[1..nmax] of real;
Begin
 write ('Введите элементы массива');
  for i:=1 to nmax do
   readln(x[i]);
xmin:=x[1];
imin:=1;
  for i:=2 to nmax do
   if x[i] < xmin then
begin
 xmin:=x[i];
 imin:=i;
end;
  writeln('Минимальное значение =',xmin:8:3);
  writeln('Порядковый номер =',imin:3);
end.
```

Пример 48. Вывести треугольную матрицу, относительно диагонали.

```
program pr48;
 type mas=array [1..4,1..4] of real;
  var
   j,i:integer;
   a:mas;
begin
  for i:=1 to 4 do
   for j:=1 to 4 do
     begin
      write('a[',i:2,j:2,']');
       readln(a[i,j]);
     end;
for i:=1 to 4 do
 begin
  for j:=1 to 4 do
```

```
  if (i=j) or (i<j)
              then  write(a[i,j]:8:3)
               else write('         ');
     writeln;
  end;
end.
```

Пример 49. Составить программу для упорядочивания элементов массива A(a1,a2,.a10.).
Решение. Во внутреннем цикле найдем наибольший элемент массива или его части. Вначале перед циклом зададим начальное значение наибольшего, равное первому элементу массива, а затем внутри цикла найдем наибольший элемент и его порядковый номер. После окончания внутреннего цикла наибольший элемент запишем на место первого элемента рассматриваемой части массива, а первый на его место. Затем найдем наибольший элемент массива, начиная со второго, далее с третьего и т.д. Следовательно, указанные действия надо повторять во внешнем цикле. Внешний цикл должен повторяться для К, изменяющегося от 1 до 9, так когда останется последний элемент, находить наибольший не имеет смысла.

```
Program pr49;
 const nk=10;
 var
  amax:real;
  k,i,kmax:integer;
  a:array[1..nk] of real;
   begin
writeln(' Введите элементы массива ');
  for i:=1 to nk do
   read(a[i]);
    for k:=1 to nk-1 do
{ поиск максимального числа и его порядкового номера}
begin
 amax:=a[k];
 kmax:=k;
  for i:=k+1 to nk do
   if a[i] > amax then
begin
 amax:=a[i];
  kmax:=1;
end;
{перестановка максимального элемента с текущим }
  a[kmax]:=a[k];
  a[k]:=amax;
end;
{Вывод элементов упорядоченного массива }
  for i:=1 to nk do
   writeln (a[i]:9:5);
    end.
```

Пример 50. Заданы два вектора x и y. Являются ли они поэлементно равными.

```
program pr50;
 var
  i:integer;
  a:boolean;
   x,y: array[1..10] of real;
```

```
begin
  write('Введите 1 вектор');
   for i:=1 to 10 do
    readln(x[i]);
    write('Введите 2 вектор');
  for i:=1 to 10 do
   readln(y[i]);
    i:=1;a:=true;
     while a and (i<> 10) do
begin
  a:=x[i]=y[i];
  i:=i+1;
writeln('Ответ :',a);
end.
```

Пример 51. Определить минимальное значения в двух массивах.

```
program pr51;
 var
  a:array [1..8] of integer;
  min,i,n,z,w,e:integer;
procedure as1;
 var
  i,s,z:integer;
   begin
    write('Введите число элементов в массиве');
    readln(z);
    writeln('Введите массив');
      for i:=1 to z do
       begin
         readln(a[i]);
       end;
min:=a[1];
 for i:=1 to z do
  begin
   if a[i]>min then a[i]:=min;
  end;
writeln('min=',min:8);
  end;
begin
 as1;
 e:=min;
  as1;
  w:=min;
if e>w then
  writeln('Минимальное значение у 1 массива=',w:8)
  else writeln('Минимальное значение у 2 массива=',e:8);
end.
```

Пример 52. Определить сумму элементов в матрице над и под главной диагональю.

```
program pr52;
 const n=4;
  var
  a:array[1..n,1..n] of integer;
```

```
 i,j,s,s1:integer;
  begin
  for i:=1 to n do
   for j:=1 to n do
      read(a[i,j]);
  for i:=1 to n do
 begin
   for j:=1 to n do
   write(a[i,j]:3;
    writeln;
 end;
  s:=0;
for i:=1 to n do
 begin
   for j:=1 to n do
if j>i then s:=s+a[i,j];
 end;
  s1:=0;
 for i:=1 to n do
 begin
   for j:=1 to n do

  if j<I then  s1:=s1+a[i,j];
  end;
    writeln('Сумма элементов над глав. диагональю = ',s:4,);
    writeln('Сумма элементов под глав. диагональю = ',s1:4);
end.
```

Пример 53. Произвести транспонирование матрицы.

```
program pr53;
 const  m=3; n=2;
  var
   a:array [1..m,1..n] of integer;
   b:array [1..m,1..n] of integer;
   i,j:integer;
begin
  writeln('Введите массив');
   for i:=1 to m do
    for j:=1 to n do
     read(a[i,j]);
writeln('Исходная матрица ');
for i:=1 to m do
 begin
   for j:=1 to n do
   write(a[i,j]:3;
    writeln;
  end;
for j:=1 to n do
 for i:=1 to m do
  b[j,I]:=a[I,j];
writeln('Транспонированная  матрица ');
for i:=1 to n do
```

```
  begin
 for i:=1 to n do
  for j:=1 to n do
     read(a[i,j]);
 for i:=1 to n do
begin
  for j:=1 to n do
  write(a[i,j]:3;
   writeln;
end;
  writeln (‘ Элементы побочной диагонали над главной ‘);
 for i:=1 to n do
   begin
    for j:=1 to n do
       if (j>i) and (i+j=2*i+1)  then   writeln (a[i,j]);
   end;
  writeln (‘ Элементы побочной диагонали под главной ‘);
 for i:=1 to n do
   begin
    for j:=1 to n do
       if (j<i) and (i+j=2*j+1)  then   writeln (a[i,j]);
   end;
end.
```

Пример 55. Дан одномерный массив. Повести ранжирование элементов массива по возрастанию.

```
program pr55;
  const  n=4;
 var
  a:array [1..n] of integer;
  i,x,k:integer;
begin
 writeln(' Введите массив из ',n,’  чисел  ‘);
  for i:=1 to n do
   read(a[i]);
 for i:=1 to n do
  for k:=n-1 downto  1 do
   if a[k]>a[k+1] then
    begin
    x:=a[k];
     a[k]:=a[k+1];
     a[k+1]:=x;
    end;
for I:=1 to n do
writeln('  ‘,a[i]:4);
 end.
```

2.24. Строки

Для обработки строки символов, например, фамилии, названия и т.д. используется тип данных string. Этот тип данных облегчает программирование операций над строками.

В описании типа string длина строки может принимать любое целое значение от 1 до 255. Если длина не указана, то берется максимальная длина 255. Для переменной типа

string[n] выделяется n+1 байт памяти. К конкретному символу строки можно обратиться как к элементу массива. Пусть строка s определена следующим образом:

```
type
  st20 = string[20]; var
  s : st20;
```

Тогда после присваивания s: = 'Едят ли кошки мошек', выражение s[7] будет иметь значение 7-го элемента присвоенной строки, а именно 'и'. Длину строки можно получить с помощью стандартной функции length. Строка, обозначенная двумя апострофами, стоящими рядом друг с другом, т.е. '', - это строка, не содержащая ни одного символа, или пустая строка.

Для слияние нескольких строк используют операцию сложения +. Рассмотрим их подробнее.

Функция length(s) возвращает длину строки **s**. Результат типа *integer*. Например, length('uiopbKH') равно 7, a length('') равно О

Строковой переменной можно присвоить любую строку, жесткого контроля соответствия типов, как в случае массивов, здесь нет. Если возможная длина строки меньше, чем длина присваиваемого значения, то лишние символы отбрасываются.

Рассмотрим несколько фрагментов программ обработки строк. Пусть, к примеру, требуется подсчитать число букв "a", в строке, введенной пользователем.

```
program pr56;
var J,x:INTEGER;
 {определение количества символов "a" в строке}
 f:string;
  begin
   write('Введите символы');
   readln(f) ;
    x:=0;
     for j:=0 to length(f) do {length(f)-конец строки f}
      if f[j]='a' then x:=x+1;
        if x=0 then write('Символа "a" в строке нет ')
          else write('Строка содержит ',x,' символ(а)\/(ов) "a" ');
end.
```

```
program pr56;
 var J:INTEGER;
 {Программа запрашивает ФИО,здоровается и приветствует}
  F,I,O,fio,gL,sgL,s,T:string;
 {используется строковый тип данных}
begin
  write('Фамилия ?');
  readln(f) ;   ;
  write('Имя ? ');
  readln(I);
  write('Отчество ? ');
  readln(o);
    fio:=F+' '+I+' '+O;
    s:='ВАМ '; gl:='     ПРИВЕТ';T:='ЗДРАСТВУЙТЕ';
    sgl:= gl+'  '+s ;
      for J:=1 to 25 do writeln;
       writeLN(' ':30,t);
       writeLN;
```

```
    writeLN;
    write(' ':20, fio);
    writeln(sgl) ;
      for J:=1 to 15 do writeln;
 READLN
end.
```

2.25. Файлы

Удобным способом сохранения информации, полученной в ходе выполнения программы, служит запись этой информации на магнитный носитель. Запись особенно желательна, если объем информации велик, предусмотрена использование в дальнейшем эту информацию, а также при использовании определенной базы данных для расчета.

В Паскале предусмотрены специальные объекты – файлы, операции над которыми сводится к работе с носителями информации.

Файл – это по*следовательность компонент*, являющихся объектами одного и того же типа. Количество компонент в файле заранее не оговаривается, компоненты файла не имеют индексов. До некоторой компоненты можно добраться, только перебрав по очереди все. Описание, имеющее вид

***V**=file of integer*

Это описание типа, имя которого ***V***. Объектами типа ***V*** будут файлы с целочисленными компонентами.

О*перации над файлы – запись в файл*. Пусть ***C*** – имя рассматриваемого файла, и пусть ***a*** – переменна того типа, объектами которого являются компоненты файла.

Запись в файл возможен, когда файл открыт. При соблюдении этого условия выполнение оператора *write(**C.a**)* приведет к тому, что в файл будет записана еще одна компонента, равная значению переменной ***a***. Перед выполнением оператора *write(**C.a**)* проверяется, описан идентификатор ***C*** как переменная, значением которой должен быть файл. Если да, то происходит запись в файл, иначе оператор *write(**C.a**)* выполняется как обычный оператор вывода. Это же касается оператора *read(**C.a**)*

Пример. Пусть ***a*** – файл, компонентами которого могут быть целые числа. Приведем фрагмент программы, обеспечивающий запись в ***a*** квадрат ста первых натуральных числе:

```
Rewrite(a); {служебное слово, для записи в файл}
 For I:=1 to 100 do
  Begin
   j:=sqr(i);
    write(f,j);
  end
```

переменные *I* и *j* должны быть типа *integer.*

Операции над файлы – чтение из файл. Если файл не пуст, то начинается чтение из файла исходной информации, если файл пуст – то при чтении указывается признак конца файла.

Пример 57. Фрагмент программы, обеспечивающей чтение из файла *C*, компонентами которого служат действительные числа, всех его компонент и вычисление суммы их квадратов.

```
reset(c);  {операции над файлы –чтение файла}
 s:=0;
  while not eof(c) do
    begin
     read(c,t);
      s:=s+sqrt(t);
    end
```

(переменные *c* и *t* должны иметь тип *real*)

Имя каждого файла, который будет использован в данной программе, должно быть указано в скобках после названия программы.

Например:

```
Program Prim(C);
Program Prim(C,C1);
```

Программа должна содержать описания тех файловых типов, к которым принадлежат файлы, упомянутые в заголовке программы. Идентификаторы, служащие именами этих файлов, должны быть описаны в программе как переменные соответствующих типов.

Пример 58. Программа, в результате выполнения которой выводятся все малые латинские буквы из данного символьного файла *prim2fail*.

```
Program pr58(prim2fail);
 Type v=file of char;
  Var
   prim2fail:v;
  s:char;
   begin
    reset (prim2fail);
     while not eof(prim2fail) do
       begin
        read(prim2fail,s);
         if (s<= ' z ') and (s>= ' a ') then writeln(s)
       end
end.
```

3. ПРАКТИЧЕСКИЕ ПРИМЕРЫ РЕАЛИЗАЦИИ ПРОГРАММ НА РАСКАЛЕ

В данном разделе рассмотрены реализованные программы на Паскале, как образец реализации поставленной задачи.

{Дано натуральное число N. Разложить его на простые множители}

```
Program pr101;
 var
  i,j,n,f:integer;
  begin
   repeat
    write(' Введите натуральное число N ');
    readln(n);
   until N>0;
    write(N:6,' = 1');
     f:=0;  j:=n;
     for i:=2 to n div 2 do
      begin
       if j mod i=0 then
        begin
         f:=1;
          while j mod i=0 do
           begin
            write(' * ',i);
```

```
          j:=j div i
         end;
       end;
     end;
  if f=0 then writeln (' * ', n)
         else writeln
end.
```

Другие примеры приведены в приложении на дискете (Практические примеры реализации)

4. РАСШИРЕНИЕ СТАНДАРТНОГО ПАСКАЛЯ В ОБЛАСТИ ГРАФИКИ

Данный раздел приведен в приложении на дискете (Расширение стандартного Паскаля в области графики).

5. САМОСТОЯТЕЛЬНАЯ РАБОТА

В данном разделе предложены задачи для вывода исходного условия по листингу программы. Примеры листинга приведены в приложении на дискете (Самостоятельная работа).

6. СРЕДА ПРОГРАММИРОВАНИЯ ТУРБО ПАСКАЛЬ

Разработка программ на языке Паскаль включает в себя следующие этапы: ввод и редактирование текста программы на языке программирования, ее трансляцию, отладку.

Для выполнения каждого этапа применяются различные средства: для ввода и редактирования текста используется редактор текстов, для трансляции программы – компилятор, для построения исполняемого компьютером программного модуля с объединением разрозненных откомпилированных модулей и библиотек процедур Паскаля – компоновщик (linker), для отладки программ с анализом ее проведения, поиском ошибок, просмотром и изменением содержимого ячеек памяти – отладчик (debugger).

Интегрированная среда программирования Турбо Паскаль имеет следующие возможности:

множество накладных окон;
поддержка мыши, меню, диалоговых окон;
многофайловый редактор, который может редактировать файлы до 1Мб;
расширенные возможности отладки;
полное сохранение и восстановление среды разработки;
объектно – ориентированная среда разработки прикладных программ;
полные возможности встроенного ассемблера;
личные поля и методы объявления объектов;
директива расширенного синтаксиса;
адресные ссылки в типизированных программах;
директивы ближних и дальних процедур;
редактирование инициализированных данных из объектных файлов;
расширенные возможности справочной системы;
более быстрый монитор кучи (куча - сплошной массив байтов в памяти).

Основные файлы пакеты Турбо Паскаль

TURBO.EXE – интегрированная среда программирования;
TURBO.HLP – файл, содержащий данные для оперативной подсказки;
TURBO.TP – файл конфигурации системы;

TURBO.TPL - библиотека стандартных модулей.

В каталоге P\BGI находятся модули, необходимые для работы в графическом режиме: GRAPH.TPU - модуль с графическими процедурами и функциями, несколько файлов с расширением .BGI – драйверы различных типов видеосистем компьютеров, несколько файлов с расширением .CHR, содержащие векторные шрифты.

Более подробно среда программирования приведена в приложении на дискете (Среда программирования Турбо Паскаль).

Список рекомендованной литературы

1. Попов В.Б. Turbo Pascal для школьников: Учебное пособие для педагогических вузов и общеобразовательных учебных заведений физико-математического профиля. - М.: Финансы и статистика, 1999.

2. Инструментальные средства персональных ЭВМ. В 10 кн. Кн.4. Программирование в среде ТурбоПАСКАЛЬ: Практ. пособие. Под ред. Б.Г.Трусова.- М.: Высш.шк.,1993.

3. Гусева А.И. Учимся информатике: задачи и методы их решения.-М.:Диалог-МИФИ, 1998.

4. Лабораторный практикум по программированию на языке Паскаль: Учебное пособие./Под общ. ред. Л.В.Найхановой и Н.Ц. Бильгаевой. -2-е изд. доп. и перераб.,- Улан-Удэ, 1999.

5. Фаронов В.В. ТурбоПаскаль 7.0: Практика программирования: Учебное пособие. - М.: Нолидж, 1999

6. Фаронов В.В. ТурбоПаскаль: В 3-х кн. -.Кн.3. Ч2: Практика программирования. – 1993, М.:МВТУ - Фесто Дидактик, 1993.

7. Лукин С.Н. Турбо-Паскаль 7.0. Самоучитель для начинающих. – 2-е изд. Испр. И доп. – М:Диалог-МИФИ, 2001

8. Иринчеев А.А. Паскаль для начинающих. Учебное пособие. ВСГТУ, - Улан-Удэ, 1999.

9.Зубов В.Е. Программирование на языке TURBO PASCAL (версия 6.0 и 7.0): Справочник по процедурам, функциям, диагностическим сообщениям. – М.: Филингъ, 1997,: ил.

ОГЛАВЛЕНИЕ

	Стр.
ВВЕДЕНИЕ	2
1. ПРЕДСТАВЛЕНИЕ ИНФОРМАЦИИ В КОМПЬЮТЕРЕ	3
1.1. Позиционная система счисления	4
1.2. Перевод целого числа из десятичного счисления в другое	6
1.3. Перевод дробного числа из десятичного счисления в другое	7
1.4. Перевод чисел в десятичную систему счисления	8
1.5. Форматы данных и машинные коды чисел	9
1.6. Нормальная форма числа или представление числе в форме с плавающей точкой	10
2. ОСНОВНЫЕ ПОНЯТИЯ О ЯЗЫКЕ ПАСКАЛЬ	11
2.1. Алгоритмизация задач	22
2.2. Виды и свойства алгоритма	23
2.3. Стандартные функции	24
2.4. Трассировка программы	25
2.5. Построение (разработка) программ	27
2.6. Управляющие конструкции языка. Условный оператор	29
2.7. Оператор выбора CASE	35
2.8. Константы	36
2.9. Метки и оператор перехода	37
2.10. Организация циклических процессов	38
2.11. Оператор цикла с параметром	39
2.12. Табуляция функций	40
2.13. Вычисление суммы	42
2.14. Вычисления произведения	42
2.15. Вычисления факториала	43
2.16. Вложенные циклы	44
2.17. Оператор цикла с предусловием	46
2.18. Оператор цикла с постусловием	50
2.19. Символьные переменные	53
2.20. Процедуры и функции	56
2.21. Функции	60
2.22. Сложный тип данных - массивы	62
2.23. Тип массива	63
2.24. Строки	69
2.25. Файлы	71
3. ПРАКТИЧЕСКИЕ ПРИМЕРЫ РЕАЛИЗАЦИИ ПРОГРАММ НА ПАСКАЛЕ	73
4. РАСШИРЕНИЯ СТАНДАРТНОГО ПАСКАЛЯ В ОБЛАСТИ ГРАФИКИ	74
5. САМОСТОЯТЕЛЬНАЯ РАБОТА	74
6. СРЕДА ПРОГРАММИРОВАНИЯ ТУРБО ПАСКАЛЬ	74
СПИСОК РЕКОМЕНДОВАННОЙ ЛИТЕРАТУРЫ	76

Алексей Алексеевич Иринчеев,
Александр Михайлович Мангадаев

Паскаль в примерах.

Учебное пособие

Редактор Т.А.Стороженко

ЛР № 020456 от 30.07.1997 г.

Подписано в печать . Формат 60x84 1/16
Усл. п.л. 6.75, уч.-изд.л. 6.24
Печать операт., бум. писч.
Тираж 150 экз.
Издательство ВСГТУ. Г. Улан-Удэ, ул. Ключевская , 40а