Федеральное агентство по образованию
АМУРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ГОУВПО «АмГУ»
Факультет математики и информатики

УТВЕРЖДАЮ
Зав. кафедрой МАиМ
Т.В. Труфанова
«___»_____ 2008г.

МОДЕЛИРОВАНИЕ В СРЕДЕ ТУРБО ПАСКАЛЬ

Методические материалы

Составитель: Д.Л. Харичева

Благовещенск
2008

ББК
Х

Печатается по решению редакционно-издательского совета факультета математики и информатики Амурского государственного университета

Харичева Д.Л.

Моделирование в среде турбо Паскаль. Методические материалы для студентов очной формы обучения по специальности 010101 «Математика». – Благовещенск: Амурский гос. ун–т, 2008. − 100 с.

СОДЕРЖАНИЕ

ПРЕДИСЛОВИЕ

Большинство изучающих Паскаль и его основы имеют нулевой уровень подготовки, поэтому за короткое время требуется создать им возможность освоить курс программирования и научить писать программы. Методическое пособие ориентировано на аудиторию, не имеющую представления об основных конструкция и алгоритмах в программировании и предназначено для использования при изучения курса «Информатика» для студентов очной формы обучения по специальности 010101 «Математика». Поэтому пособие построено следующим образом:

1 часть – основные конструкции языка

2 часть – основные приемы работы с оболочкой TURBO Pascal

3 часть – работа с данными

4 часть – графические возможности TURBO Pascal

5 часть – основные приемы программирования и алгоритмы

Данное пособие не дает полного описания языка паскаль и не включает в себя описание системы TurboVision, оно предназначено для начала изучения языка и принципов программирования в целом.

Все примеры и программы составлены в среде TURBO Pascal 7.0.

1. ОСНОВНЫЕ КОНСТРУКЦИИ ЯЗЫКА ПАСКАЛЬ

1.1. Типы данных, константы, переменные, выражения, структура программы, последовательные алгоритмы.

Основной принцип работы компьютеров – последовательное выполнение команд для работы с данными. Последовательность команд, выполняемая основным модулем компьютера, – процессором, называется программой. Программу набирают в любом текстовом редакторе или встроенной интерактивной среде.

При необходимости в текст программы можно добавить комментарии, которые выделяются фигурными скобками:

Var x,y: integer; {координаты фигуры}

Любая программа оперирует с некоторым набором данных. В Паскале существует несколько простых типов данных, которые можно разделить на виды: числовые, строковые, булевы, и сложные типы данных, такие как массивы структуры, множества.

Рассмотрим каждый тип данных:

Числовой. Паскаль, как и многие другие языки программирования, имеет несколько числовых типов: целочисленные (ShortInt, Integer, Longint, Byte, Word).

Тип	Диапазон
Shortint (короткое целое)	-128...128
Integer (целое)	-32768...+32767
Longint (длинное целое)	-2147483648...+2147483647
Byte (длиной в байт)	0...255
Word (длиной в слово)	0...65535

Вещественные (Single, Real, Double, Extended, Comp). Паскаль допускает представление вещественных значений в виде как с плавающей,

так и с фиксированной точкой.

Примеры записи чисел с плавающей точкой;

Обычная запись в математике	Запись с плавающей точкой в TP.
$1.3653*10^{8}$	1.3653E+8
$6.63*10^{-34}$	6.63E-34
$-1.6*10^{-19}$	-1.6E-19
$3*10^{8}$	3E8

Вещественные числа имеют ограничения, связанные с фиксированным размером выделяемой под них памяти:

Тип	Диапазон	Число значащих цифр	Байт памяти
Real	-2.9E-39...1.7E38	11-12	6
Single	-1.5E-45...3.4E38	7-8	4
Double	-5.0E-324...3.4E308	15-16	8
Extended	-3.4E-4932...1.1E4932	19-20	10
Comp	-9.2E-18…9.2E18	8	8

Выражение в Паскале может содержать операции и функции, как встроенные так и пользовательские. Приоритет операций приведен ниже.

Приоритет	Операция
1	not, @
2	*, /, div, mod, and, shl , shr
3	+, - , or, xor
4	=, <>, >, >=, <, <=, in

Значение операций:

Not – логическое НЕ

And – логическое И

Or – логическое ИЛИ

Xor – логическое исключающие ИЛИ

Div – целочисленное деление

Mod -- взятие остатка от деления

In – вхождение в подмножество

@ – получение значения, на которое ссылается указатель

Shl – побитовый сдвиг влево

Shr – побитовый сдвиг вправо

Для вычисления выражений в Паскале реализованы основные математический функции. Если передаваемый параметр не соответствует декларированному (объявленному) типу, то Паскаль по умолчанию делает привидение значения параметра к нужному типу. Список и назначение основных арифметических функций:

SIN(X) - синус X (аргумент задается в радианах);

COS(X) - косинус X (аргумент задается в радианах);

ARCTAN(X) - арктангенс X (аргумент задается в радианах);

LN(X) - натуральный логарифм X;

EXP(X) – e^X;

SQR(X) - квадрат X;

SQRT(X) - квадратный корень X;

ABS(X) - модуль X;

TRUNC(X) - целая часть числа X, результат - integer;

INT(X) - целая часть числа X, результат - real;

FRAC(X) - дробная часть числа X, результат - real;

ROUND(X) - округление до ближайшего целого, результат - integer;

PI - число 3.1415926535897932385;

ODD(X) - проверка четности целого числа "X" - функция выдает значение TRUE (истина), если число нечетное, FALSE (ложь) - если число четное.

Символьный (char) тип определяется множеством значений кодовой таблицы ASCII. Для переменной символьного типа требуется 1 байт.

Символьная константа может записываться в тексте программы тремя способами:

– как один символ, заключенный в апострофы, например: 'A', 'a', 'Ю', 'ю';

– с помощью конструкции вида #K, где K - код соответствующего символа, при этом значение K должно находиться в пределах 0..255;

– с помощью конструкции вида ^C, где C - символ, код которого на 64 больше кода управляющего символа.

К величинам символьного типа применимы все операции отношения.

Для величин символьного типа определены две функции преобразования: Ord(C) и Chr(K).

Первая функция определяет порядковый номер символа C в наборе символов, вторая определяет по порядковому номеру K символ, стоящий на K-ом месте в наборе символов. Порядковый номер имеет целый тип.

К аргументам символьного типа применяются функции Pred(C) и Succ(C), которые определяют предыдущий и последующий символы:

Pred('F') = 'E' ; Succ('Y') = 'Z' .

При отсутствии предыдущего или последующего символов значение соответствующих функций не определено.

Для литер из интервала 'a'..'z' применима функция UpCase(C), которая переводит эти литеры в верхний регистр 'A'..'Z'.

Логический тип (Boolean, ByteBool, WordBool, LongBool) представлен двумя значениями: (истина) и (ложь). Он широко применяется в логических

выражениях и выражениях отношения. Типы ByteBool, WordBool, LongBool являются нововведением Turbo Pascal 7.0, они были введены для обеспечения совместимости создаваемых программ в Windows. Над логическими переменными действуют стандартные операции not, and, or. Так

True OR False = True

Указательный тип (Pointer) - значениями переменных и констант данного типа являются адреса оперативной памяти, состоящие из адреса сегмента и смещения.

В Паскале существуют типы данных, определяемые пользователем. Это перечислимый тип (когда непосредственно, в разделе описания типов, заранее записываются все значения для переменных этого типа) и интервальный (когда задаются границы диапазона значений для данной переменной), указательный тип (кроме Pointer), структурированные типы и процедурный тип. Для получения значения переменной, хранящееся в памяти по адресу, на который указывает указатель используется унарная операция @.

Перечислимый тип состоит из списка констант. Переменные этого типа могут принимать значения любой из этих констант. Описание перечислимого типа имеет вид:

Type

< имя типа > = <(список констант)>;

Var

< имя переменной >: < имя типа >;

Под константой понимается особый вид констант, задаваемый пользователем. Под списком понимается перечень констант, разделенных запятыми. Сам список заключается в круглые скобки. Например:

Type

year = (winter, spring, summer, autumn);

Var

avar : year;

Здесь year - имя перечислимого типа, winter ,... - константы; avar - переменная, которая может принимать значение любой из констант. Допускается указывать константы перечислимого типа непосредственно в разделе описания переменных без использования

раздела TYPE, например:

Var

avar : (winter, spring, summer, autumn);

Каждая из констант имеет порядковый номер, который начинается нуля. Так, winter имеет порядковый номер 0, а autumn - 3. Упорядоченность констант позволяет применять к ним операции отношений <, <=, =, <>, >=, >, а также стандартные функции:

ORD(avar) - для определения порядкового номера указанного элемента.

PRED(avar) - для определения элемента, являющегося предыдущим для указанного элемента.

SUCC(avar) - для определения элемента, являющегося следующим для указанного элемента

Интервальный тип позволяет задавать две константы, определяющие границы диапазона значений для каждой переменной. Обе константы должны принадлежать одному и тому же стандартному типу (кроме вещественных типов). Например,

Var S: 1..30;

Ch: 'a'..'f';

Указательный тип – их значениями являются адреса памяти. В отличие от стандартного указательного типа Pointer, пользовательский тип определяет множество значений, которые указывают на динамические

переменные определенного типа, называемого базовым типом. Указатель на какой-либо тип может быть описан до объявления самого типа:

```
Type PtStack = ^Stack;
     Stack = array [1..40] of real;
```

Процедурный тип позволяет объявлять переменные, которым допускается присваивание имен процедур, функций и методов, а также передавать такие переменные и имена в качестве параметров. Описание процедурных типов имеет такой же синтаксис, как и объявление процедур и функций:

```
Type Tproc1 = procedure (var x, y: real);
     Tproc2 = function (x: real): real;
```

Структурированные типы данных. Составные типы данных определяют упорядоченную совокупность скалярных переменных и характеризуются типом своих компонентов. К структурированным типам данных в Turbo Pascal относят: тип-массив (array), тип-множество (set), тип-запись (record), файловый тип (file), объектный тип (object), строковый тип (string);

Строковый тип. Строка – в общем случае это последовательность символов. Строка представляет собой особую форму одномерного массива символов, которая имеет существенное отличие. Массив символов имеет фиксированную длину (количество элементов), которая определяется при описании. Строка имеет две разновидности длины:

Общую длину строки, которая характеризует размер памяти, выделяемый строке при описании. Текущую длину строки (всегда меньше или равную общей длине), которая показывает количество смысловых символов строки в каждый конкретный момент времени. Для определения данных строкового типа в Turbo Pascal 7.0 введены стандартные типы String и PChar.

В строках типа String, текущая длина строки указывается в нулевом (то есть имеющем индекс 0) элементе строки. Максимальная текущая длина строки может быть не более 255 символов. Например,

```
Const Adres = 'ул. Королева, 2';
Type FileName = String [150];
Var St1: FileName;
    St2, St3: String [10];
```

В Паскале существует ряд встроенных процедур и функций для работы со строками типа String: Delete, Str, Val, Insert, Copy, Concat, Length, Pos, UpCase.

Тип PChar поддерживает формат представления строк, признаком конца строки которых служит символ с кодом 0 (нуль) и которые называются строками с завершающим нулем, или ASCIIZ-строками.

Тип-массив (array). Массив – это структурированный тип данных, состоящий из фиксированного числа элементов, имеющих один и тот же тип. Элементами массива могут быть данные любого типа, включая структурированный тип. Описывается в виде:

Type Имя = array [тип индекса] of тип компонентов;

При выполнении программы к каждому элементу массива можно обратиться, используя его индекс в массиве. Например, M = array [1..4] of integer; – массив из четырех целых чисел. M[1], M[2], M[3], M[4] – элементы массива M.

Тип-множество (set). Множество – набор взаимосвязанных по какому-либо признаку (или группе признаков) элементов, которые можно рассматривать как единое целое. Элементы множества должны принадлежать к одному и тому же типу данных, которые называют базовым типом множества. Type Имя = set of "элемент1, … элементN"; Например, A = set of (3, 5, 7, 11, 13) – множество простых чисел.

Тип запись (record). Запись – тип данных, состоящий из фиксированного числа компонентов одного или нескольких типов. Определение типа записи начинается идентификатором record и заканчивается зарезервированным словом end. Между ними заключен список компонентов, называемых полями, с указанием идентификаторов полей и типа каждого поля:

```
Type имя = record
        имя поля1: тип компонента;
        имя поля2: тип компонента;
        …
        End;
```

Например,

```
Mash = record
        Nomer: Integer;
        Marka: String [20];
        Year: Integer;
       End;
```

Файловый тип (file). Файлы классифицируются по двум признакам: по типу (логическая структура) и по методу доступа к элементам файла.

В Паскале существует три класса файлов по типу – типизированные, текстовые, нетипизированные. По методу доступа они делятся на файлы последовательного доступа и прямого доступа.

К типизированным файлам относятся файлы строго определенного типа (их иногда рассматривают как последовательность записей определенного типа). Стандартное задание в программе такой файловой переменной осуществляется следующим образом:

```
Type <RecordName> = Record
            …
           End;
```

Var <ИмяПеременной>: File of <RecordName>;

Текстовый файл рассматривается как последовательность символов, разбитая на строки. Каждая строка завершается символом конца строки. На практике это два символа: перевод строки Chr(13) и возврат каретки Chr(10).

Нетипизированный файл рассматривается в Pascal как набор символов или байтов. Представление Char или Byte не играет роли, а важно лишь с точки зрения объема занимаемого данными. Нетипизированный файл является файлом прямого доступа, что говорит о возможности одновременного использования операций чтения записи. При объявлении нетипизированного файла указывается только ключевое слово file:

Var F: file;

Объектный тип (object). Обьект – тип данных ставший основой обьектно-ориентированного программирования на языке Паскаль. Он похож на запись, так как содержит данные различных типов в качестве полей, и отличается тем, что также содержит методы работы с этими данными в рамках объекта. Методы работы с данными объекта называются правилами. Например,

```
Type Coordinate = object;                  {объект с именем Coordinate}
            X, Y: byte;                    {X,Y – данные, входящие в
                                            объект}
     Procedure Init(Xinit, Yinit: byte);   {метод инициализации
                                            данных}
     Function GetX: byte;                  {метод получение значения
                                            X}
     Function GetY: byte;                  {метод получение значения
                                            Y}
End;
```

Операции. Для арифметических операций используют обычные знаки +, –, *, / –деление, ^ – возведение в степень, div – целочисленное деление, mod – остаток от деления. Операторы отношений: < – больше, > – меньше, <= – больше или равно, >= – меньше или равно, <> – неравно.

Логические отношения: AND (и), OR (или), NOT, XOR (логическое исключающее или).

Конкатенация – '+' (слияние двух строк).

Операции над множествами: + – объединение, – разность, * пересечение, in- принадлежность, <= является подмножеством, >= является надмножеством, <> неравно.

Операторы. Составной оператор представляет собой группу из произвольного числа операторов, отделенных друг от друга ";" и ограниченную операторными скобками Begin и End. Например,

```
Begin
    Read (B);
    A := B + 0.75;
    Write (A);
End;
```

Функции ввода / вывода.

Для ввода и вывода данных используются стандартные процедуры ввода и вывода Read и Write, оперирующие стандартными последовательными файлами INPUT и OUTPUT.

Эти файлы разбиваются на строки переменной длины, отделяемые друг от друга признаком конца строки. Конец строки задается нажатием клавиши ENTER.

Для ввода исходных данных используются операторы процедур ввода:

```
Read(A1,A2,...AK);
ReadLn(A1,A2,...AK);
```

```
ReadLn;
```

Первый из них реализует чтение К значений исходных данных и присваивание этих значений переменным А1, А2, ..., АК. Второй оператор реализует чтение К значений исходных данных, пропуск остальных значений до начала следующей строки, присваивание считанных значений переменным А1, А2, ..., АК. Третий оператор реализует пропуск строки исходных данных.

Операторы ввода при чтении значений переменных целого и действительного типа пропускает пробелы, предшествующие числу. В то же время эти операторы не пропускают пробелов, предшествующих значениям символьных переменных, так как пробелы являются равноправными символами строк. Пример записи операторов ввода:

```
var A, B: Real;
C, D: Integer;
E, F: Char;
.....
Read(A,B, C, D);
Read(E, F);
```

Значения исходных данных могут отделяться друг от друга пробелами и нажатием клавиш табуляции и Enter. Для вывода результатов работы программы на экран используются операторы:

```
Write(A1,A2,...AK);
WriteLn(A1,A2,...AK);
WriteLn;
```

Первый из этих операторов реализует вывод значений переменных А1,

A2,...,AK в строку экрана. Второй оператор реализует вывод значений переменных A1, A2, ..., AK и переход к началу следующей строки. Третий оператор реализует пропуск строки и переход к началу следующей строки.

Структура программы в Паскале выглядит примерно так:

Program <имя>(input,output); -- заголовок

Uses <имя1,имя2,..>; – список подключаемых библиотечных модулей

Label: <список меток>; – раздел описания меток

Const : <определение констант>; – раздел описания констант

Type : <объявление типов>; – раздел описания типов данных

Var : <объявление переменных>; – раздел описания переменных

Procedure <имя процедуры> (<список параметров и из типы>); – описание процедур и запись их операторов

<основная часть процедуры>;

End;

Function <имя> (<список параметров и из типы>): <тип возвращаемого значения>; – описание функций

<основная часть функции>;

End;

Begin

<основная часть программы >

End.

Пример линейного алгоритма

```
{рассчитать выражение y = x*exp(-x) / (cos(y)*sin(y))
при заданных x и y}
program lin1;
var
```

```
  x, y:real;
begin

  s:=0;
  for n:=1 to NN do
  begin
    Write(',ўҐ¤ЁвҐ a[', n, ']:');
    Readln(a[n]);
    s:=s+a[n];
  end;

  Writeln('ђҐ§г«мв в: ', s);

end.
```

Варианты заданий

1. Даны два числа a и b. Получить их сумму, разность и произведение.

2. Даны действительные числа x и y. Получить (|x| – |y|) / (1 + |x·y|).

3. Дана длина ребра куба. Найти площадь грани, площадь полной поверхности и объем этого куба.

4. Даны два действительных положительных числа. Найти среднее арифметическое и среднее геометрическое этих чисел.

5. Даны катеты прямоугольного треугольника. Найти его гипотенузу и площадь.

6. Определить периметр правильного n-угольника, описанного около окружности радиуса r.

7. Даны x, y, z. Вычислить a, b, если

а) $a = \frac{\sqrt{|x-1|} - \sqrt{|y|}}{1 + x^2/2 + x^2/4}, \; b = x(arctg(z) + e^{-(x+3)})$;

б) $a = \dfrac{3 + e^{y-1}}{1 + x^2 \left| y - tg(z) \right|}, b = 1 + \left| y - x \right| + \dfrac{(y-x)^2}{2} + \dfrac{\left| y - x \right|^3}{3}$;

в) $a = (1+y)\dfrac{x + y/(x^2+4)}{e^{-x-2} + 1/(x^2+4)}, b = \dfrac{1 + \cos(y-2)}{x^4/2 + \sin^2 z}$;

г) $a = y + \dfrac{x}{y^2 + \left| \dfrac{x^2}{y + x^3/3} \right|}, b = \left(1 + tg^2 \dfrac{z}{2}\right)$;

д) $a = \dfrac{2\cos(x - \pi/6)}{1/2 + \sin^2 y}, b = 1 + \dfrac{z^2}{3 + z^2/5}$;

е) $a = \dfrac{1 + \sin^2(x+y)}{2 + \left| x - 2x/(1 + x^2 y^2) \right|} + x, b = \cos^2\left(arctg \dfrac{1}{z}\right)$;

ж) $a = \ln\left| \left(y - \sqrt{|x|}\right)\left(x - \dfrac{y}{z + x^2/4}\right) \right|, b = x - \dfrac{x^2}{3!} + \dfrac{x^3}{4!}$.

8. Дана сторона равностороннего треугольника. Найти площадь этого треугольника.

9. Известна длина окружности. Найти площадь круга, ограниченного этой окружностью.

10. Найти площадь кольца, внутренний радиус которого равен 20, а внешний – заданному числу r (r>20).

11. Найти площадь равнобочной трапеции с основаниями a и b и углом α при большем основании a.

12. Вычислить расстояние между двумя точками с координатами x_1, y_1 и x_2, y_2.

13. Треугольник задан координатами своих вершин. Найти:

а) периметр треугольника; б) площадь треугольника.

14. Найти площадь сектора, радиус которого равен 3,7, а дуга содержит заданное число радиан φ.

15. Дано действительное число a. Не пользуясь никакими другими арифметическими операциями, кроме умножения, получить: а) a^4 за две операции; б) a^6 за три операции; в) a^7 за четыре операции; г) a^8 за три

операции; д) a^9 за четыре операции; е) a^{10} за четыре операции; ж) a^{13} за пять операций; з) a^{15} за пять операций (указание:$a^{15}=(a^5)^3$); и) a^{21} за шесть операций; к) a^{28} за шесть операций; л) a^{64} за шесть операций.

16. Дано действительное число a. Не пользуясь никакими другими арифметическими операциями, кроме умножения, получить: а) a^3 и a^{10} за четыре операции; б) a^4 и a^{20} за пять операций; в) a^5 и a^{13} за пять операций; г) a^5 и a^{19} за шесть операций; д) a^2 и a^5, a^{17} за шесть операций; е) a^4 ,a^{12}, a^{28} за шесть операций.

1.2. Конструкции условия и выбора, разветвляющиеся алгоритмы.

Условный оператор IF...THEN....ELSE (Если...то...иначе) позволяет организовать разветвление в алгоритме программы. Условный оператор записывается следующим образом:

IF <ЛогическоеУсловие> THEN <Оператор1>

ELSE <Оператор2>;

Если логическое условие выполняется, то выполняется оператор 1. В противном случае выполняется оператор 2. Например:

```
program Moscow;
var k,y:integer;
begin
writeln('Знаете ли Вы год основания Москвы?');
write('Введите год основания Москвы: k=');
read(k);
if k=1147 then writeln('Вы совершенно правы!')
        else writeln('Вы ошиблись!');
writeln('Для выхода из програмы введите число 1.');
readln(y);
end.
```

Если разветвляющийся алгоритм более сложный, например при выполнении условия необходимо выполнить несколько операторов, то операторы объединяются в конструкцию begin … end; Условные конструкции могут быть вложены, например:

```
If k=10 then
        Begin
                N=20;
                R=10;
        End;
Else
        if k=12 then
                N=5;
        Else
                Begin
                        N=15;
                        R=9;
                End;
```

Конструкция несколько громоздка. Для более простой записи можно использовать конструкцию выбора. Эта же конструкция с применение CASE выглядит так:

```
Case k Of
        10 : Begin
                N=20;
                R=10;
                End;
        12 : N=5
        Else
```

```
        Begin
            N=15;
            R=9;
        End;
    End;
```

В элементе списка выбора можно использовать несколько констант выбора, а также диапазоны:

```
    CASE X OF
        'A', 'C', 'E', 'G' : WriteLn( 'Указано несколько констант');
        'K' . . 'R' : WriteLn( 'Указан интервал')
    END;
```

Пример программы

Вычислить и вывести на экран корень квадратный из положительного числа X, значение которого не превышает 1000. В случае отрицательного числа никаких вычислений в программе не производится.

```
Program KOR(input, output);
VAR {описание действительных переменных X и Y }
    X, Y : real;
Begin
    Readln( X ); { ввод переменной X }
    If X >= 0 Then
        Begin { составной оператор }
            Y :=SQRT ( X );
            WriteLn( ' Корень из X равен ', Y : 6:2)
        End;
```

End.

Варианты заданий

1. Даны действительные числа x, y. Получить: а) max(x,y); б) min(x,y); в) max(x,y), min(x,y).

2. Даны действительные числа x, y, z. Получить: а) max(x,y,z); б) min(x,y,z); в) max(x,y,z), min(x,y,z).

3. Даны действительные числа x, y, z. Вычислить: а) max2(x+y+z,xyz); б) min(x+y+z/2,xyz)+1; в) min(x+5y+z,2x-y+z) + max(x+y+z,xy+z).

4. Даны действительные числа a, b, c. Удвоить эти числа, если a > b > c, и заменить их абсолютными значениями, если это не так.

5. Даны действительные числа x, y. Вычислить z:

$$y = \begin{cases} x - y, \text{если } x > y, \\ y - x + 1, \text{в противном случае.} \end{cases}$$

6. Даны два действительных числа. Вывести первое число, если оно больше второго, и оба числа, если это не так.

7. Даны два действительных числа. Заменить первое число нулем, если оно меньше или равно второму, и оставить без изменения в противном случае.

8. Даны действительные числа x, y (x ¹ y). Меньшее из этих двух чисел заменить их полусуммой, а большее – их удвоенным произведением.

9. Даны действительные числа x, y, z. Выяснить, существует ли треугольник с длинами сторон x, y, z.

10. Даны действительные числа a, b, c (a ¹ 0). Выяснить, имеет ли уравнение $ax^2 + bx + c = 0$ действительные корни. Если действительные корни имеются, то найти их. В противном случае ответом должно служить сообщение, что действительных корней нет.

11. Дано действительное число a. Вычислить f(a), если

а) $$f(x)=\begin{cases} x^2, при\ -2 \le x < 2, \\ 4, в\ противном\ случае; \end{cases}$$

б) $$f(x)=\begin{cases} x^2+4x+5, при\ x \le 2, \\ \dfrac{1}{x^2+4x+5}, в\ противном\ случае, \end{cases}$$

в) $$f(x)=\begin{cases} 0, при\ x \le 0, \\ x, при\ 0 < x \le 1, \\ x^2, в\ противном\ случае; \end{cases}$$

в) $$f(x)=\begin{cases} 0, при\ x \le -5, \\ x^2 - x, при\ -5 < x \le 5, \\ x^2 - \sin x^2, в\ противном\ случае, \end{cases}$$

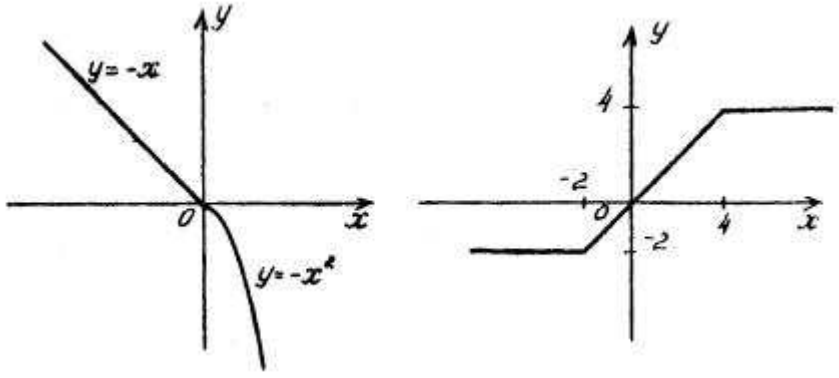
12. Дано действительное число a. Для функций f(x), графики которых представлены на рис. 1, вычислить f(a).

13. Даны действительные числа x, y. Определить, принадлежит ли точка с координатами x, y заштрихованной части плоскости (рис. 2).

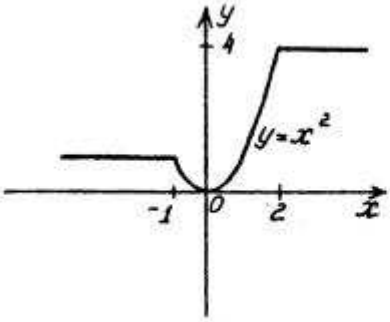
14. Дано натуральное число n (n < 100), определяющее возраст человека (в годах). Дать для этого числа наименования "год", "года" или "лет": например, год, 23 года,5 лет и т.д.

15. Дата вводится в формате "дд.мм.гг". Записать словесно название месяца и полностью указать год. Например, для 10.02.96 вывести 10 февраля 1998 года.

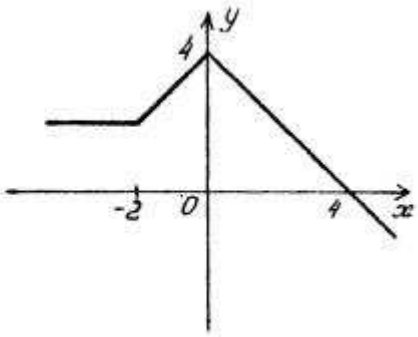
16. Дано натуральное число n. Определить является ли число четным.



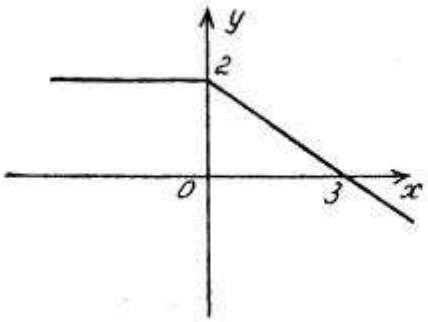
а



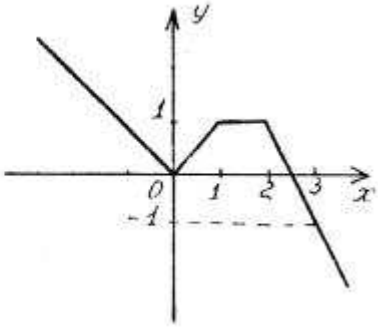
б



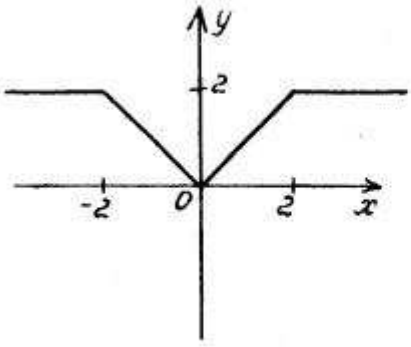
в



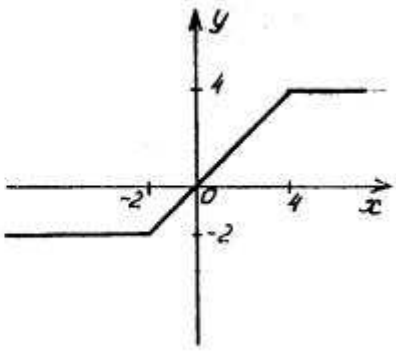
г



д

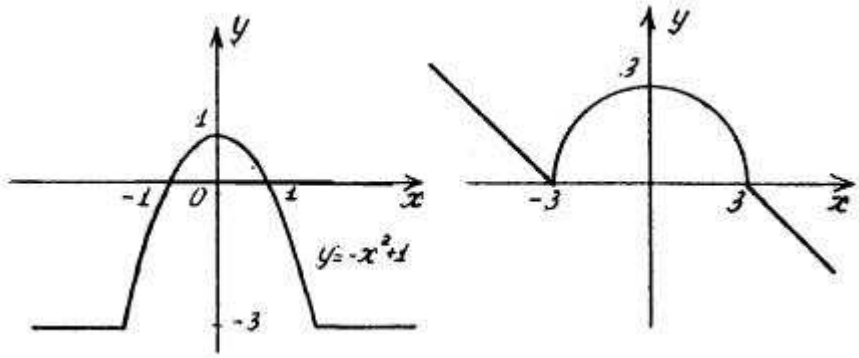


е

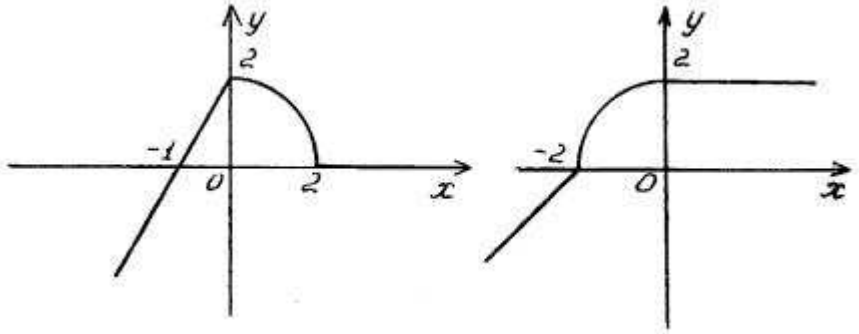


ж

з

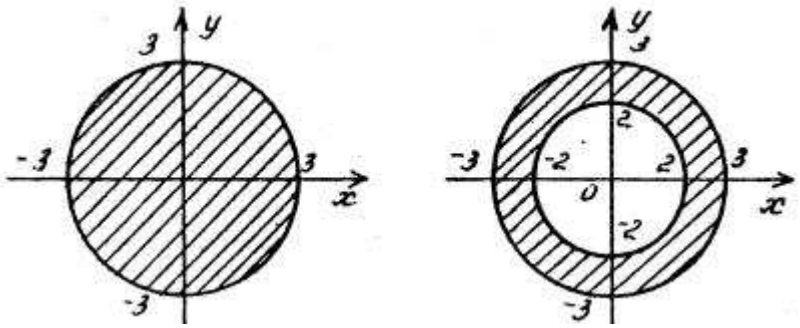


и к

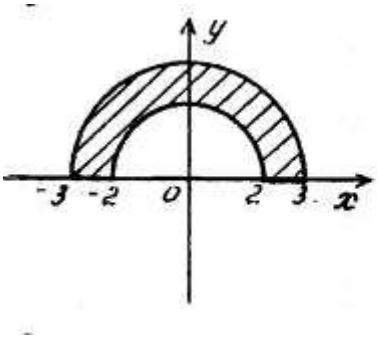


л м

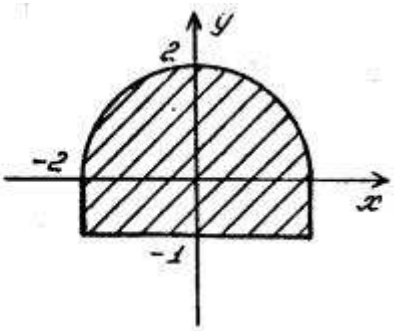
Рис. 1



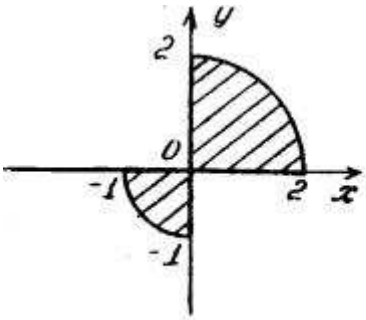
а б



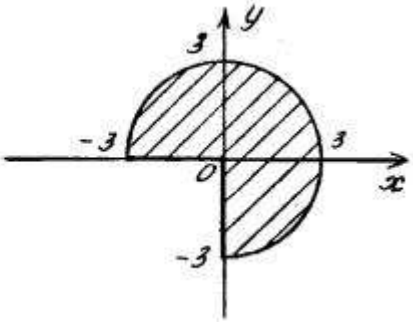
в



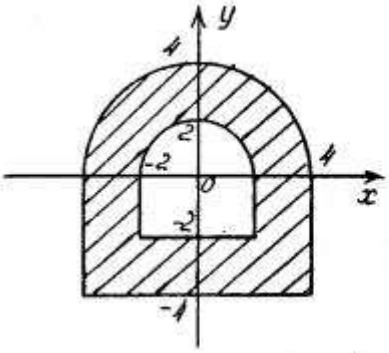
г



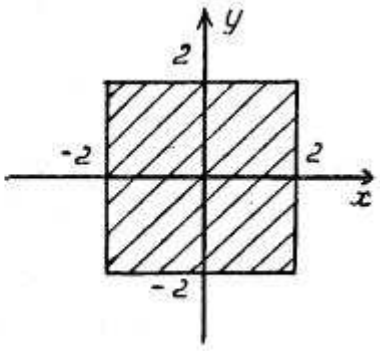
д



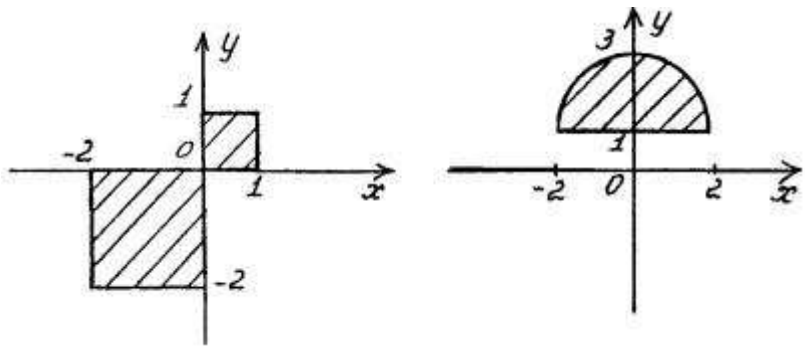
е



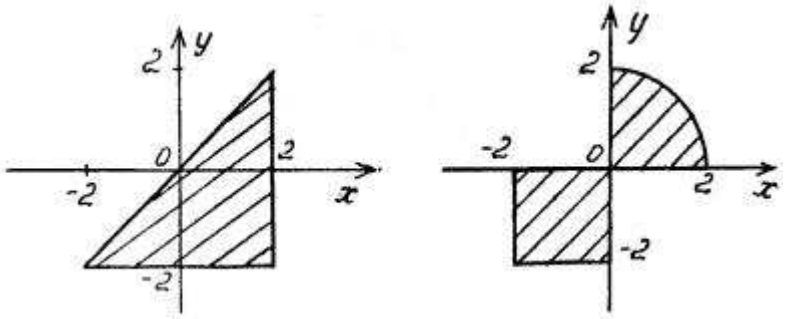
ж



з



и к



л м

Рис. 2

17. Сравнить между собой значение величины x и y. Вывести на печать результат сравнения в виде "x > y", "x = y" или "x < y".

18. Определить, какая из двух точек T1(x1,y1) или T2(x2,y2) расположена ближе к началу координат. Вывести на печать координаты этой точки.

19. Определить, какая из двух фигур - круг или квадрат имеет большую площадь. Известно, что сторона квадрата равна a, радиус круга r. Вывести на печать название и значение площади большей фигуры.

1.3. Циклы, оператор перехода, циклические алгоритмы.

Основным процессом программы, как правило, является цикл.

Например, когда идет обработка действий пользователя (щелчка мыши или нажатия клавиши), после выполнения заданных действий программа возвращается в исходное состояние – ожидание следующего действия, и процесс повторяется. Обработка многих процессов – сигналы датчиков, прием/передача информации также являются циклами, т.е. повторяются многократно. В программе часто выделяют основной цикл, в который уже включены остальные. Однако любой цикл бывает с постусловием а) или предусловием б).

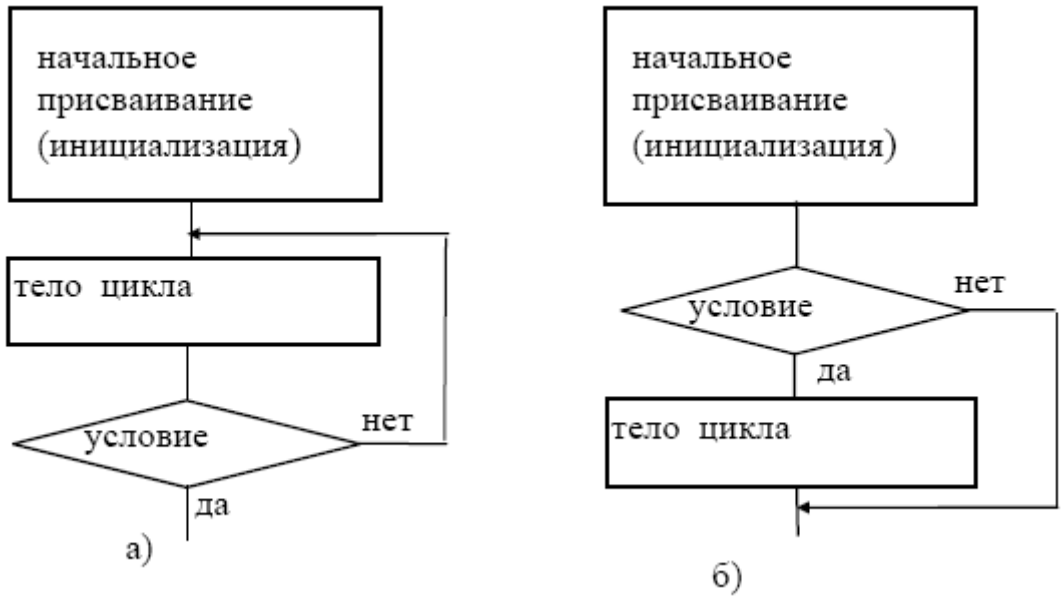


Рис. 3.

К циклам с предусловием относятся конструкции FOR…NEXT и WHILE … DO, к циклам с постусловием REPEAT … UNTIL.

Синтаксис оператора WHILE:

WHILE < выражение> DO <оператор>;

или

WHILE < выражение> DO

BEGIN

<оператор >;

…

<оператор >;

```
END;
```

Цикл будет выполняться до тех пор, пока <выражение> будет принимать значение Истина. Например, посчитаем сумму всех степеней двойки пока суммируемое значение не превысит 1000.

```
X:=2; S:=0;
WHILE X<1000 DO BEGIN
    S:=S+X;
    X:=X*2;
END;
WriteLn(S);
```

Синтаксис оператора цикла с "постусловием":

```
REPEAT
    < операторы тела цикла >
UNTIL < выражение>;
```

Цикл будет выполняться до тех пор, пока <выражение> не примет значение Истина. Цикл с постусловием всегда выполняется хотя бы один раз. Перепишем пример выше с применением цикла REPEAT:

```
X:=2; S:=0;
REPEAT
    S:=S+X;
    X:=X*2;
UNTIL X>=1000;
WriteLn(S);
```

Заметим, что в данном случае сумма не может быть меньше 2.

Синтаксис оператора цикла с параметром:

```
FOR i:= N TO K DO
        <оператор >;
или
FOR i:= N TO K DO
BEGIN
        <оператор >;
        …
        <оператор >;

END
```

Например FOR i:= 1 TO 5 DO

```
        WriteLn(i);
```

Цикл выполнится 5 раз и на экране мы увидим цифры 1, 2, 3, ..5.

Вместо выражения инкремента TO может быть использовано выражение декремента DOWNTO, например:

```
FOR i:= 10 DOWNTO 1 DO
        <оператор >;
```

Цикл повторится 10 раз, переменная i будет принимать последовательно значения 10, 9, …, 1.

Вложенные циклы

Циклы могут быть вложены один в другой. Иногда их называют сложными циклами. При использовании вложенных циклов необходимо составлять программу таким образом, чтобы внутренний цикл полностью входил в тело внешнего цикла. Внутренний цикл может, в свою очередь,

иметь другой внутренний цикл (циклы). Структуру вложенных циклов рассмотрим на примере.

Пример. Вычислить значение функции Y = 2L + M при всех значениях переменных M=1, 2, 3 и L=2, 4, 6, 8. Программа должна содержать два вложенных друг в друга цикла. В качестве параметра внешнего цикла удобно использовать переменнуюM, а внутреннего - переменную L. Тогда при первом значении M переменная L будет принимать значения 2, 4, 6, 8. При следующем значении M переменная L также будет принимать значения 2, 4, 6, 8 и так до конца.

Программа имеет вид:

```
Program Circles;
Var
    M, L, Y : integer;
Begin
    For M := 1 to 3 do
        Begin { начало тела внешнего цикла }
        L := 2;
        While L <= 8 do
            Begin { начало тела внутреннего цикла }
            Y := 2 * L + M;
            Writeln('M = ', M:4, ' L = ', L:4, ' Y = ', Y:4);
            L := L + 2
            End { конец тела внутренного цикла }
        End { конец тела внешнего цикла }
End.
```

Метка. Оператор перехода. Пустой оператор.

Каждый оператор в программе может быть помечен меткой – произвольным идентификатором. Метка позволяет именовать некоторый

оператор программы и таким образом ссылаться на него. Метка располагается перед оператором и отделяется от него двоеточием.

Например:

10: ReadLn(' Введи значение переменной A:', A);

124: Y := X * X + S * Z;

Метки не влияют на выполнение оператора. Они должны быть описаны в разделе

описания меток. Описание меток состоит из ключевого слова LABEL и следующего за ним

списка меток.

Например:

LABEL 10, 124, 540, L1, L2;

Описания меток располагаются до совокупности всех описаний переменных. Описанной меткой должен быть помечен только один оператор программы. Оператор перехода прерывает естественный порядок выполнения операторов. Он состоит из ключевого слова GOTO, за которым следует метка. Дальнейшее выполнение программы должно продолжаться, начиная с оператора, помеченного указанной меткой.

В языке Паскаль существует довольно строгая дисциплина использования операторов перехода. Сформулируем эти ограничения:

1) с помощью оператора перехода нельзя войти внутрь составного оператора, но внутри составного оператора разрешены любые передачи управления;

2) с помощью оператора перехода нельзя войти внутрь тела цикла, минуя его заголовок. Внутри тела цикла разрешены любые передачи управления;

3) с помощью оператора перехода нельзя войти ни в одну из ветвей условного оператора, а также передать управление из одной ветви в другую;

4) с помощью оператора перехода нельзя войти внутрь оператора выбора или передать управление на другую константу выбора;

5) с помощью оператора перехода нельзя войти в тело процедуры;

6) с помощью оператора перехода можно выйти из любой конструкции языка программирования за единственным исключением: если в программе содержатся многократно вложенные друг в друга процедуры, то из любой внутренней процедуры можно выйти с помощью оператора перехода, ведущего к метке, которой помечен оператор 29 на самом внешнем уровне вложенности блоков (то есть выйти можно только в главную программу).

Операторы выхода. Для завершения работы программ, процедур и функций без предварительного перехода по меткам к закрывающему end в TURBO PASCAL введены процедуры Exit и Halt.

Вызов Exit завершает работу своего программного блока и передает управление вызывающей программе. Если Exit выполняется в подпрограмме, то выполнение этой подпрограммы прекратится, и далее будет выполняться следующий за вызовом этой подпрограммы оператор. Если Exit выполняется в основной программе, выход из нее будет эквивалентен ее нормальному завершению.

Вызов процедуры Halt, где бы она не находилась, завершает работу программы и передает управление операционной системе. Процедура Halt имеет структуру Halt(n), где n - код возврата, который может быть проанализирован операционной системой с помощью команды IF ERRORLEVEL. Значение n=0 соответствует нормальному завершению работы программы. Вызов процедуры Halt без параметра эквивалентен вызову Halt(0).

Варианты заданий.

1. а). Высчитать какова максимальная непрерывная серия выпадения герба или последовательности орёл-решка. Найти количество выпадения последовательностей меньшей длины.

б). Программа расставляет на шахматной доске 8 ферзей так, чтобы они не били друг друга.

2. а). Провести несколько серий испытаний из десяти подбрасываний монетки. Высчитать, возможен ли вариант, что в одной серии выпадает десять орлов подряд и сравнить с частотой выпадения последовательности орёл-решка.

б). В урне находится N белых и K черных шариков. Вытаскиваем шарик, смотрим на него, запоминаем цвет и бросаем обратно. Найти вероятность попадания шарика одного и другого цвета.

3. Монетка подбрасывается вверх. Исход испытания - орёл или решка записывается, а потом рассчитывается вероятность выпадения того и другого. Программа работает как в ручном, так и в автономном режиме.

4. Вытаскивание из урны, содержащей N белых шаров и K черных, двух шаров без возврата. Найти вероятность попадания шариков только белого цвета, только черного, черного и белого, белого и черного.

5. Вывести квадраты простых чисел от 1 до N.

6. Кинетическая энергия движущегося тела $W = mV^2/2$, где m - масса тела, V - его скорость. Получить зависимость W от m при значениях V, изменяющихся от V1 до V2 с шагом h1. Масса m изменяется от m1 до m2 с шагом h2.

7. При сооружении железобетонного фундамента площадь сечения арматуры определяется по формулам:

$F = (N - RS)/(D - R)$, если $F > 0.03S$;

$F = (N - RS)/D$, если $F \leq 0.03S$,

где N - расчетная сила, приложенная по оси элемента; S - площадь сечения бетона; R - сопротивление бетона осевому сжатию; D - сопротивление сжатой арматуры. Рассчитать значения переменной N, если известны R, S и D, а F изменяется от F1 до F2 с шагом h.

8. Вычислить значения функции Y = 10 SinDX / (1+D2 X2), если X изменяется от 0.1 до 10 с шагом 0.13, а D - от 1.2 до 5.4 с шагом 1.1.

9. Объем цилиндрической подковы вычисляется по формуле:

$$V = \frac{\pi}{3b} \cdot \left[a(3r^2 - a^2) + 3r^2(b - r)\frac{f \cdot \pi}{180} \right]$$

Вычислить значения объема V, который зависит от угла f, если a, b, h и r известны, а f изменяется от f1 до f2 с шагом h.

10. Температура T молока в бидонах после содержания их на открытом воздухе и перевозки в крытом брезентом автомобиле в течение W часов выражается формулой:

$$T = T0 + (T1 - T0)e^{\frac{-KSW}{V}}$$

где T1 - начальная температура молока, °C; T0 - температура окружающего воздуха, °C; S - площадь поверхности бидона, м2; V - объем бидона, м; K = 0.00448 м/г - постоянный коэффициент. Вычислить значения температуры молока после хранения и перевозки с выдачей результатов для W, изменяющегося от 1 до 10 часов с интервалом 1 час.

11. Производительность станка для резки бетонно-мозаичных плит составляет A см^2/ч. После того, как на станке нарезали плиты общей площадью B см^2, станок останавливают на T1 мин., после чего вновь нарезают B см^2 плит с последующим перерывом на T1 минут и т.д. Вычислить значение площади нарезанных плит S см^2 за общее время работы T2 часов.

12. Вычислить значения функции Z = X· b/(X + b), если X изменяется от начального значения a с шагом h.

13. Вычислить значения функции $Z =$, где X изменяется от a до b с шагом h. Известно, что c > 0.

14. Вычислить значения функции Y = nSinX - CosnX, если X изменяется от X0 до XK с шагом h.

15. Вычислить значения функции Y = SinX, если X изменяется от X0 до XK с шагом h.

1.4. Функции, процедуры, передача параметров по ссылке и значению, рекурсия.

Процедуры и функции используются для придания иерархической структуры программе, для облегчения понимания программы, для выделения повторяющихся частей в отдельные блоки. Процедуры и функции аналогичны программам и имеют общее название- подпрограммы. Взаимодействие между процедурами/функциями и их вызовом из программы как правило осуществляется через параметры вызова. Существуют и другие способы передачи данных в подпрограммы, например, через глобальные переменные. Отличие между функцией и процедурой заключается в том, что функция возвращает значение некоторого типа, а процедура не возвращает значений. Поэтому обращение к функции выполняется внутри выражений, а обращение к процедуре всегда является оператором.

В программе, написанной на языке Паскаль, описание процедур и функций должно располагаться в разделе описаний. Каждая процедура или функция определяется только один раз, но может использоваться многократно. Структура процедур и функций аналогична структуре основной программы на языке Паскаль. В процедурах и функциях могут быть описаны собственные константы, типы, локальные переменные, локальные процедуры

и функции. Последовательность разделов в процедуре/функции та же, что и в основной программе.

Синтаксис описания процедуры имеет вид:

```
Procedure <имя процедуры> (<формальные параметры>);
        <раздел описаний>
Begin
        <операторы>
End;
```

Примеры заголовков процедуры.

```
        Procedure simple;
```

или

```
        Procedure hard(x, y: integer; z : real);
```

Здесь simple и hard - названия процедур. Первый пример - это заголовок процедуры без параметров, второй- с параметрами. Тело процедуры, заключается в блок begin ... end и заканчивающееся точкой с запятой. Все описания процедур идут после описаний переменных. В процедуре могут содержаться обращения к ранее описанным процедурам. Если описание вызываемой процедуры идет по тексту позже, то перед использованием процедуры необходимо сделать ее объявление. Это делается строкой заголовка с ключевым словом FORWARD. Например:

```
Procedure hard(x, y: integer; z : real) FORWARD;
```

В области описания можно объявлять локальные переменные и метки. Переменная, описанная в процедуре, называется локальной, т.е. местной. Ее «видимость» определена только в рамках текущей процедуры / функции. Если вне процедуры уже описана глобальная переменная с тем же именем p,

то это две разных переменных, различающихся временем жизни и областью видимости. Так глобальная переменная существует все время, пока выполняется программа, а локальная переменная существует кратковременно пока работает процедура. Область видимости глобальной переменной – вся программы, кроме тех процедур и функций, в которых описана локальная переменная с тем же именем. В то же время локальная переменная процедуры видна только в той процедуре, в которой она описана, и разные процедуры могут иметь различные локальные переменные, используя одно и то же имя.

Например, в программе вы можете объявить переменную X и присвоить ей значение 5. В функции sqrt2 вы можете использовать имя X для переменной параметра, значение этой переменной будет определяться в момент вызова этой функции. Более того определив локальную переменную X в функции sum2 вы сможете ее использовать, например для вычислений в цикле. Это 3 различных переменных, для которых будет выделено 3 различных участка памяти.

Аналогично, в процедуре могут описываться и использоваться локальные метки и типы.

Отметим что в процедурах могут быть использованы передача параметров по значению и по ссылке. В первом случае Паскаль будет копировать значение переменной для передачи ее функции или процедуры в отдельную память (обычно ее выделяют в области стека, специально области используемой для вызов процедур и функций а также хранения локальных параметров). В этом случае при возвращении из процедуры или функции передаваемое значение изменено не будет. Во втором случае будет передаваться ссылка на значение. Различие двух этих способов передачи рассмотрим на примере.

Синтаксис описания функции:

function <имя функции> (<формальные параметры>):<тип результата>;

```
    <раздел описаний>
begin
    <операторы>
end;
```

Пример: Пусть требуется составить программу вычисления площади выпуклого четырехугольника, заданного длинами четырех сторон и диагонали. Диагональ делит выпуклый четырехугольник на два треугольника, к которым применима формула Герона $S = \sqrt{p(p-a)(p-b)}$, где a , b , c - длины сторон треугольника, p=(a+b+c)/2.

Опишем процедуру вычисления площади треугольника, а затем будем обращаться к ней в программе.

```
Program UseHeron(input, output);
    var ab, bc, cd, da, ac, s1, s2 : real;
procedure heron(a, b, c : real; var s : real);
    var p : real;
begin
    p := (a+b+c)/2;
    s := sqrt(p*(p-a)*(p-b)*(p-c));
end;
begin
    read(ab, bc, cd, da, ac);
    heron(ab, bc, ac, s1);
    heron(cd, da, ac, s2);
    write(s1+s2) ;
end .
```

Обратите внимание, что параметр s описан в процедуре heron с

модификатором var. Это означает, что параметр передается по ссылке. При передаче переменной по ссылке передается не само значение переменной а ссылка на участок памяти, где хранится передаваемое значение. Поэтому при изменении переменной внутри процедуры, передаваемое значение по указанному адресу тоже изменится. В нашем случае вычисленный результат формулы Герона будет помещен в переменные s1 и s2. Однако язык не запрещает использование в качестве такого параметра вычисляемое значение, например heron(ab, bc, ac, s1+s2), но в этом случае мы не получим результата. Дело в том что при вычислении выражений Паскаль помещает временно полученное значение в область стека, затем передаст ссылку на эту область процедуре heron, которая поместит в нее вычисленный результат. Но так как эта область памяти не принадлежит ни одной из переменной, то мы не сможем этот результат использовать. Передача параметров по ссылке широко используется в случаях, когда нужно вернуть не одно, а более значений или когда нужно передать процедуре или функции большой объем данных, например массив или структуру. Если возвращаемое значение одно и не велико, то логично использовать функцию. Перепишем вышеприведенный пример с использованием функций.

```
Program UseHeron(input, output);
        var ab, bc, cd, da, ac, s1, s2 : real;
function heron(a, b, c : real): real;
        var p : real;
begin
        p := (a+b+c)/2;
        heron := sqrt(p*(p-a)*(p-b)*(p-c));
end;
begin
        read(ab, bc, cd, da, ac);
        s1:=heron(ab, bc, ac);
```

```
        s2:=heron(cd, da, ac);
        write(s1+s2) ;
end .
```

Рекурсия – вызов процедуры или функции самой себя. Разберем рекурсию на примере.

Выдать на печать в обратном порядке цифры целого положительного числа N. Составим процедуру REVERSE.

```
PROCEDURE REVERSE (N: integer);
Begin
        Write (N mod 10);
        If (N Div 10)<>0 Then REVERSE (N Div 10)
END;
```

Рассмотрим работу этой процедуры для числа N = 153.

- Оператор Write(N mod 10) выведет на экран остаток от деления 153 на 10, то есть 3.
- Оператор IF (N DIV 10)<>0 Then REVERSE (N DIV 10) проверяет целую часть частного 153/10 = 15 на ноль. Если целая часть не ноль, то с этим значением (15) идет вновь обращение к процедуре REVERSE.
- Оператор Write (N MOD 10) выводит на экран остаток от деления 15 на 10, т.е.5; затем со значением 15/10 = 1 идет обращение к REVERSE.

После вывода цифры 1 оператором Write (N MOD 10) проба N DIV 10 на ноль передает управление на конец процедуры. На экране будет записано число 351.

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

Рекурсивные процедуры и функции (подпрограммы) имеют одну из двух форм: прямую рекурсию и косвенную рекурсию. В первом случае подпрограмма содержит оператор вызова этой же подпрограммы (как в примере с процедурой REVERSE). Во втором случае одна подпрограмма вызывает другую подпрограмму, которая либо сама, либо посредством других процедур или функций вызывает исходную подпрограмму.

Если A, B - имена подпрограмм, то схема вызова может быть такой: A-B-A. В случае косвенной рекурсии возникает проблема: как, и где описать вызываемую подпрограмму. По правилам языка Паскаль каждая подпрограмма должна быть описана до её вызова. Но если подпрограмма A вызывает B, а B вызывает A, то получается замкнутый круг. Для подобных ситуаций принято следующее правило: одна из рекурсивных подпрограмм, вызывающих друг друга, описывается предварительно следующим образом:

PROCEDURE P(<Список формальных параметров>); FORWARD;

Здесь P - имя процедуры, FORWARD - ключевое слово. Это описание указывает транслятору, что текст процедуры P помещен ниже. Подобным же образом описывается функция: к оператору FUNCTION добавляется слово FORWARD. Список формальных параметров и тип результата (для FUNCTION) включается только в это предварительное описание и опускается в заголовке соответствующей функции.

Варианты заданий.

1. Даны действительные числа x1, y1, x2, y2, …,x10, y10. Найти периметр десятиугольника, вершины которого имеют соответственно координаты (x1, y1), (x2, y2), …, (x10, y10). (Определить процедуру вычисления расстояния между двумя точками, заданными своими координатами.)

2. Даны действительные числа a, b, c, d, e - стороны пятиугольника. Найти площадь пятиугольника. (Определить процедуру вычисления площади треугольника по его сторонам.)

3. Написать программу вычисления P по формуле: где n - заданное натуральное число.

4. Описать функцию Stepen (x,n) от вещественного x и целого n, вычисляющую (посредством умножения) величину xn, и использовать ее для вычисления b=2.7k+(a+1)-5.

5. Даны отрезки a,b,c и d. Для каждой тройки этих отрезков, из которых можно построить треугольник, напечатать площадь данного треугольника. Определить процедуру Plo(x,y,z), печатающую площадь треугольника со сторонами x,y и z, если такой треугольник существует.

6. Пусть процедура Socr(a,b,p,q) от целых параметров a,b,p,q приводит дробь a/b к несократимому виду p/q. Описать данную процедуру и использовать ее для приведения дроби 1+1/2+1/3+…+1/20 к несократимому виду.

7. Даны длины a, b и c сторон некоторого треугольника. Найти медианы треугольника, сторонами которого являются медианы исходного треугольника. Длина медианы, проведенной к стороне a, равна $\frac{1}{2}\sqrt{2b^2+2c}$.

8. Даны координаты вершин двух треугольников. Определить, какой из них имеет большую площадь.

9. Даны координаты вершин треугольника и координаты некоторой точки внутри него. Найти расстояние от данной точки до ближайшей стороны треугольника. (При определении расстояний учесть, что площадь треугольника вычисляется и через три его стороны, и через основание и высоту.).

10. Три прямые на плоскости заданы уравнениями akx+bky=ck, k=1,2,3. Если эти прямые попарно пересекаются и образуют треугольник, тогда найти его площадь.

11. Два натуральных числа называются "дружественными", если каждое из них равно сумме всех делителей другого, за исключением его самого (таковы, например, числа 220 и 284). Напечатать все пары "дружественных" чисел, не превосходящих заданного натурального числа.

12. Дано четное число n > 2. Проверить для этого числа гипотезу Гольдбаха. Эта гипотеза (по сегодняшний день не опровергнутая и полностью не доказанная) заключается в том, что каждое четное n, большее двух, представляется в виде суммы двух простых чисел. Воспользоваться функцией распознавания простых чисел.

13. Дано натуральное число n. Выяснить, является ли оно полным квадратом. Определить функцию, позволяющую распознавать полные квадраты.

14. Дан массив A[1..50], элементы которого отличны от нуля. Расположить их в таком порядке, чтобы первыми были все положительные элементы, а затем - все отрицательные, причем порядок следования как положительных, так и отрицательных элементов должен сохраниться (при решении задачи новый массив не заводить!).

15. Преобразовать массив S, "поворачивая" его вокруг центра на 90, 180, 270 градусов против часовой стрелки.

16. Рассматривая массивы X, Y и Z как представление некоторых множеств из объектов типа индекс (X[k]=TRUE, если элемент k принадлежит множеству X, и X[k]=FALSE иначе, и т.п.), реализовать следующую операцию над этими массивами-множествами: переменной t присвоить значение TRUE, если множество X является подмножеством множества Y, и значение FALSE иначе.

17. Элемент двухмерного массива называется локальным минимумом, если он строго меньше всех имеющихся у него соседей. Подсчитать количество локальных минимумов заданной матрицы размером NxN найти максимум среди всех локальных минимумов.

18. Составить функцию для нахождения точного значения суммы натуральных чисел, в десятичной записи которых более 20 знаков. Указание. Исходные данные и ответ представить в виде массивов цифр.

19. Даны две строки символов. Символ будем называть общим, если он встречается и в первой, и во второй строке. Пусть K1 - число вхождений в первую строку общего символа, а K2 - во вторую. Минимальное из чисел K1, K2 будем называть числом общности. Вывести все общие символы с указанием для них числа общности.

20. Рассмотрим произвольное натуральное число и найдем сумму его цифр, затем сумму цифр полученного числа и так далее, пока не получим однозначное число. Назовем это число цифровым корнем. Требуется написать программу, которая для заданного N (N<10100) находит его цифровой корень.

21. Задано N натуральных чисел a1, a2,…, aN ($1 \le N \le 20$), каждое из которых находится в интервале от 1 до 10000. Необходимо определить количество натуральных делителей произведения a1*a2*…*aN.

22. Написать функцию поиска корней полинома степени n. Исходными параметрами будут числа a1, a2,…,an. Комплексные корни учитывать.

23. Требуется написать программу, которая выводит в порядке возрастания все правильные несократимые дроби, знаменатели которых не превосходят N (2<=N<=500).

24. На экране компьютера, работающего в операционной системе Windows, было открыто N (N <= 20) окон, положение каждого из которых однозначно определяется четверкой натуральных чисел - X1 Y1 X2 Y2 - координатами левого верхнего и правого нижнего угла окна. Очевидно, что окна, открытые позже, могут частично или полностью перекрывать открытые ранее. Окно считается видимым, если виден хотя бы один образующий его пиксел.

2. РАБОТА С ДАННЫМИ

2.1. Строки

Турбо-Паскаль предоставляет средства для работы с данными строкового типа. Строковый тип данных представляет собой цепочку символов. Длина цепочки может изменяться от 0 до 255. Для определения строкового типа используется служебное слово STRING, за которым в квадратных скобках указывается максимальная длина строки, например:

```
TYPE
    line = string[25];
VAR
    mline : line;
    ...
```

В данном примере переменная mline представляет собой последовательность, включающую до 25 символов, причем каждый символ имеет стандартный тип CHAR. Значение строковой переменной может быть назначено оператором присваивания, либо введено с устройства ввода, например:

```
aline := 'ВСТИ';
mline := aline;
readln(mline);
```

Изображение строки строится из цепочки символов и заключается в апострофы. Максимальная длина строки может быть задана целым числом, или константой целого типа. Указание максимальной длины может быть опущено; в этом случае подразумевается число 255, например:

```
TYPE
line = string;
line1 = string[255];
```

Описания типов в данном примере эквивалентны. Основное отличие строк от символьных массивов заключается в том, что строки могут динамически изменять свою длину, например:

```
. . .
mline := 'строка';
mline := mline + 'стала длинной';
. . .
```

В приведенном примере после первого присваивания длина переменной mline равна шести. Второе присваивание увеличивает ее длину до 19 символов. Динамические строки организуются в Турбо-Паскале следующем образом: память под строки отводится по максимуму (согласно описанию), а используется лишь та ее часть, которая реально занята символами строки. При такой организации работы со строками Турбо-Паскаль должен знать реальную длину строки. Поэтому для строковой переменной длиной N символов отводится (N+1) байтов памяти, из которых N байтов предназначены для хранения символов строки, а один байт - для хранения текущей длины строки. Символы строки нумеруются целыми числами, начиная с единицы, а байт с текущей длиной строки считается нулевым ее элементом. Длину текущей строки можно определить следующим образом:

```
len := ord(st[0]);
```

Здесь st - переменная строкового типа.

Если строковой переменной присваивается строка с длиной, большей чем максимально допустимая, происходит отсечение строки до

максимальной длины, например:

```
VAR
st : string[5];
BEGIN
st := 'очень длинная строка';
writeln(st); { будет отображено только: 'очень'}
. . .
```

Для строковых типов данных определена операция "*конкатенация*", обозначаемая символом '+'. Смысл операции заключается в формировании новой строки. Динамическая длина сформированной строки равна сумме символов строк-операндов, а ее значение равно последовательности символов исходных строк. Например:

```
VAR
str1, str2 : string[10];
st : string[25];
BEGIN
str1 := 'Паскаль - ';
str2 := 'программа';
st := str1 + str2;
WriteLn(st)
END.
```

В результате выполнения программы будет на экране отображена текстовая строка: 'Паскаль - программа'. Кроме операции конкатенации над значениями строковых типов разрешены операции сравнения < , <=, > , >=, = , <>, IN, при выполнении которых действуют следующие правила:

а) более короткая строка всегда меньше более длинной;

б) если длины сравниваемых строк равны, то происходит поэлементное сравнение символов этих строк с учетом лексикографической упорядоченности значений

в) компаратор IN определяет вхождение левого операнда в правый. Если левый операнд входит в правый, то результат сравнения будет истинным (TRUE), в противном случае - ложным (FALSE). Левым операндом может быть только элементарное данное (здесь символ), а правым - любое множество элементов, в данном случае строка или литерный ряд.

Доступ к отдельным элементам строк осуществляется аналогично доступу к элементам одномерного массива: после имени строковой переменной необходимо в квадратных скобках указать арифметическое выражение целого типа, например:

```
VAR
mline : string;
i : integer;
BEGIN
. . .
for i := 1 to length( mline ) do
if mline[i] IN ['a'...'z'] then
mline[i] := chr( ord( mline[i] ) + 1);
. . .
```

Можно заметить, что работа со строковыми данными аналогична работе с символьными массивами, однако, это не означает их полную идентичность. Так, распространенной ошибкой является работа с элементами строки без учета ее текущей длины. Необходимо помнить, что если длина символьного ряда статична, то длина строковой переменной динамична.

Регулярную переменную типа ARRAY OF CHAR (или литерный ряд) можно рассматривать как строку постоянной длины. Данные указанного типа

могут быть использованы в любых строковых выражениях. При этом согласование операндов по типам обеспечивается компилятором, который в подобных случаях просто преобразует литерный ряд в строку длиной, равной количеству элементов ряда. Это позволяет, например, сравнивать литерные ряды между собой и обращаться с ними точно так же, как с переменными типа STRING.

Допускается выполнять присваивание, в левой части которого стоит имя литерного ряда, в правой – строковое выражение (константа) длиной, равной количеству элементов ряда. Однако нельзя присваивать какой-либо переменной типа литерный ряд переменную типа STRING или наоборот. Например:

```
Const
        message = 'верно';
Type
        CharArray = array[1..5] of char;
Var
        FixedString, FiveChar : CharArray;
        VarString : string[10];
Begin
. . .
        FiveString := 'мерно';
        FixedString := message;
        if FiveChar > FiveString then
                writeln("", FiveChar,' "больше, чем"', FiveString,"");
        VarString := 'при';
        VarString := concat(VarString,FiveChar);
        VarString := 'голос';
        FiveChar := VarString; { так нельзя}
. . .
```

Пояснение к программе.

В рассмотренном примере объявлено два литерных ряда (типа CharArray) - FixedString и FiveChar, а также стринг VarString.

1. Стринговый литерал 'мерно' присваивается переменной FiveChar.

2. Переменной FixedString присваивается стринговая константа message, имеющая значение 'верно'.

3. Сравниваются два ряда, в результате которого выясняется, что FiveChar больше FixedString (поскольку 'мерно' > 'верно').

4. Переменная VarString получает значение 'при'.

5. В результате конкатенации VarString получает новое значение - 'примерно', которое на следующем шаге затирается значением 'голос'.

6. В последнем операторе делается попытка литерному ряду назначить стринговую

переменную, это недопустимо по обычной в таких случаях ошибке "несоответствие типов". В программе левый операнд объявлен как литерный ряд, а правый как стринг.

Пустой строка

Строка, длина которой равна 0, называется пустой. Пустая строкоа изображается в виде двух апострофов, записанных рядом, без пробела: ''.

Например:

```
If st = '' then
        writeln('строка st пустая');
```

Следует отметить, что переменную типа STRING необходимо инициировать пустым значением.

Функции преобразования. Для строковых типов данных определены следующие функции преобразования:

– STR(x[:width[:decimals]];var s:string) - эта функция преобразует численное значение x в его строковое представление s.

– VAL(s:string; var x code:integer) - эта функция преобразует строковое значение s в его численное представление x.

Для строковых типов данных определены следующие процедуры и функции:

– INSERT(source: string; var s: string; index: integer) - эта процедура предназначена для вставки строки SOURCE в строку S, начиная с символа с номером INDEX в этой строке.

– DELETE(var s: string; index, count: integer) - эта процедура производит удаление из строки-параметра S подстроки длиной COUNT, начиная с символа с номером INDEX.

– CONCAT(s1, [s2,...]: string): string - эта функция выполняет слияние строк-параметров, которых может быть произвольное количество. Каждый параметр является выражением строкового типа. Если длина результирующей строки превышает 255 символов, то она усекается до этой длины.

– COPY(s: string; index: integer; count: integer): string - эта функция возвращает подстроку, выделенную из исходной строки S, длиной COUNT символов, начиная с символа под номером INDEX.

– POS(substr, s: string): byte - эта функция производит поиск в строке S подстроки SUBSTR. Результатом функции является номер позиции подстроки в исходной строке.

– LENGTH(s: string): integer - эта функция возвращает текущую длину строки S.

– MOVE(var x, y; count: word) - эта функция копирует заданное количество COUNT последовательных байт из источника X в приемник Y.

– FILLCHAR(var x; count: word; value) - эта функция заполняет заданное количество

– COUNT последовательных байт переменной X указанным значением VALUE.

Варианты заданий

1. Дана строка символов. Подсчитать сколько среди символов данной строки встречается буква x.

2. Дана строка символов. Подсчитать:

a) сколько раз в данной строке встречается символ + и сколько раз символ *;

b) общее число вхождений символов +, -, * в строку.

3. Дана строка символов. Преобразовать данную строку, заменив в ней:

a) все восклицательные знаки точками;

b) каждую точку многоточием.

4. Дана строка символов S. Выяснить имеются ли в данной строке такие символы si, si+1, что si - это запятая, а si+1 - это тире.

5. Даны две строки символов S1 и S2. Выяснить, верно ли, что среди символов строки S1 имеются все буквы строки S2.

6. Дана строка символов. Удалить из данной строки все группы букв вида asdf.

7. Дана строка символов. Преобразовать строку, удалив каждый символ * и повторив каждый символ, отличный от *.

8. Дана строка символов. Заменить в данной строке каждую группу букв child группой букв children.

9. Дана строка символов. Исключить из строки группы символов, расположенные между парами скобок (,), {, }. Сами скобки тоже должны быть исключены. Предполагается, что внутри каждой пары скобок нет других скобок.

10. Дана строка символов. Словом будем называть группы символов, разделенных пробелами (одним или несколькими). Подсчитать количество слов в данной строке.

2.2. Массивы

Массив, в отличии от простой переменной, представляет собой не одно значение, а множество значений, объединенных одним именем. В языке Turbo Pascal’е все значения из этого множества должны иметь один и тот же тип. Каждое из значений массива называется элементом массива. Доступ к элементам массива производится посредством указания имени массива и номера элемента массива, заключенного в квадратные скобки, например mas[1] – первый элемент массива mass. Номер элемента массива называется индексом элемента массива. Использование элемента массива не отличается от использования простой переменной, имеющей тот же тип, что и элемент массива.

Синтаксис объявления массива N-мерного массива:

Var <Name>: Array [<From1>..<To1>, …, <FromN>..<ToN>] of <Type>;

Примеры описаний массивов

Var B Array [5..8] of real; {массив B, состоящий из 4 элементов вещественного типа с индексами от 5 до 8}

Var C Array [1..3, 5..9] of ClockStruc; {двумерный массив C, состоящий из 3*5 элементов типа ClockStruc. Тип ClockStruc должен быть определен перед объявлением массива.

Пример работы с массивами:

```
Begin
A[1]:=3; {в элемент массива A с индексом 1 записали число 3}
A[4]:=A[1]+1; {в элемент массива A с индексом 4 записали
число 3+1=4}
B[5]:=0.111; {в элемент массива B с индексом 5 записали
число 0.111}
B[A[1]+A[4]]:=B[5]*2; {в элемент массива B с индексом=
A[1]+A[4]=3+4= 7 записали число 0.222}
End.
```

Индексы массива

В качестве индекса массива можно использовать любой порядковый тип, кроме типа Longint. Напомним, что порядковый тип – это тип, все значения которого можно перечислить. К таким типам относятся все целые типы(integer, shortint, longint, byte, word), все логические (boolean, wordbool, longbool, bytebool), символьный тип (char), перечисляемые типы и типы-диапазоны.

Примеры использования в качестве индексов порядковых типов:

```
Var {примеры объявления массивов}
A: Array [Byte] of integer; {массив A, состоящий из 256 элементов, нижняя граница индекса 0, верхняя 255}
B: Array [Char] of real; {массив B, состоящий из 256 элементов, нижняя граница индекса #0(символ с кодом 0), верхняя граница индекса #255(символ с кодом 255)}
i:Byte; {переменная, используемая как индекс массива A}
c:Char; {переменная, используемая как индекс массива B}
Begin {примеры обращения к элементам массива}
A[45]:=0; {B элемент массива A, имеющий индекс 45, записали 0 }
B[‘t’]:=2.4; {B элемент массива B, имеющий индекс ‘t’, записали 2.4}
i:=200; {i присвоили значение 200 }
c:=’#’; {c присволили значение ‘#’ }
A[i]:=23400; {B элемент массива A, имеющий индекс i=200, записали 23400}
B[c]:=123.456; {B элемент массива B, имеющий индекс c=’#’, записали 123.456}
End.
```

Обычно в качестве индекса используют диапазон значений какого-либо перечисляемого типа. Например:

```
Var {примеры объявления массивов}
C: Array [-10..5] of integer; {массив C, состоящий из 16 элементов, нижняя граница индекса -10, верхняя 5}
D: Array [‘A’..’Z’] of char; {массив D, состоящий из 26 элементов, нижняя граница индекса ’A’, верхняя граница индекса ‘Z’}
j: -10..5; {переменная, используемая как индекс массива C}
c1: ‘A’..’Z’; {переменная, используемая как индекс массива D}
k: integer; {эту переменную можно использовать в качестве индекса массива C, т.к. –10..5 – это диапазон значений целого типа}
c2: char; {эту переменную можно использовать в качестве индекса массива D, т.к.’A’..’Z’ – это диапазон значений символьного типа}
begin {примеры обращения к элементам массивов}
C[-4]:=3;
D[‘F’]:=’%’;
j:=4; C[j]:=-10;
c1:=’R’; D[c1]:=’q’;
k:=-3; C[k]:=80;
c2:=’G’; D[c2]:=’Й’;
end.
```

Чаще же всего используют диапазон значений целого типа, причем нижний индекс обычно берут равным 1. Например:

```
Var E: Array [1..10] of integer; {массив E, состоящий из 10 элементов,
нижняя граница индекса 1, верхняя 10}
```

Представление массива в памяти Элементы массива размещаются в памяти в последовательных ячейках. Массив занимает количество байт, равное произведению количества элементов массива на размер одного элемента:

SizeOfArray = NumElement * SizeOfElement

где SizeOfArray – размер массива

NumElement – количество элементов в массиве

SizeOfElement – размер одного элемента

Адрес первого (по порядку) элемента массива является адресом массива (будем обозначать его AdrArray). Адрес i-го элемента массива (его будем обозначать AdrI) можно вычислить по формуле:

AdrI = AdrArray + (i – нижняя_граница_индекса) * SizeOfElement

Для примера рассмотрим массив B, определенный выше. Нижняя граница индекса этого массива = 5. Первый (по порядку) элемент массива - B[5]. Пусть его адрес = 100. Размер каждого элемента 6 байт, поскольку тип элементов - Real.

Вычислим адреса остальных элементов массива

Adr6 = 100 + (6-5)*6 = 100 + 1*6 = 106

Adr7 = 100 + (7-5)*6 = 100 + 2*6 = 112

Adr8 = 100 + (8-5)*6 = 100 + 3*6 = 118

Графически покажем взаимное расположение элементов этого массива:

Адрес элемента	Элемент
100	B[5]
106	B[6]
112	B[7]
118	B[8]

Замечание: Один массив может занимать в памяти не более 65520 байт. Нельзя, например,

определить такой массив C:

```
Var C: array[1..50000] of integer;
```

Каждый элемент этого массива занимает в памяти 2 байта, элементов 50000, значит весь массив занимает 100000 байт > 65520 байт.

Пользовательский тип – массив.

В программе можно определить тип массива, для того чтобы потом его использовать для определения переменных типа массива.

Пример:

```
Type
Arr = array[1..20] of integer; {определили тип массива целых чисел содержащего 20 элементов}
Var
A,B: Arr; {A и B – массивы целых чисел, содержащие по 20 элементов}
```

Дальше с массивами A и B можно работать как с обычными массивами:

A[3]:=2; B[4]:=A[3]; и т.д.

Кроме типа массива в программе можно определить и специальный тип для индексов. Этот тип должен быть интервальным.

Пример:

```
Type
IndexEl = 1 .. 20; {тип индекса элемента}
Arr = array[IndexEl] of integer; {тип массива целых чисел содержащего 20 элементов}
Var
A,B: Arr; {A и B – массивы целых чисел, содержащие по 20 элементов}
i,j: IndexEl; {переменные, используемые для указания индекса элемента}
```

Одномерные и n - мерные массивы.

Все массивы, которые приведены выше, называются одномерными – у элементов одномерных массивов в квадратных скобках указывается только один индекс (у таких массивов только одно измерение).

Кроме одномерных массивов могут быть и двумерные, и трехмерные, и прочие n-мерные массивы. «Мерность» массивов определяется количеством

индексов, указываемых в квадратных скобках, для того чтобы определить элемент массива.

Пример:

A[7] – A – одномерный массив

S[2,-3] – S – двумерный массив

W[1,0,0] – W – трехмерный массив

Z[-1,3,4,3,0] – Z – пятимерный массив

На практике чаще всего используются одномерные массивы, реже двумерные, и значительно реже массивы больших размерностей.

Двумерные массивы.

Одномерный массив можно представить в виде строки. Например, массив целых чисел можно представить строкой целых чисел, например такой: 3 2 4 1 3.

Двумерный массив можно представить в виде прямоугольной таблицы, например такой:

2 3 4 5

0 4 8 3

7 1 5 3

Чтобы определить такой массив, в программе надо написать:

```
Var
A: array[1..3,1..4] of integer;
```

Здесь в массиве A первый интервал индексов - 1..3 – обозначает индекс номера строки, а второй интервал индексов – 1..4 – обозначает индекс номера столбца. Для обращения к элементу двумерного массива необходимо в квадратных скобках сначала указать номер строки, а затем номер столбца.

Например:

```
Writeln(A[2,3]); {будет выведено число 8}
Writeln(A[3,1]); {будет выведено число 7}
```

```
Writeln(A[1,1]); {будет выведено число 2}
```

Пример: Подсчитать сумму элементов массива.

Алгоритм содержит два пункта:

1. Сумма S=0.

2. Проход по всем элементам массива и прибавление их значений к сумме S.

Приведем текст программы этой задачи:

```
Program SumExample;
Const maxN = 20; {максимально возможное количество элементов в массиве}
Type IndexEl = 1 .. maxN; {индексы массива лежат в интервале от 1 до maxN}
arrInt = array[interval] of integer; {массив целых чисел содержащий до maxN элементов}
Var
    a:arrInt; {массив}
    n:interval; {размерность массива}
    i:IndexEl; {переменная для сканирования массива}
    S:integer; {сумма элементов массива}
Begin
    { ввод массива с клавиатуры }
    write('Введите n=');
    read(n); {ввод количества элементов}
    for i:=1 to n do
        read(A[i]); {ввод самих элементов}
        {Подсчет суммы элементов массива}
    s:=0;    {Инициализируем переменную результата}
    for i:=1 to n do
        s:=s+A[i];   {подсчитываем сумму}
```

```
        {Вывод полученного значения суммы}
        writeln('сумма элементов массива S=', S);
end.
```

Варианты заданий

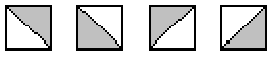
1. Даны целые числа a1, a2, a3. Получить целочисленную матрицу b[i,j] = ai-3aj.

2. Даны, i=1,…10; j=1,…12. Получить целочисленную матрицу b, для которой bij=i+2j.

3. Дано натуральное число n, i, j=1,…n. Получить действительную матрицу адля которой a[i,j]= 3i+j^2.

4. Дана действительная квадратная матрица порядка n. Найти наибольшее из значений элементов, расположенных в заштрихованной части матрицы.

а) б) в) г)



5. Дана квадратная вещественная матрица размерности n. Найти количество нулевых элементов, стоящих: выше главной диагонали; ниже главной диагонали; выше и ниже побочной.

6. Дана вещественная матрица размерности n * m. По матрице получить логический вектор, присвоив его k-ому элементу значение True , если выполнено указанное условие и значение False иначе: - все элементы k столбца нулевые; - элементы k строки матрицы упорядочены по убыванию; - k строка массива симметрична.

7. Дана вещественная матрица размерности n * m. Сформировать вектор b, в котором элементы вычисляются как: - произведение элементов соответствующих строк; - среднее арифметическое соответствующих столбцов; - разность наибольших и наименьших элементов соответствующих строк; - значения первых отрицательных элементов в столбце.

8. Дан двухмерный массив A[1..m,1..n]. Написать программу построения одномерного массива B[1..m], элементы которого соответственно равны а) суммам элементов строк, б) произведениям элементов строк, в) наименьшим средних арифметических элементов строк.

9. Расположить элементы данного массива в обратном порядке (первый элемент меняется с последним, второй - с предпоследним и т.д. до середины; если массив содержит нечетное количество элементов, то средний остается без изменения).

10. В данном массиве поменять местами элементы, стоящие на нечетных местах, с элементами, стоящими на четных местах.

11. В массиве A[1..N,1..N] определить номера строки и столбца какой-нибудь седловой точки. Некоторый элемент массива называется седловой точкой, если он является одновременно наименьшим в своей строке и наибольшим в своем столбце.

12. Массив A[1..5,1..7] содержит вещественные числа. Требуется ввести целое число K и вычислить сумму элементов A[I,J], для которых I+J=K. Прежде, однако следует убедиться, что значение K позволяет найти решение, в противном случае нужно напечатать сообщение об ошибке.

13. Дан массив A[1..N,1..N]. Составить программу, которая прибавила бы каждому элементу данной строки элемент, принадлежащий этой строке и главной диагонали.

14. Дана матрица NxM. Переставляя ее строки и столбцы, переместить наибольший элемент в верхний левый угол. Определить можно ли таким же образом поместить минимальный элемент в нижний правый угол.

15. Заполнить двухмерный массив T[1..n,1..n] последовательными целыми числами от 1 до n, расположенными по спирали, начиная с левого верхнего угла и продвигаясь по часовой стрелке:

1	2	3	4	5	6
20	21	22	23	24	7
29	32	33	34	25	8

18 31 36 35 26 9

17 30 29 28 27 10

16 15 14 13 12 11

16. Элемент двухмерного массива называется локальным минимумом, если он строго меньше всех имеющихся у него соседей. Подсчитать количество локальных минимумов заданной матрицы размером NxN найти максимум среди всех локальных минимумов.

17. В массиве все четные элементы обнулить. Пример: из массива A[5]: 1 3 4 5 6 должен получиться массив 1 3 0 5 0

18. В массиве все нечетные элементы заменить на 1. Пример: из массива A[5]: 1 3 4 5 6 должен получиться массив 1 1 4 1 6

19. В массиве все элементы, стоящие перед четными, заменить на 9. Пример: из массива A[5]: 1 3 4 5 6 должен получиться массив 1 9 4 9 6

20. Из массива удалить последний из четных элементов. Пример: из массива A[5]: 1 3 4 5 6 должен получиться массив A[4]: 1 3 4 5

21. Удалить максимальный из четных элементов. Пример: из массива A[5]: 2 3 4 7 5 должен получиться массив A[4]: 2 3 7 5

22. Из массива удалить четные элементы, имеющие значение больше среднего арифметического всех элементов массива. Пример: из массива A[5]: 8 7 2 6 5 должен получиться массив A[3]: 7 2 5 (среднее арифметическое всех элементов =(8+7+2+6+5)/5=5.6)

23. Из массива удалить элементы, кратные трем, стоящие между максимальным и минимальным элементами. Пример: из массива A[7]: 1 9 3 4 9 0 0 должен получиться массив A[5]: 1 9 4 0 0

24. Из массива удалить нечетные элементы, встречающиеся в массиве только один раз. Пример: из массива A[7]: 4 1 4 3 1 9 0 должен получиться массив A[5]: 4 1 4 1 0

25. Из массива A удалить те элементы, которые встречаются и в массиве A и в массиве B по крайней мере по 2 раза. Пример: массив A[8]: 3 3 4 5 2 3 5 9 массив B[7]: 1 2 3 4 5 2 5. По 2 раза в обоих массивах встречается

только элемент, равный 5. Массив A после удаления примет вид: A[6]: 3 3 4 2 3 9

26. Из массива A удалить те цепочки нечетных элементов, в которых нет ни одного элемента из массива B. Пример: массив A[10]: 3 2 7 5 2 1 2 6 3 9 массив B[5]: 1 2 5 4 8 Массив A после удаления примет вид: A[7]: 2 7 5 2 1 2 6

27. Между массивами A и B обменять их самые длинные цепочки из четных элементов. Пример: массив A[10]: 3 2 4 6 2 1 1 1 8 9 массив B[7]: 1 0 5 5 4 3 3 В массиве A самая длинная цепочка: 2 4 6 2 (элементы со 2 по 5) В массиве B самая длинная цепочка: 0 (элемент 2) Массив A после перестановки в него цепочки из массива B: A[7]: 3 0 1 1 1 8 9 Массив B после перестановки в него цепочки из массива A: B[10]: 1 2 4 6 2 5 5 4 3 3

28. Найти максимальный и минимальный элементы среди элементов, расположенных на побочной диагонали квадратной матрицы.

29. Найти в каждой строке матрицы минимальный элемент. Выделить максимальное из полученных значений.

30. Найти в квадратной матрице сумму элементов, расположенных ниже побочной диагонали.

31. Найти на главной диагонали квадратной матрицы максимальный и минимальный элементы. Поменять местами строки, в которых они расположены.

32. В строке матрицы с максимальным элементом обнулить все элементы, кроме максимального.

33. Сформировать одномерный массив, элементы которого находятся суммированием элементов строк матрицы, в которых находятся максимальный и минимальный элементы.

34. Найти в каждой строке матрицы минимальный среди среди положительных элементов.

35. Обнулить столбцы матрицы, в которых находятся максимальный и минимальный элементы.

2.3. Записи

Комбинированный тип характеризует объекты, называемые записями. Синонимом понятия "комбинированный тип" является понятие "структурный тип". Запись (структура) – это сложная переменная с несколькими компонентами. При определении комбинированного типа задаются имя всей записи, имя и тип каждой компоненты. Описание комбинированного типа начинается со служебного слова RECORD и заканчивается словом END. Записи, как и другие данные, объявляются в разделе описаний и используются в разделе операторов. Записи можно объявлять в разделе TYPE либо VAR. Объявление записи в разделе VAR имеет следующий вид:

```
VAR
        < имя записи > : RECORD
        < имя компоненты 1: тип >;
        < имя компоненты 2: тип >;
        . . .
        < имя компоненты N: тип >
END;
```

Здесь служебное слово RECORD (запись) выполняет роль открывающей операторной скобки, END - закрывающей. Внутри операторных скобок описываются компоненты записи. Допускается вместо имени записи указывать список имен, т.е. имена записей, разделенные запятыми. Компоненты записи вместе с их описанием называются полями записи. Более универсальной формой объявления записи является описание с использованием раздела TYPE, которое имеет вид:

```
TYPE
        < имя типа > = RECORD
```

```
        < имя компоненты 1 >: тип;
        < имя компоненты 2 >: тип;
        . . .
        < имя компоненты N >: тип
END;
VAR < имя записи >: имя типа;
```

Пример. Дана ведомость списка студентов с их оценками (см таб.) Для представленной ведомости объявление записи в разделе переменных выглядит следующим образом:

```
VAR
vedom : RECORD
n : integer;
fio : string[15];
progr,fizika : integer
END;
```

Таблица успеваемости.

№ п/п	Фамилия имя, отчество	Оценка программирование	Оценка физика
1	Бадмаев И.П.	5	4
2	Иванов А.Р.	4	4
3	Павликова Ю.Т.	5	3

В данном примере фамилия имеет тип STRING, состоящий из 15 элементов, порядковый номер и оценки по предметам - тип INTEGER. Длина записи Vedom равна 21 байту. Объявление ведомости с использованием раздела типов имеет вид:

```
TYPE
    vedom = RECORD
    n : integer;
    fio : string[15];
    progr,fizika : integer
END;
VAR v : vedom;
```

Здесь сначала введен тип с именем VEDOM, а затем указана переменная V, имеющая тип записи.

Поле записи используется в программе так же, как обычная переменная. Таким образом, поле записи можно указывать как в левой части оператора присваивания, так и в выражениях. Над полем записи можно выполнять действия, допустимые для данных его типа. Если тип поля записи - INTEGER, то выполняются все операции, допустимые для целых данных.

Доступ к полям записи производится с помощью селектора записи, имеющего следующий вид:

NAME_Z.NAME_P , где NAME_Z - имя переменной комбинированного типа (всей записи); NAME_P - имя поля.

В практическом программировании такая запись называется уточненным именем данного. Для переменных, введенных выше, допустимы следующие конструкции:

```
Vedom.n := 5;
Vedom.fio := 'Иванов А.Р.';
или
V.n := 35;
V.fio := ' Павликова Ю.Т.';
```

Комбинированные типы можно использовать для построения более сложных структур: массивов; файлов; вложенных структур с одним или более полей, которые, в свою очередь, могут быть записью. Например:

```
VAR
    group : array[1..10] of vedom;
    database : file of vedom;
```

Для переменных GROUP доступ к полям записей, составляющих этот массив, производится следующим образом:

```
. . .
Group[i].fio := ' Бадмаев И.П. ';
If group[i].fio = ' Бадмаев И.П. ' then
WriteLn (group[i].progr)
Else writeln ('Нет такой фамилии!');
. . .
```

Рассмотрим случай, когда в составе записи содержатся поля, имеющие тип записи. Пусть для комбинированного типа VEDOM необходимо хранить информацию о дате сдачи экзамена. Эту информацию можно представить в виде трех полей: месяц, день, год, дополняющих предыдущий состав типа VEDOM. Однако, логичнее дату сдачи экзамена определить как отдельный тип. Это позволит использовать тип DATE в описании других типов и переменных:

```
TYPE
    Date = RECORD
    Mounth : (jan,feb,mar,apr,may,jun,jul,aug,sep,oct,nov,dec);
    Day : 1..31;
    Year : 1900..2000
END;
```

Теперь тип DATE можно использовать в записи VEDOM:

```
TYPE
    Vedom = RECORD
    N : integer;
    Fio : string[15];
    Progr,Fizika : integer;
    D_exem : date
END;
```

Доступ к полям D_EXEM осуществляется по общим правилам, т.е. при записи селектора слева от символа 'точка ' всегда должна находиться переменная типа запись, а справа идентификатор поля этой записи, например:

```
V.D_exem.Mounth := jan;
V.D_exem.Day := 25;
```

Комбинированный тип может употребляться для спецификации параметров подпрограмм. Например, можно определить специальный тип для представления комплексных чисел как пары вещественных переменных (действительную и мнимую части комплексного числа):

```
TYPE
    Complex = RECORD
    Re,Im : real
END;
```

Далее можно с помощью процедур определить операции над комплексными числами (сложение, умножение, деление):

```
Procedure Addc( c1,c2: complex; var R: complex);
Procedure Mulc( c1,c2: complex; var R: complex);
```

```
Procedure Divc( c1,c2: complex; var R: complex);
```

Записи с вариантами

Часто в зависимости от конкретного значения некоторого поля возникает необходимость в пределах одной записи иметь различную информацию. В таких случаях используются записи с вариантами.

Рассмотрим тип PERSON, содержащий информацию о человеке. Если поле POL имеет значение M (мужской), то пусть необходимо предусмотреть такие поля:

- служил в армии или нет;
- если служил, то дату последних военных сборов.

Если поле POL имеет значение W (женский), то необходима информация о цвете глаз. Запись с вариантами типа PERSON имеет вид:

```
Type
    Date = Record
        Mounth : (jan,feb,mar,apr,may,jun,jul,aug,sep,oct,nov,dec);
        Day : 1..31;
        Year : 1900..2000
    End;
    Person = Record
    Fio : string[20];
    Special : word;
    Birthday : date;
    PersonPol: (M,W);
    Case pol : PersonPol of
        M: (Army : boolean; D_Army : date);
        W: (EyesColor : (blue,brown, gray,green))
End;
```

Записи с вариантами имеют фиксированную и вариантную части. Изменяющаяся часть записи называется вариантом. Вариант всегда располагается в конце записи. Поле (в данном случае POL), позволяющее различать варианты, называется полем признака. Вариантная часть содержит несколько альтернатив (в данном примере - M и W), в каждой из которых в круглых скобках задается список полей, присущих данному варианту (ARMY и D_ARMY -> M, EYESCOLOR -> W). Списку полей предшествует метка, являющаяся конкретным значением признака POL. Метка служит критерием выбора вариантов. Перечисление альтернатив начинается с определения признака POL. Началом вариантной части является служебное слово CASE; после признака выбора вариантов записывается служебное слово OF. Вариантная часть завершается служебным словом END вместе с завершением всей записи. В определении комбинированного типа может быть только одна вариантная часть и она должна быть задана в конце записи. Альтернативы вариантной части помечаются допустимыми значениями поля POL, которое определено в фиксированной части. Иногда поле, значения которого задают варианты, называют дискриминантом записи. Идентификаторы полей во всех вариантах должны быть различными и отличаться от идентификаторов полей фиксированной части. В этом случае после метки, соответствующей этим значениям может стоять пустой список вида ( ). Любой вариант, в свою очередь, может иметь свою вариантную часть, которая должна располагаться в конце списка полей данного варианта. При использовании вариантных записей необходимо учитывать следующие особенности:

1. Для размещения переменной комбинированного типа всегда отводится фиксированный объем памяти, причем если в записи есть варианты, то объем определяется по самому большому варианту. Различные варианты одной записи как бы накладываются " друг на друга" в памяти, занимая одну и ту же область.

2. Система Турбо-Паскаль не содержит никаких средств контроля за правильностью работы с вариантами записей. За соответствием текущего значения дискриминанты и доступа к полям записи должен следить программист.

Оператор присоединения предназначен для более наглядной и эффективной организации работы с данными комбинированного типа и используется для доступа к полям записи. Оператор присоединения начинается со служебного слова WITH, далее следует имя записи и служебное слово DO. Операторы, содержащие имена полей записи, заключаются в операторные скобки:

```
WITH < имя записи > DO
BEGIN
    < операторы, содержащие имена полей записи >
    Фиксированная часть
    Вариантная часть
END;
```

Например: для рассмотренной записи (списка студентов) операции присваивания, суммирования и ввода можно объединить в один оператор:

```
. . .
With v do
    Begin
        N := 4;
        SUM := progr + fizika;
        Read (N)
    End;
```

Варианты заданий

1. Дан список учебной группы, включающий 20 человек. Для каждого студента известны: фамилия, имя, дата рождения, оценки по всем дисциплинам за последний семестр. Составить программу, которая обеспечивает ввод информации и отображение ее на экран в виде таблицы. Отобразить на экран анкетные данные студентов-отличников в виде таблицы. Отобразить на экран фамилию и имя студентов, родившихся зимой и весной.

2. Сведения об экзамене содержат следующие данные: дисциплину (программирование, алгебра, история, геометрия), дату сдачи экзамена (год, месяц, день), сведения о студенте (факультет, курс, группа, номер в журнале) и экзаменационную оценку. Задан набор сведений об экзаменах, сданных студентами за последние два года; в них факультет и предмет кодируются первыми буквами названия. Определить количество неуспевающих по программированию на экономическом факультете среди студентов первого курса, сдававших экзамены зимой 1995 года, вывести на экран их группу и номер в журнале.

3. Сведения об экзамене содержат следующие данные: дисциплину (программирование, социология, иностранный язык, физика), дату сдачи экзамена (год, месяц, день), сведения о студенте (фамилия, факультет, курс, группа) и экзаменационную оценку. Задан набор сведений об экзаменах, сданных студентами за последние несколько лет; в них факультет и предмет кодируются первыми буквами названия. Определить количество отличников по программированию на технологическом факультете среди студентов первого курса, сдававших экзамены летом 1995 года, вывести на экран их фамилии и группу.

4. Сведения об экзамене содержат следующие данные: дисциплину (программирование, вычислительная техника, информатика), дату сдачи экзамена (год, месяц, день), сведения о студенте (факультет, курс, группа, номер в журнале) и экзаменационную оценку. Задан набор сведений об

экзаменах, сданных студентами за последние несколько лет; в них факультет и предмет кодируются первыми буквами названия. Определить, на каком факультете самый высокий средний балл по программированию среди студентов первого и второго курсов, сдававших экзамены зимой 1995 года.

5. Сведения об экзамене содержат следующие данные: дисциплину (программирование, вычислительная техника, информатика), дату сдачи экзамена (год, месяц, день), сведения о студенте (факультет, курс, группа, номер в журнале) и экзаменационную оценку. Задан набор сведений об экзаменах, сданных студентами за последние несколько лет; в них факультет и предмет кодируются первыми буквами названия. Определить, на каком факультете самый высокий показатель качества успеваемости по информатике (то есть самый высокий процент отличников и хорошистов) среди студентов первого курса, сдававших экзамены зимой 1995 года или летом 1996 года.

6. Справка о междугороднем телефонном разговоре содержит: номер телефона абонента (6 цифр), дату (год, месяц, день), время (час, минута), код города (3 цифры), номер телефона в другом городе (7 цифр), продолжительность разговора (в минутах), категорию (срочный, обычный) и тариф (плата в рублях за минуту). Определить дату такого телефонного разговора, которой является максимальным по продолжительности среди срочных разговоров за указанный месяц.

7. Справка о междугороднем телефонном разговоре содержит: номер телефона абонента (6 цифр), дату (год, месяц, день), время (час, минута), код города (3 цифры), номер телефона в другом городе (7 цифр), продолжительность разговора (в минутах), категорию (срочный, обычный) и тариф (плата в рублях за минуту). Вывести на экран код города и номер телефона в другом городе для телефонных разговоров, состоявшихся с телефона 235678 8 марта 1996 года.

8. Справка о междугороднем телефонном разговоре содержит: номер телефона абонента (6 цифр), дату (год, месяц, день), время (час, минута), код

города (3 цифры), номер телефона в другом городе (7 цифр), продолжительность разговора (в минутах), категорию (срочный, обычный) и тариф (плата в рублях за минуту). Вывести на экран номер телефона абонента, код города и номер телефона в другом городе для срочных телефонных разговоров, состоявшихся между 15 марта и 12 апреля 1996 года.

9. Деталь автомобиля описывается инвентарным номером (положительное целое число), весом (в килограммах), ценой и стоимостью (в рублях), датой начала производства (год, месяц, день), статусом (имеет или не имеет знак качества) и объемом производства (в штуках за смену). В заданной последовательности сведений о деталях найти инвентарные номера деталей с наибольшей датой начала производства среди всех заданных деталей. Вывести на экран инвентарный номер, объем производства, цену и стоимость деталей со знаком качества.

10. Деталь автомобиля описывается инвентарным номером (положительное целое число), весом (в килограммах), ценой и стоимостью (в рублях), датой начала производства (год, месяц, день), статусом (имеет или не имеет знак качества) и объемом производства (в штуках за смену). В заданной последовательности сведений о деталях найти инвентарные номера деталей с минимальным весом среди деталей без знака качества. Вывести на экран инвентарный номер, объем производства, цену и стоимость деталей, выпускаемых с февраля 1977 года.

2.4 Множества

В математике под множеством понимается некоторый набор элементов. Например, множество фигур на плоскости (прямоугольник, круг, ромб, квадрат). К множествам применимы следующие операции:

1. Объединение множеств (C = A □ B). Каждый элемент множества C является элементом либо множества A, либо множества B.

2. Пересечение множеств (C = A ∩ B). Каждый элемент множества C является элементом множеств A и B одновременно.

3. Разность двух множеств (C = A \ B). Каждый элемент множества C является элементом множества A, но не является элементом множества B.

Например:

а) { круг, ромб } □ { круг, квадрат } = {круг, ромб, квадрат};

б) { круг} ∩ { круг, ромб, квадрат } = { круг };

в) { круг, ромб, квадрат } \ { круг, квадрат } = { ромб }.

Под множеством в языке Турбо - Паскаль понимают ограниченный, неупорядоченный набор различных элементов одинакового типа.

Множественный тип задается с помощью двух служебных слов SET и OF, после которых указывается базовый тип. В качестве базового типа можно использовать следующие типы: INTEGER, BYTE, CHAR, перечислимый и ограниченный.

При определении множественных типов существует два ограничения:

1) вещественный тип в качестве базового в множествах использовать нельзя;

2) число элементов в множестве определяется каждой конкретной реализацией ЭВМ.

Множества объявляются либо в разделе описания переменных VAR, либо в разделе описании типов TYPE. Объявление множества в разделе описания переменных имеет вид:

VAR

< имя множества > : SET OF < базовый тип >;

Например:

```
Var
        god : set of 1900..2000;
```

```
    symbol : set of char;
```

Объявление множества с использованием раздела описания типов имеет вид:

```
Type
    < имя типа > = set of < базовый тип >;
Var
    < имя множества > :  <имя типа>;
```

Например:

```
Type
    god = set of 1900..2000;
    symbol = ( 'A'..'Z');
Var
    g : god;
    s : set of symbol;
```

Значения переменных и констант множества задаются в разделе операторов с помощью конструктора. Конструктор представляет собой список элементов базового типа, заключенный в квадратные скобки, который затем можно присвоить переменной, или обработать. Конструктор множества можно рассматривать как константу типа множества. Например:

```
figura := [romb];
figura := [krug,romb,kvadrat];
simv := ['A','B','C'];
M1 := [1,3,5,10];
M2 := []; { пустое множество }
```

Операции над множествами.

В языке Турбо-Паскаль имеются следующие группы операций над множествами:

1) объединение, пересечение, вычитание множеств;

2) проверка принадлежности элемента множеству;

3) проверка на равенство и неравенство множеств;

4) проверка на принадлежность одного множества другому.

Операции объединения, пересечения и вычитания являются традиционными действиями над множествами и обозначаются символами '+', '*', '-' соответственно. Например:

[1,2] + [3,4] = [1,2,3,4];
[1..10] + [5..15] = [1..15];
[1..10] * [5..15] = [5..10];
[1,2] * [3,4] = [];
[1..10] - [5..15] = [1..4];

Проверка принадлежности множеству - это логическая операция, которая обозначается служебным словом IN. Правый операнд должен быть множеством, левый – значением базового типа множества. Операция возвращает TRUE, если значение входит в множество, и FALSE в противном случае. Например:

2 in [1..10,12]; { имеет значение true}

5 in [1,2,7,10]; { имеет значение false}

Операцию проверки принадлежности удобно использовать для исключения более сложных проверок, например, оператор вида

if (symb = 'a') or (symb = 'b') or (symb = 'x') or (symb ='y') then s;

может быть переписан в более компактной форме

if symb in ['a','b','x','y'] then s;.

Второй вариант эффективен с точки зрения быстродействия.

Проверка на равенство, неравенство и включение множеств - это бинарные логические операции, которые обозначаются следующими символами:

= равенство (совпадение) двух множеств;

<> неравенство множеств;

<= проверка на вхождение множества из левого операнда в множество из правого операнда;

>= проверка на вхождение множества из правого операнда в множество из левого операнда.

Все эти операции вырабатывают логическое значение TRUE или FALSE в зависимости от успеха проверки. Например:

[1,2,3] = [1,2] - false;

[1,2,3] >= [1,2] - true;

[S] <= [1..10] - true, если S - целое число из диапазона 1..10;

[1,2,3] <> [1,2,2] - true.

Синонимом логической операции над множествами является слово "сравнение". Набор операций над множествами в языке Турбо-Паскаль не содержит одной практически важной операции - выборки значений из множества (или близко связанного с ней средства циклического перебора значений множества). Поэтому при необходимости подобных действий приходится организовывать цикл по всему диапазону значений базового типа, проверяя на каждой итерации принадлежность очередного значения данному множеству,

например:

```
Var
        symbols : set of char;
        s : char;
Begin
        . . .
```

```
For s := chr(0) to chr(255) do
        if s in symbols then
                < действия с переменной s >
. . .
```

Варианты заданий

1. Даны два конечных множества A и B, элементами которых могут быть любые целые числа в диапазоне от 1 до 30. Найти прямое произведение этих множеств и вывести его на экран.

2. Даны два прямоугольника. Множества A и B - это множества точек, принадлежащих соответствующим прямоугольникам. Координаты точек - это натуральные числа от 1 до 10. Определить пересекаются ли данные прямоугольники, если пересекаются, то вывести на экран их общие точки.

3. Даны два конечных множества X и Y, состоящие из целых чисел. Определить выполняется ли равенство: (A□B)\B=A.

4. Даны два конечных множества X и Y, состоящие из целых чисел. Определить выполняется ли равенство: (A\B) □ (B\A)=(A□B) \ (A∩B).

5. Пусть A, B, C - конечные множества, такие что B □ A □ C. Найдите множество X, удовлетворяющее условиям A ∩ X =B и A □ X =C.

6. Пусть A, B, C - конечные множества, такие что B □ A, A ∩ C= □. Найдите множество X, удовлетворяющее условиям A \ X=B и X \ A =C.

7. Даны следующие множества A={1, 2, 3}, B={2, 3, 5, 4}, U={0, 1, 2, 3, .. ,9 }. Найти и вывести на экран A □ B, , A \ B, B \ A, U \ A.

8. Даны два конечных множества A и B, состоящие из целых чисел. Найти и вывести на экран (A □ B) \ (A ∩ B).

9. Даны два множества A={1, 2}, B={3, 4, 5}. Выведите на экран элементы множеств A × B, B × A.

10. Пусть A={b, o}. Перечислите элементы множеств A3 и A4.

2.4. Файлы

Файловый тип данных или файл определяет упорядоченную совокупность произвольного числа однотипных компонент. Общее свойство массива, множества и записи заключается в том, что количество их компонент определено на этапе написания программы, тогда как количество компонент файла в тексте программы не определяется и может быть произвольным.

Файлы на внешних устройствах часто называют физическими файлами. Их имена определяются операционной системой. В программах на языке Паскаль имена файлов задаются с помощью строк. Например, имя файла на диске может иметь вид:

```
'A:LAB1.DAT'
'c:\ABC150\pr.pas'
'lab3.pas'.
```

Для работы с файлами в программе необходимо определить файловую переменную. TURBO PASCAL поддерживает три файловых типа: текстовые файлы, компонентные файлы, бестиповые файлы. Описание файловых переменных текстового типа производится с помощью служебного слова Text, например:

```
var tStory: Text;
```

Описание компонентных файлов имеет вид:

```
var fComp: File of T;
```

где T - тип компоненты файла. Примеры описания файловой переменной компонентного типа:

```
type M= array[1..500] of Longint;
var f1: File of Real;
```

```
f2: File of Integer;
    fLi: File of M;
```

Бестиповые файлы описываются с помощью служебного слова File:

```
var f: File;
```

Файловые переменные, которые описаны в программе, называют логическими файлами. Все основные процедуры и функции, обеспечивающие ввод - вывод данных, работают только с логическими файлами. Физический файл должен быть связан с логическим до выполнения процедур открытия файлов.

Существует ряд процедур и функций, применимых для любых типов файлов: Assign, Reset, Rewrite, Close, Rename, Erase, Eof, IOResult.

Процедура Assign (var f; FileName: String) связывает логический файл f с физическим файлом, полное имя которого задано в строке FileName.

Процедура Reset (var f) открывает логический файл f для последующего чтения данных или, как говорят, открывает входной файл. После успешного выполнения процедуры Reset файл готов к чтению из него первого элемента.

Процедура Rewrite (var f) открывает логический файл f для последующей записи данных (открывает выходной файл). После успешного выполнения этой процедуры файл готов к записи в него первого элемента.

Процедура Close (var f) закрывает открытый до этого логический файл. Вызов процедуры Close необходим при завершении работы с файлом. Если по какой-то причине процедура Close не будет выполнена, файл все-же будет создан на внешнем устройстве, но содержимое последнего буфера в него не будет перенесено. Для входных файлов использование оператора закрытия файла необязательно.

Логическая функция EOF (var f): Boolean возвращает значение TRUE, когда при чтении достигнут конец файла. Это означает, что уже прочитан последний элемент в файле или файл после открытия оказался пуст.

Процедура Rename (var f; NewName: String) позволяет переименовать физический файл на диске, связанный с логическим файлом f. Переименование возможно после закрытия файла.

Процедура Erase (var f) уничтожает физический файл на диске, который был связан с файловой переменной f. Файл к моменту вызова процедуры Erase должен быть закрыт.

Функция IOResult: Integer возвращает целое число, соответствующее коду последней ошибки ввода - вывода. При нормальном завершении операции функция вернет значение 0. Значение функции IOResult необходимо присваивать какой - либо переменной, так как при каждом вызове функция обнуляет свое значение. Функция IOResult работает только при выключенном режиме проверок ошибок ввода - вывода или с ключом компиляции {$I-}.

Текстовые файлы.

Особое место в языке ПАСКАЛЬ занимают текстовые файлы, компоненты которых имеют символьный тип. Для описания текстовых файлов в языке определен стандартный тип Text:

```
var   TF1, TF2: Text;
```

Текстовые файлы представляют собой последовательность строк, а строки - последовательность символов. Строки имеют переменную длину, каждая строка завершается признаком конца строки.

С признаком конца строки связана функция EOLn(var T:Text):Boolean, где T - имя текстового файла. Эта функция принимает значение TRUE, если достигнут конец строки, и значение FALSE, если конец строки не достигнут.

Для операций над текстовыми файлами, кроме перечисленных, определены также операторы обращения к процедурам:

ReadLn(T) - пропускает строку до начала следующей;

WriteLn(T) - завершает строку файла, в которую производится запись, признаком конца строки и переходит к началу следующей.

Для работы с текстовыми файлами введена расширенная форма операторов ввода и вывода. Оператор Read(T,X1,X2,...XK) эквивалентен группе операторов

```
begin
   Read(T,X1);
   Read(T,X2);
   ..........
   Read(T,XK)
end;
```

Здесь T - текстовый файл, а переменные X1, X2,...XK могут быть либо переменными целого, действительного или символьного типа, либо строкой. При чтении значений переменных из файла они преобразуются из текстового представления в машинное. Оператор Write(T,X1,X2,...XK) эквивалентен группе операторов

```
begin
  Write(T,X1);
  Write(T,X2);
  ..........
  Write(T,XK)
end;
```

Здесь T - также текстовый файл, но переменные X1,X2,...XK могут

быть целого, действительного, символьного, логического типа или строкой. При записи значений переменных в файл они преобразуются из внутреннего представления в текстовый.

К текстовым файлам относятся стандартные файлы INPUT, OUTPUT. Рассмотренные ранее операторы ввода - вывода являются частным случаем операторов обмена с текстовыми файлами, когда используются стандартные файлы ввода - вывода INPUT, OUTPUT. Работа с этими файлами имеет особенности:

– имена этих файлов в списках ввода - вывода не указываются;

– применение процедур Reset, Rewrite и Close к стандартным файлам ввода - вывода запрещено;

– для работы с файлами INPUT, OUTPUT введена разновидность функции EOLn без параметров.

TURBO PASCAL вводит дополнительные процедуры и функции, применимые только к текстовым файлам, это SetTextBuf, Append, Flush, SeekEOLn, SeekEOF.

Процедура SetTextBuf(var f: Text; var Buf; BufSize: Word) служит для увеличения или уменьшения буфера ввода - вывода текстового файла f. Значение размера буфера для текстовых файлов по умолчанию равно 128 байтам. Увеличение размера буфера сокращает количество обращений к диску. Рекомендуется изменять разиер буфера до открытия файла. Буфер файла начнется с первого байта переменной Buf. Размер буфера задается в необязательном параметре BufSize, а если этот параметр отсутствует, размер буфера определяется длиной переменной Buf.

Процедура Append(var f: Text) служит для специального открытия выходных файлов. Она применима к уже существующим физическим файлам и открывает из для дозаписи в конец файла. Процедура Flush(var f: Text) применяется к открытым выходным файлам. Она принудительно записывает данные из буфера в файл независимо от степени его заполнения.

Функция SeekEOLn(var f: Text): Boolean возвращает значение True, если до конца строки остались только пробелы.

Функция SeekEOF(var f: Text): Boolean возвращает значение True, если до конца файла остались строки, заполненные пробелами.

Пример

```
{ Сформировать файл из некоторых чисел. }
{ Записать во второй файл количество положительных, }
{ отрицательных и нулевых элементов файла. }
uses CRT; { Подключение библиотеки ввода-вывода }
const n=10; { Максимальный размер массива }
{ Раздел объявления переменных }
var
File1, File2: FILE of Integer; { Переменные файлового типа }
Road1, Road2: String[14]; { Строки для хранения имен файлов }
a : Array[1..n] of Integer; { Массив для хранения введенных чисел }
i, pol, otr, nul: Integer; { i-счетчик цикла }
{ pol-количество положительных элементов массива}
{ otr количество отрицательных элементов массива}
{ nul количество нулевых элементов массива}
{ Основной блок программмы }
begin
ClrScr; { Очистка экрана }
WriteLn('Выполнил Поляков Д.Г., КТФ, гр. Р-11');
Road1:='Test1.dat'; { Задание имен файлов }
Road2:='Test2.dat';
Assign(File1,Road1); { Связь файловой переменной с внешним файлом Road1}
Rewrite(File1); { Создание и открытие файла Test1.dat }
```

```
Assign(File2,Road2); { Связь файловой переменной с внешним файлом Road2}
Rewrite(File2); { Создание и открытие файла Test2.dat }
pol:=0; { Инициализация переменных }
otr:=0;
nul:=0;
for i:=1 to n do
begin
WriteLn('Введите ',i,' элемент массива'); { Запрос у пользователя
элементов массива}
Read(a[i]); { Ввод элементов массива }
Write(File1,a[i]); { Запись их в файл Test1.dat }
end;
Close(File1); { Закрытие файла Test1.dat }
WriteLn; { Пропуск строки }
WriteLn('_____');
Reset(File1); { открытие файла Test1.dat для проверки правильности записи }
while not eof(File1) do { Чтение файла до тех пор, пока указатель }
{ текущей компоненты файла находится перед }
{ последней компонентой файла }
begin
Read(File1,a[i]); { Считывание чисел из файла }
Write(a[i]:2); { Распечатка считываемых чисел }
if a[i]>0 then pol:=pol+1; { Подсчет числа положительных, отрицательных}
if a[i]<0 then otr:=otr+1; { и нулевых элементов }
if a[i]=0 then nul:=nul+1;
Inc(i); { Приращение счетчика цикла }
```

Компонентный или типизированный файл – это файл с объявленным типом его компонент. Компонентные файлы состоят из машинных представлений значений переменных, они хранят данные в том же виде, что и память ЭВМ. Описание величин файлового типа имеет вид:

```
type M= File Of T;
```

где M - имя файлового типа, T - тип компоненты. Например:

```
type
  FIO= String[20];
  SPISOK=File of FIO;
var
  STUD, PREP: SPISOK;
```

Здесь STUD, PREP - имена файлов, компонентами которых являются строки.

Описание файлов можно задавать в разделе описания переменных:

```
var
   fsimv: File of Char;
   fr: File of Real;
```

Компонентами файла могут быть все скалярные типы, а из структурированных - массивы, множества, записи. Практически во всех конкретных реализациях языка ПАСКАЛЬ конструкция "файл файлов" недопустима. Все операции над компонентными файлами производятся с помощью стандартных процедур:

Reset, Rewrite, Read, Write, Close.

Для ввода - вывода используются процедуры: Read(f, X); Write(f, X); где f - имя логического файла, X - либо переменная, либо массив, либо строка, либо множество, либо запись с таким же описанием, какое имеет

компонента файла. Выполнение процедуры Read(f, X) состоит в чтении с внешнего устройства одной компоненты файла и запись ее в X. Повторное применение процедуры Read(f, X) обеспечит чтение следующей компоненты файла и запись ее в X.

Выполнение процедуры Write(f, X) состоит в записи X на внешнее устройство как одной компоненты. Повторное применение этой процедуры обеспечит запись X как следующей компоненты файла.

Для работы с компонентными файлами введена расширенная форма операторов ввода и вывода:

Read(f,X1,X2,...XK)

Write(f,X1,X2,...XK)

Здесь f - компонентный файл, а переменные X1, X2,...XK должны иметь тот-же тип, что и объявленный тип компонент файла f.

Бестиповые файлы позволяют записывать на диск произвольные участки памяти ЭВМ и считывать их с диска в память. Операции обмена с бестиповыми файлами осуществляется с помощью процедур BlokRead и BlockWrite. Кроме того, вводится расширенная форма процедур Reset и Rewrite. В остальном принципы работы остаются такими же, как и с компонентными файлами. Перед использованием логический файл

var f: File;

должен быть связан с физическим с помощью процедуры Assign и открыт для чтения или для записи процедурой Reset или Rewrite, а после окончания работы закрыт процедурой Close.

При открытии файла длина буфера устанавливается по умолчанию в 128 байт. TURBO PASCAL позволяет изменить размер буфера ввода - вывода, для чего следует открывать файл расширенной записью процедур

Reset (var f: File; BufSize: Word)

или

Rewrite (var f: File; BufSize: Word)

Параметр BufSize задает число байтов, считываемых из файла или записываемых в него за одно обращение. Минимальное значение BufSize – 1 байт, максимальное - 64 К байт.

Чтение данных из бестипового файла осуществляется процедурой

BlockRead(var f: File; var X; Count: Word; var QuantBlock: Word);

Эта процедура осуществляет за одно обращение чтение в переменную X количества блоков, заданное параметром Count, при этом длина блока равна длине буфера. Значение Count не может быть меньше 1. За одно обращение нельзя прочесть больше, чем 64 К байтов. Необязательный параметр QuantBlock возвращает число блоков (буфе ров), прочитанных текущей операцией BlockRead. В случае успешного завершения операции чтения QuantBlock = Count, в случае аварийной ситуации параметр QuantBlock будет содержать число удачно прочитанных блоков. Отсюда следует, что с помощью параметра QuantBlock можно контролировать правильность выполнения операции чтения.

Запись данных в бестиповой файл выполняется процедурой

BlockWrite (var f: File; var X; Count: Word; var QuantBlock: Word);

которая осуществляет за одно обращение запись из переменной X количества блоков, заданное параметром Count, при этом длина блока равна длине буфера. Необязательный параметр QuantBlock возвращает число блоков (буферов), записанных успешно текущей операцией BlockWrite.

Последовательный и прямой доступ к файлам.

Смысл последовательного доступа заключается в том, что в каждый момент времени доступна лишь одна компонента из всей последовательности. Для того, чтобы обратиться (получить доступ) к компоненте с номером К, необходимо просмотреть от начала файла К-1 предшествующую компоненту. После обращения к компоненте с номером К можно обращаться к компоненте с номером К+1. Отсюда следует, что процессы формирования (записи) компонент файла и просмотра (чтения) не

могут произвольно чередоваться. Таким образом, файл вначале строится при помощи последовательного добавления компонент в конец, а затем может последовательно просматриваться от начала до конца.

Рассмотренные ранее средства работы с файлами обеспечивают последовательный доступ. TURBO PASCAL позволяет применять к компонентным и бестиповым файлам, записанным на диск, способ прямого доступа. Прямой доступ означает возможность заранее определить в файле блок, к которому будет применена операция ввода - вывода. В случае бестиповых файлов блок равен размеру буфера, для компонентных файлов блок - это одна компонента файла.

Прямой доступ предполагает, что файл представляет собой линейную последовательность блоков. Если файл содержит n блоков, то они нумеруются от 1 через 1 до n. Кроме того, вводится понятие условной границы между блоками, при этом условная граница с номером 0 расположена перед блоком с номером 1, граница с номером 1 расположена перед блоком с номером 2 и, наконец, условная граница с номером n находится после блока с номером n.

Реализация прямого доступа осуществляется с помощью функций и процедур FileSize, FilePos, Seek и Truncate.

Функция FileSize (var f): Longint возвращает количество блоков в открытом файле f.

Функция FilePos(var f): Longint возвращает текущую позицию в файле f. Позиция в файле - это номер условной границы. Для только что открытого файла текущей позицией будет граница с номером 0. Это значит, что можно записать или прочесть блок с номером 1. После чтения или записи первого блока текущая позиция переместится на границу с номером 1, и можно будет обращаться к ьлоку с номером 2. После прочтения последней записи значение FilePos равно значению FileSize.

Процедура Seek(var f; N: Longint) обеспечивает назначение текущей позиции в файле (позиционирование). В параметре N должен быть задан

номер условной границы, предшествующей блоку, к которому будет производиться последующее обращение. Например, чтобы работать с блоком 4, необходимо задать значение N, равное 3. Процедура Seek работает с открытыми файлами.

Процедура Truncate(var f) устанавливает в текущей позиции признак конца файла и удаляет (стирает) все последующие блоки.

Пример. Пусть имеется текстовый файл ID.DAT, который содержит числовые значения действительного типа по два числа в каждой строке - значения аргумента и функции соответственно. Количество пар чисел не более 200. Составить программу, которая читает файл, значения аргумента и функции записывает в одномерные массивы, подсчитывает их количество, выводит на экран дисплея и записывает в файл компонентного типа RD.DAT.

```
Program F;
 var
  rArg, rF: Array[1..200] of Real;
    inf: Text;
    outf: File of Real;
    n, l: Integer;
 begin
   Assign(inf,'ID.DAT');
   Assign(outf,'RD.DAT');
   Reset(inf);
   Rewrite(outf);
   n:=0;
   while not EOF(inf) do
     begin
       n:=n+1;
       ReadLn(inf,rArg[n],rF[n])
     end;
```

```
 for l:=1 to n do
  begin
   WriteLn(l:2,rArg[l]:8:2,rF[l]:8:2);
   Write(outf,rArg[l], rF[l]);
  end;
 close(outf)
end.
```

Варианты заданий

1. Сформировать файл из некоторых чисел. Записать во второй файл количество положительных, отрицательных и нулевых элементов файла.

2. Сформировать два файла из некоторых чисел. Записать в третий файл положительные элементы первого и второго файлов.

3. Сформировать два файла из некоторых чисел. Записать в третий файл положительные элементы первого файла и отрицательные элементы второго.

4. Сформировать файл из некоторых чисел. Записать во второй файл положительные элементы первого файла, а в третий – отрицательные элементы первого файла.

5. Сформировать два файла из некоторых чисел. Записать в третий файл минимальный элемент первого файла и максимальный элемент второго.

6. Сформировать два файла из некоторых чисел. Определить, в каком файле больше положительных элементов и результат записать в третий файл.

7. Сформировать файл из некоторых чисел. Записать во второй файл элементы первого файла с четными номерами, в третий с нечетными.

8. Сформировать два файла из некоторых чисел. Записать в третий файл два первых элемента первого файла и два последних элемента второго.

9. Создать файл, содержащий сведения о месячной заработной плате рабочих завода. Каждая запись содержит поля - фамилия рабочего,

наименование цеха, размер заработной платы за месяц. Количество записей - произвольное.

10. Вычислить общую сумму выплат за месяц по цеху X, а также среднемесячный заработок рабочего этого цеха. Вывести ведомость для начисления заработной платы рабочим этого цеха.

11. Создать файл, содержащий сведения о количестве изделий, собранных сборщиками цеха за неделю. Каждая запись содержит поля - фамилия сборщика, количество изделий, собранных им ежедневно в течение шестидневной недели (в понедельник, вторник и т.д.). Количество записей - произвольное.

12. По каждому сборщику просуммировать количество деталей, собранное им за неделю. Определить сборщика, собравшего наибольшее число изделий, и день, когда он достиг наивысшей производительности труда.

13. Создать файл, содержащий сведения о количестве изделий категорий A, B, C, собранных рабочим за месяц. Структура записи имеет поля - фамилия сборщика, наименование цеха, количество изделий по категориям, собранных рабочим за месяц. Количество записей - произвольное.

14. Считая заданными значения расценок Sa, Sb, Sc за выполненную работу по сборке единицы изделия категорий A, B, C соответственно, подсчитать:

- общее количество изделий категорий A, B, C, собранных рабочим цеха X;

- ведомость заработной платы рабочих цеха X;

- средний размер заработной платы работников этого цеха.

15. Создать файл, содержащий сведения о телефонах абонентов. Каждая запись имеет поля - фамилия абонента, год установки телефона, номер телефона. Количество записей - произвольное.

16. По вводимой фамилии абонента выдать номер телефона. Определить количество установленных телефонов с XXXX года. Номер года вводится с терминала.

17. Создать файл, содержащий сведения о сдаче студентами первого курса сессии. Структура записи - индекс группы, фамилия студента, оценки по пяти экзаменам, признак участия в общественной работе: "1" - активное участие, "0" - неучастие.

18. Зачислить студентов группы X на стипендию. Студент, получивший все оценки "5" и активно участвующий в общественной работе, зачисляется на повышенную стипендию (доплата 50 %), не активно участвует - доплата 25 %. Студенты, получившие "4" и "5" , зачисляются на обычную стипендию. Студент, получивший одну оценку "3", но активно занимающийся общественной работой, также зачисляется на стипендию, в противном случае зачисление не производится. Индекс группы вводится с терминала.

19. А. Создать файл, содержащий сведения о сдаче студентами сессии. Структура записи - индекс группы, фамилия студента, оценки по пяти экзаменам и пяти зачетам ("З" означает зачет, "Н" - незачет). Количество записей - 25.

20. Определить фамилии неуспевающих студентов с указанием индексов групп и количества задолженностей. Найти средний балл, полученный каждым студентом группы X, и всей группой в целом.

21. Создать файл, содержащий сведения о личной коллекции книголюба. Структура записи - шифр книги, автор, название, год издания, местоположение (номер стеллажа, шкафа и т.д.). Количество записей - произвольное.

Б. Найти:

1) местонахождение книги автора X названия Y;

2) список книг автора Z, находящихся в коллекции;

3) число книг издания XX года, имеющееся в библиотеке.

САМОСТОЯТЕЛЬНАЯ РАБОТА СТУДЕНТОВ

В качестве самостоятельной работы по дисциплине «Информатика» студентам предлагается рассмотреть и изучить следующие вопросы:

1. Операционная система MS DOS, оболочка NC, операционная система WINDOWS.
2. Работа с текстовыми, табличными редакторами, базами данных - основные возможности.
3. Глобальные и локальные сети. Работа в Интернет, основные услуги, предоставляемые сетями. Посещение Интернет-центра.
4. Среда программирования Турбо Паскаль. Функциональные возможности меню.
5. Методы сортировки массивов.
6. Численные методы решения уравнений (половинного деления, касательных, Ньютона, простой итерации).
7. Приближенное интегрирование функции (метод прямоугольников, метод трапеций).
8. Типы данных (стек, очередь).

ТЕМЫ РЕФЕРАТОВ ПО ДИСЦИПЛИНЕ

1) История Windows?
2) Что такое Internet? Классификация доменов. Почта, поисковые системы.
3) Сетевые технологии, TCP/IP, IP, MAC – адрес?
4) Языки программирования их классификация и назначение?
5) Развитие компьютерного оборудования с 80гг? Предполагаемые направления развития оборудования.
6) Базы данных. Сетевые, иерархические, реляционные.
7) История информатики. История Windows?
8) Программы для ОС Windows, их назначение, классификация.

9) Ассемблер основы программирования.
10) Нейронные сети, экспертные системы, системы реального времени.
11) Структура Windows? Компоненты, реестр.
12) BIOS, начальная загрузка Windows 98, Me, XP.
13) Организация хранение данных в ПК, серверах. Особенности RAID, SCSI, IDE и др.
14) Компьютерные вирусы, механизмы заражения.
15) Антивирусы, сравнение, принципы защиты.
16) Сетевое оборудование

ЛИТЕРАТУРА

1. Фаранов В.В. Turbo Pascal. – СПб: BHV, 2003. – 1056 с.: ил. ISBN: 5-94157-295-6.

2. Гусева А.И. Учимся информатике. Задачи и методы их решения. - М: Диалог-Мифи, 2001.

3. Зуев Е.А. Программирование на языке Turbo Pascal 6.0, 7.0.-М.: Радио и связь, Веста, 2003.-380с.

4. Емелина Е.И. Основы программирования на языке Паскаль.-М.: Финан¬сы и статистика, 1999.- 208с.

5. Епанешников А.М. Программирование в среде Turbo Pascal 7.О.- М.: Диалог-Мифи, 2000.-288с.

6. Королев Л. Н., Миков А. И. Информатика. Введение в компьютерные науки.-М.:Высшая школа, 2003.

7. Могилев А. В., Пак Н. И., Хеннер Е. К. Информатика.-М.: Академия, Серия: Высшее образование ("Академия"), 2004. Е. К.

8. Сафронов И. К. Задачник-практикум по информатике.-СПб.:ВНУ-СПб, 2002.

9. Семакин И.Г., Хеннер Е.К. Информатика. Задачник-практикум.-М.: Ла¬боратория Базовых Знаний, Том 2, 2002.

10. Стариченко Б.Е. Теоретические основы информатики. Учебное пособие для вузов.-М: Горячая Линия - Телеком, Серия: Специальность для высших учебных заведений, 2003.

11. Абрамов С.А., Зима В.С. Начало программирования на языке Паскаль.-

М.: Наука, 1987.- 112с.

12. Вирт Н. Алгоритмы + структура данных = программы : перевод с англ.-М.Мир,1985.-406с.

13. Левин А. Самоучитель работы на компьютере. - М. Международное агенство A.D.&T.,1997 - 480с.

14. Перминов О.Н. Язык программирования Паскаль. - М.: Радио и связь, 1983.

15. Петров А.В. и др. Вычислительная техника в инженерных и экономических расчетах.-М.: Высшая школа, 1990. - 478с.

16. Фигурнов В. IBM PC для пользователя.-С.Петербург, 1997г.

17. Франклен Г. MS DOS 5.0 для пользователя.- Киев: Торгово- издательское бюро BHV, 1992.

18. Гордиенко А.М., Рогов Д.К. Численные методы проектирования на ЭВМ./ Методические указания, М., 1990. - 31с.

19. Ракитин В.И., Первушин В.Е. Практическое руководство по методам вычислений. - М.: Выс.шк., 1998. - 383с.

20. Марченко А.И., Марченко Л.А. Программирование в среде Turbo Pascal 7.0. – М.: Бином Универсал, К.: ЮНИОР, 1997. – 496 с.

21. А.М. Епанешников, В.А. Епанешников. Программирование в среде Turbo Pascal 7.0. – М.: «ДИАЛОГ-МИФИ», 1998. – 367с.

22. Фаронов В.В. Turbo Pascal 7.0. Начальный курс. – М.: «Нолидж», 1999. – 616 с.

23. Турбо Паскаль в примерах: Методические указания для студентов (в 2-х частях). Часть 1/ Составитель Афанасьева Т.В. – Ульяновск, 1997.

24. Д. Кнут. Искусство программирования для ЭВМ. Т. 3. Сортировка и поиск. – М.: Мир, 1978. – 848 с.