## §3. ПАСКАЛЬ КАК ЯЗЫК СТРУКТУРНО-ОРИЕНТИРОВАННОГО ПРОГРАММИРОВАНИЯ

### 3.1. ВВЕДЕНИЕ

Язык Паскаль, начиная с момента своего создания Н.Виртом в 1971 г., играет особую роль и в практическом программировании, и в его изучении. С непревзойденной четкостью в нем реализованы принципы структурного программирования, которые мы рассматривали в п. 1.8. Паскаль стал первым языком, с которым знакомится большинство будущих программистов в мире.

Трансляторы для программ, написанных на Паскале, разработаны для различных компьютеров и в настоящее время имеют множество разновидностей. Они являются компиляторами, обрабатывающими разработанные программистами тексты программ.

Существует много версий языка Паскаль. Различия между ними порой весьма велики. Так, базовая версия Вирта имеет многократно меньшие возможности чем, скажем, версия Турбо-Паскаль 7.0 (первая, фактически - язык для обучения будущих программистов, а вторая - орудие профессиональных разработчиков прикладного программного обеспечения). Тем не менее, это версии одного языка, что, в частности, подтверждается их совместимостью «сверху вниз», т.е. любая программа, соответствующая «младшей» версии, соответствует и «старшей» (за исключением малозначащих синтаксических оговорок). Приведенные ниже тексты программ и примеры соответствуют (если нет специальных оговорок) версиям не ниже Турбо-Паскаль 3.0.

Любая Паскаль-программа является текстовым файлом с собственным именем и с расширением .pas. Рассмотрим в качестве примера текст программы 1 решения квадратного уравнения. Паскаль-программа имеет вид последовательности символов латинских и русских букв, арабских цифр, знаков операции, скобок, знаков препинания и некоторых дополнительных символов. В ней можно выделить описания данных и операторы, описывающие действия, которые надо выполнить машине над этими данными.

*Программа I*

program KvadUravn; (заголовок программы)

var (список переменных)

а,b,с: real; (коэффициенты уравнения)

d,xlx2: real; (вспомогательные переменные)

begin (начало программы)

writein; (пропуск строки на экране)

writein('введи a,b,c'); read(а,Ь,с); (ввод данных)

d:=b\*b-4\*a\*c; (дискриминант)

if d<0 then (если d<0, то)

write('корней нет') (печатать)

else (иначе)

begin (начало серии команд)

xl:=(-b+sqrt(d))/(2\*a);

x2:=(-b-sqrt(d))/(2\*а); (вычисляем корни)

write('х1=',х1,' х2=',х2) (печать корней)

end (конец серии)

end.(конец программы)

Схематически программа представляется в виде последовательности восьми разделов:

1) заголовок программы;

2) описание внешних модулей, процедур и функций;

3)описание меток;

4) описание констант;

5) описание типов переменных;

6) описание переменных;

7) описание функций и процедур;

8) раздел операторов.

Не в каждой программе обязательно присутствуют все восемь разделов, в простейшей программе, например, могут быть только 5-й и 8-й разделы.

Каждый раздел начинается со служебного слова, назначение которого зафиксировано в Паскале так, что его нельзя употреблять для других целей (список и перевод служебных слов дан в Приложении).

Рассмотрим пример программы 2, вычисляющей длину окружности и площадь круга по данному радиусу.

*Программа 2*

program circle;

const

pi=3.14159;

var

r,s,1 : real;

begin

writeln (введите радиус'); readln(r);

s:=pi\*r\*r; l:=2\*pi\*r;

writeln('площадь круга=',S:8:4); writeln('длина окружности=', l:8:4)

end.

В этой программе можно выделить четыре раздела. Описание заголовка начинается со служебного слова program, описание констант - const, описание переменных -var, раздел операторов начинается с begin. Программа заканчивается служебным словом end, после которого ставится точка. Описания величин и операторы друг от друга отделяются знаком «точка с запятой».

Для обозначения величин используются имена. Они составляются из латинских букв и цифр, причем первым символом должна быть буква. В примере использованы имена величин - pi, r,s и 1.

Имя программы (в примере - circle) выбирается автором и составляется по такому же правилу.

Постоянные величины (константы) чаще всего бывают числовыми или символьными (но могут быть и других типов, о которых речь пойдет ниже). Значения символьных констант заключаются в апострофы.

Постоянные величины описываются в разделе констант по схеме:

const <имя> = <константа>

В разделе констант может быть описано несколько постоянных величин.

Например:

const

pi=3.14159; k=-15; s='площадь';

Данные, обрабатываемые программой, могут быть разных типов (числа, символы, строки, массивы и т.д.). Тип определяет область допустимых значений, а также операции и функции, применяемые к величинам этого типа. В Паскале имеется несколько встроенных простых типов со стандартными именами.

Группа типов, значения каждого из которых можно перечислить в некотором списке **- скалярные типы.** Для них определена порядковая функция ord(x) - номер значения х в списке (для целочисленного х ord(x)=x); функции pred(x) - значение в списке, предшествующее х, и suce(x) - значение в списке, следующее за х.

**Упорядоченный тип** - это тип, значения которого упорядочены в обычном смысле. К данным такого типа применимы операции отношения <, **>, <=** (меньше или равно), >=(больше или равно), *=,* <> ( не равно). Для логического типа выполняется неравенство:

false < true.

Переменные описываются в разделе описания переменных посхеме:

var <список имен переменных>: <тип>

Имена в списке разделяются запятой. В этом разделеможет быть описано несколько переменных разного типа, например:

var a,b,c:real; k,l:integer; p:boolean;

Над **целыми величинами** (тип integer) определены арифметические операции : \* (умножение), div (деление нацело), mod (вычисление остатка от деления), + , -(сложение и вычитание); операции перечислены в порядке старшинства.

Например: 25 div 4 *= 6 ;* 25 mod 4=1. Целый результат дают некоторые стандартные функции (аргумент функции заключается в круглые скобки):

abs(x) - абсолютная величина целого х;

sqr(x) -квадрат значения х;

trunc(x) - целая часть вещественной величины х;

round(x) - целое число, полученное из вещественного х по правилу округления;

random(x) - случайное целое числоиз интервала от 0 до х.

Например: trunc(4.7)=4 ; round(4.7)=5 ; sqr(3)=9 . Для данных типа byte определены те же операции и функции, что и для данных типа integer.

Над **вещественными величинами** определены операции: \*, +, -, /, а также стандартные функции при вещественном или целом аргументе: abs(x), sqr(x), sin(x), cos(x), arctan(x), ln(x), exp(x), sqrt(x) - квадратный корень из х, int(x) - целая частьизх, random - случайное число из интервала от 0 до 1. Указанные операции и функции дают вещественный результат.

Над **логическими величинами** определены операции: not - отрицание, and -конъюнкция, or-дизъюнкция. Логическая функция odd(x) принимает значение true, если целочисленное х является нечетным и false , если четным.

Множество всех символов образуют **символьные величины** (тип char), которые являются упорядоченными, причем 'А' < 'В'< 'С' **<...<** 'Z', 'а' < 'b' **<...<** 'z', '0' < 'I' **<...<** '9'.

**Выражения** - это конструкции, задающие правила вычисления значений переменных. В общем случае выражения строятся из переменных, констант, функций с помощью операций и скобок.

Эта роль выражений отражена в основном операторе языка - **операторе присваивания.** Он имеет следующий вид:

<имя переменной> := <выраженне>

Тип переменной и тип выражения должны быть согласованы (величины принадлежат одному и тому же типу). Есть исключение: имя переменной может относиться к типу real, а значение выражения - к типу integer.

*Примеры.*

l:=2\*pi\*r; p:=(a+b+c)/2; z:=sqrt(sqr(x)+sqr(y))

В Паскале можно вводить с клавиатуры числовые и символьные данные. Имеются две встроенные процедуры (подпрограммы) ввода:

1. read(<cписок переменных>); 2) readln(<cписок переменных>).

При выполнении процедуры read(xl,x2,...,xN) программа прерывается и компьютер ждет ввода с клавиатуры N значений переменных из списка х1, х2,..., xN. Эти значения - константы соответствующих типов - должны при вводе разделяться пробелами. Набор данных завершается клавишей ввода.

Процедура readin отличается от read только тем, что при завершении ввода курсор перемещается в начало следующей строки.

*Пример.*

var a,b:real; c:char; d:integer;

...

read(a,c,d,b);

...

Допустимый ввод: 83.14 k 200-7.15

Программа на Паскале может выводить на экран или на принтер значения числовых или символьных выражений. Имеются две процедуры вывода на экран:

1) write(<cписок выражений>); 2) writeln(<cписок выражений>).

Процедура write(xl,x2,...,xN) печатает на экране значения выражений из списка х1, х2, ..., xN. Процедура writeln отличается от write тем, что переводит курсор в начало следующей строки. Для вывода на принтер используются те же процедуры с добавлением служебного слова 1st перед списком выражений.

Пример: write(lst,'нет решений');

На бумаге будет напечатан текст «нет решений».

Для управления печатью используются **форматы данных.** Пусть х - переменная типа real. Если не использовать форматы, то значение х будет выводиться в «плавающей» форме (типа 1.654887892Е-04). Форматы позволяют напечатать вещественное число в естественной форме. Пусть m, n - целые числа. Процедура write(x:m:n) выводит на экран значение переменной х в виде десятичной дроби, причем m определяет общее число выводимых символов, включая цифры, точку и знак числа, n - количество цифр после точки. Если количество выводимых символов меньше m, то перед числом добавляются пробелы.

Пусть, например, х = 387.26. Следующая таблица демонстрирует влияние форматов на вывод значения х:

оператор строка вывода

writeln('\*',x) \* 3.8726000000E+02

writeln('\*',x:8:3) \* 387.260

writeln('\*',x:8:l) \* 387.3

Один формат - ширину поля вывода - можно использовать и для вывода значений выражений типов integer, boolean, char.

### *Контрольные вопросы*

1. Какова последовательность разделов в программе на Паскале?

2. Какие типы данных называются скалярными? упорядоченными?

3. Какие действия допустимы над величинами целого типа? вещественного типа?

4. Как в Паскале осуществляется ввод и вывод данных?

### 3.2. ОСНОВНЫЕ КОНСТРУКЦИИ ЯЗЫКА

Паскаль - язык структурного программирования. Это означает, что программист должен выражать свои мысли очень дисциплинированно, с использованием малого числа четко оговоренных конструкций, используя как чередование их, так и вложения друг в друга. Не рекомендуется (хотя и возможно) использовать оператор перехода goto.

Реализация последовательности действий (т.е. структуры следования) выполняется с помощью составного оператора:

begin <последовательность операторов> end

Раздел операторов в программе всегда является составным оператором. Служебные слова begin и end часто называют операторными скобками.

Для реализации **развилки** в Паскале предусмотрены два оператора: условный оператор и оператор варианта (выбора). Они предназначены для выделенияизсоставляющих их операторов одного, который и выполняется.

Структура и действие **условного** **оператора** таковы:

If <логическое выраженне>

then <оператор 1>

else <оператор 2>

Условный оператор может быть неполным,т.е. не содержать часть «else <оператор 2>». В этом случае, если значение логического выражения равно false, условный оператор не вызывает никаких действий.

*Пример:* составим программу, которая определяет длину общей части двух отрезков числовой оси, заданных координатами своих концов соответственно а, b и с, d (а < b, с < d). Если отрезки имеют общую часть, то левая координата общей части отрезков m равнамаксимальному из чисел а и с, а правая n - минимальному из чисел b и d.

*Программа 3*

program cross;

var a,b,c,d,m,n,l:real;

begin

writeln('введите координаты концов отрезков') ;

read(a,b,с,d) ;

writeln;

if a<c then m:=c else m:=a;

if b<d then n:=b else n:=d;

if m<n then l:=n-m else 1:=0;

writeln("длина общей части отрезков=',1:6:2)

end.

**Оператор варианта** имеет следующую форму:

case <выражение> of

<список констант 1> : <оператор 1>;

<список констант 2> : <оператор 2>;

…………………………

<список констант N> : <оператор N>

end.

Выражение, стоящее между служебными словами case и of, должно иметь значение ординального типа. Любой список констант может состоять из одной константы.

Оператор варианта вычисляет значение выражения, записанного после case. Если его значение совпадает с одной из констант в некотором списке, то выполняется оператор, стоящий после этого списка. Если значение выражения не совпало ни с одной константой во всех вариантах, то оператор варианта ничего не делает.

В качестве примера приведем программу, которая в зависимости от номера месяца выдает сообщение о времени года.

*Программа 4*

program seasons;

var k:integers

begin

writeln('введите номер месяца') ;

readin(k);

case k of

1,2,12:writeln('зима');

3, 4,5:writeln('весна') ;

6, 7,8:writeln('лето');

9,10,11:writeln('осень')

end

end.

Для реализации циклов в Паскале имеются три оператора Если число повторений известно заранее, то удобно воспользоваться оператором цикла с параметром. В других случаях следует использовать операторы цикла с предусловием (цикл «пока» ) или с постусловием (цикл «до»).

**Цикл с предусловием** является наиболее мощным в Паскале. Другие операторы цикла можно выразить через него. Его форма такова:

while - <логическое выражение> do <оператор>

Действие: вычисляется значение логического выражения. Если оно равно true, то выполняется оператор, после чего снова вычисляется значение логического выражения, в противном случае действие заканчивается.

В качестве примера использования такого цикла решим следующую задачу. На склад привозят однородный груз на машинах различной грузоподъемности. На компьютер, управляющий работой склада, поступает информация о весе груза очередной машины. Составить программу подсчета количества машин,прибывшихна склад до его заполнения, если вместимость склада не более 100 тонн.

Введем обозначения: sum - сумма веса груза, хранящегося в этот момент на складе; num - число разгруженных машин; w - масса груза очередной машины. Вначале величины sum и num равны нулю. Цикл разгрузки продолжается, пока выполняется неравенство sum < 100.

*Программа 5*

program store;

var sum,w:real;

num:integer;

begin

num:=0;sum:=0;

while sum<100do

begin

writeln('введите вес груза машины');

readln(w);

sum:=sum+w;

if sum<=100 then num:=num+l

else writeln('груз уже не поместится')

end;

writeln('число разгруженных машин =',num;3)

end.

Оператор **цикла с постусловием** имеет форму:

repeat <последовательность операторов>

until <логическое выражение>

Действие: выполняется последовательность операторов. Далее вычисляется значение логического выражения. Если оно равно true, то действие заканчивается, в противном случае снова выполняется последовательность операторов и т.д.

Решим предыдущую задачу, применяя цикл с постусловием. Цикл разгрузки заканчивается, если выполняется условие: sum > 100.

*Программа 6*

program store2;

var sum,w:real; man:integer;

begin

num: =0; sum: =0;

repeat

writeln('введите вес груза машины');

readln(w); sum:=sum+w;

if sum<=100 then num:=num+l .

else writeln('груз ухе не поместится')

until sum>=100;

writeln('количество разгруженных машин =',num:3)

end.

Оператор **цикла с** **параметром** предусматривает повторное выполнение некоторого оператора с одновременным изменением по правилу арифметической прогрессии значения управляющей переменной (параметра) этого цикла. Оператор цикла с параметром имеет две формы.

Форма1:

for <параметр>:= <выражение1> to <выражение 2> do <оператор>

Параметр, выражение 1, выражение 2 должны быть одного ординального типа;

Параметр в этом цикле возрастает. Действие эквивалентно действию следующего составного оператора:

begin

<параметр>:=<выражение 1>;

while <параметр> <= <выражение 2> do

begin

<оператор>;

<параметр>:=suсс(<параметр>)

end

end.

Если в этом описании отношение <= заменить на >= , а функцию succ на pred, то параметр в цикле будет убывать, в этом случае цикл с параметром принимает форму 2.

Форма 2:

for <параметр>:=<выражение 1> downto <выражение 2> do <оператор>

*Пример:* составим программу, по которой будет напечатана таблица перевода километров в мили (1 миля = 1,603 км). Параметром цикла можно считать целую величину k - количество километров. Пусть эта величина изменяется от 1 до 10 (с шагом 1, разумеется).

*Программа 7*

program mili;

const a=1.603;b='км'; с='мили';

var k:integer; m:real;

begin

writeln(b:5,c:7); writeln;

for k:=l to 10 do

begin

m:=k/a; writeln(k:5,m:6:3)

end

end.

Запишем в этой программе цикл с параметром в форме 2:

for k:=10 downto 1 do

begin

m:=k/a; writeln(k:5,m:6:3)

end.

Тогда значения k в таблице будут убывать от 10 до 1 с шагом 1.

### *Контрольные вопросы*

1. Как в Паскале реализуется развилка?

2. В чем различие в назначениях условногооператора и оператора варианта? Как реализуется оператор варианта?

3. Какого типа циклы реализуются в языке Паскаль? каким образом?

### 3.3. СТРУКТУРЫ ДАННЫХ

Мы уже познакомились с простыми типами real, integer, boolean, byte,char.

В Паскале программист по своему желанию может определить новый тип путем перечисления его элементов - перечисляемый тип, который относится *к простым ординальным типам.*

Описание перечисляемого типа выполняется в разделетипов по схеме:

type <имя типа> = <список имен>

Примеры:

type operator = (plus,minus,multi, divide);

color = (white,red,blue,yelow,purple,green);

В списке должно быть не более 256 имен.

Поскольку перечисляемый тип относится к ординальным, то к его элементам можно применять функции ord(x), pred(x), succ(x) и операции отношения. Отметим, что данные этого типа не подлежат вводу и выводу с помощью функций ввода/вывода и могут использоваться внутри программы для повышения ее читабельности и понятности.

Для иллюстрации работы с перечисляемыми типами приведем программу 8, переводящую английские названия дней недели на русский язык.

Программа 8

programweek;

type days=(mon,tue,wed,thu,fri,sat,sun);

var d:days;

begin

for d:=mon to sun do

case d of

mon: writeln("понедельник");

tue writeln("вторник");

wed writeln("среда");

thu writeln("четверг");

fri writeln("пятница");

sat writein("суббота") ;

sun writeln("воскресенье")

end

end.

**Интервальный тип** - это подмножество другого уже определенного ординального типа, называемого *базовым.* Интервал можно задать в разделе типов указанием наименьшего и наибольшего значений, входящих в него и разделяющихся двумя последовательными точками, например:

type days=(mon,tue,wed,thu,fri,sat,sun);

workday s=mon.. fri;

index=1..30;

letter='a'..'z';

Можно задать интервал и в разделе переменных:

vara:1..100;b:-25..25;

Операции и функции - те же, что и для базового типа. Использование интервальных типов в программе позволяет экономить память и проводить во время выполнения программы контроль присваиваний.

*Пример:* если k - номер месяца в году, то вместо описания

var k:integer;

можно написать

vark:1..12;

Интервальный тип тоже относится к простым ординальным типам.

СОСТАВНЫЕ СТРУКТУРЫ. Данные, с которыми имеет дело ЭВМ, являются абстракциями некоторых реальных или существующих в воображении людей объектов. Мы уже познакомились с некоторыми типами данных: стандартными, перечислимыми, интервалами. Этими типами можно было обойтись при решении простых задач. Однако естественно и часто очень удобно группировать однотипные данные в последовательности - массивы, строки символов, объединять разнотипные данные об одном и том же объекте в виде записей. Значительные удобства представляются пользователю в Паскале при организации однотипных величин в виде множества с соответствующим набором операций: объединения, пересечения и т.д. Наконец, последовательность однотипных величин переменной длины можно представить в Паскале в виде файла данных и хранить на внешних носителях, используя его в разных программах.

Таким образом, подобно составному оператору, содержащему несколько других операторов, составные типы образуются из других типов, при этом существенную роль играет метод образования или структура составного типа.

Часто используемый составной тип - массив. *Массив -* это последовательность, состоящая из фиксированного числа однотипных элементов. Все элементы массива имеют общее имя (имя массива) и различаются индексами. Индексы можно вычислять, их тип должен быть ординальным. При описании массивов используются служебные слова array и of. В описании массива указывается тип его элементов и типы их индексов.

Схема описания такова:

type <имя типа> = array [<список типов индексов>] оf <тип элементов>

Тип элементов - произвольный, он может быть составным. Число типов индексов называется размерностью массива. После описания типа массива конкретные массивы можно задать в разделе описания переменных.

Например:

type vector = array [1.. 10] of real;

table = array ['A'..'Z',1..5] of integer;

var a,b : vector;

с: table;

Обращение к элементу массива осуществляется с помощью задания имени переменной, за которым следует заключенный в квадратные скобки список индексов элемента.

Например:

а[7]:=3.1; b[k\*k+l]:=0; с['М',3]:=-14;

Если массивы имеют одно и то же описание, то во многих версиях Паскаля допустимо их копирование, например b:=а;

Описание массива можно совместить с описанием соответствующих переменных:

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

d : array [byte] of char;

В Турбо-Паскале разрешена инициализация начальных значений составных переменных с помощью, так называемых, типизированных констант. Типизированные константы используются как переменные того же типа. Их применение экономит память, однако они не могут быть использованы для определения других переменных того же типа.

Схема описания констант массива:

const <имя массива>: <тип массива> = (<список значений элементов>)

Тип массива может быть описан ранее:

type digits *=* array [1 ..5] of char;

const a : digits =('0';2','4';6';8');

*Пример:* используя массив, составим программу, которая напечатает на экране 20 чисел Фибоначчи.

Последовательность Фибоначчи определяется равенствами

а[1]=а[2]=1; a[k]=a[k-l]+a[k-2] прик>2.

Использование массива позволяет создать эффективную программу. Для вывода каждого члена последовательности отведем на экране 5 позиций.

*Программа 9*

program fibon; *"''*

const n=20;

var a: array[l..n] of integer;

**k:** integer;

begin

a[l]:=l;a[2]:=l;

for k:=3 to n do a[kl:=a[k-l]+a[k-2];

for k:=l to n do write(a[k]:5);

writeln

end.

Рассмотрим часто встречающуюся задачу упорядочения членов числовой последовательности по какому-либо признаку.

*Пример:* упорядочить члены числовой последовательности по возрастанию.

Используем метод упорядочения, носящий имя «пузырек». Будем просматривать пары соседних элементов последовательно справа налево и переставлять элементы в

паре, если они стоят неправильно:

5,3,2,4,1 → 5,3,2,1,4 → 5,3,1,2,4 →5,1, 3,2,4 → 1,5,3,2,4

В начале просмотра присвоим некоторой логической переменной значение true:

p:=true; если при просмотре пар была хотя бы одна перестановка, изменим значение логической переменной на противоположное: p:=false; это означает, что последовательность еще не была упорядочена и просмотр пар надо повторить. Цикл просмотров заканчивается, если после очередного просмотра выполняется условие: p=true. Последовательность зададим в программе как константмассив **из** 10 элементов - целых чисел.

*Программа 10*

program bubble; '

const a:array[l..10] of integer=(19,8,17,6,15,4,13,2,11,0);

var b,i:integer; p :boolean;

begin c1rscr;

for i:=l to 10 do write(a[i]:3);writeln;writeln;

repeat p:=true;

for i:=10 downto 2 do

if a[i]<a[i-l]

then begin

b:=a[il;a[i]:=a[i-l]; ali-1]:=b;p:=false

end

until p=true;

for i:=l to 10 do write(a[i]:3);

writeln

end.

Обработка элементов двумерных массивов (матриц) обычно выполняется с помощью двойного цикла. Один цикл управляет номером строки, другой - номером столбца.

При решении задач на ЭВМ часто возникает необходимость в использовании последовательностей символов. Такую последовательность можно описать как массив символов, однако в Паскале для таких целей имеется специальный тип - string[n] - ***строка*** из n символов, где n <= 255. Способы описания переменных - строк - аналогичны описанию массивов.

1. Строковый тип определяется в разделе описания типов, переменные этого типа - в разделе описания переменных:

type word : string[20];

var a,b,c : word;

2. Можно совместить описание строкового типа и соответствующих переменных в разделе описания переменных:

var a,b,c : string[20];

d : string[30];

3. Можно определить строковую переменную и ее начальное значение какконстант-строку:

const l:string[l 1]='информатика';

Символы, составляющие строку, занумерованы слева направо; к ним можно обращаться с помощью индексов, как к элементам одномерного массива.

Для переменных одного строкового типа определен лексикографический порядок, являющийся следствием упорядоченности символьного типа:

'fife' < 'tree' (таккак 'f' **<** 't'); '4' > '237' (так как '4' > '2'*).*

Кроме логических операций **<,** >, =, для величин строкового типа определена некоммутативная операция соединения, обозначаемая знаком плюс:

а:='кол'+'о'+'кол'; (в результате а='колокол').

Для строковых величин определены следующие четыре стандартные функции.

1. Функция соединения - concat(sl,s2,...,sk). Значение функции - результат соединения строк sl ,s2,...sk, если он содержит не более 255 символов.

2. Функция выделения - copy(s,i,k). Из строки s выделяется k символов, начиная с i-того символа:

а:=сору('крокодил',4,3); (в результате а='код\*).

3. Функция определения длины строки - length(s). Вычисляется количество символов, составляющих текущее значение строки s:

b:=length('каникулы'); (b=8).

4. Функция определения позиции - pos(s,t). Вычисляется номер позиции, начиная с которого строка s входит первый раз в строку t; результат равен 0, если строка s не входит в t:

с:=роs('ом','компьютер'); (с=2).

В Паскале определены также четыре стандартные процедуры для обработки строковых величин:

1. Процедура удаления delete(s,i,k).Из строки s удаляется k символов, начиная с i-того символа.

s:='таракан'; delete(s,5,2); (в результате S='таран').

2. Процедура вставки - insert(s,t,i). Строка s вставляется в строку t, начиная с позиции i:

t:='таран'; insert ('ka',t,5); (t='таракан').

3. Процедура преобразования числа в строку символов - str(k,s). Строка s получается «навешиванием» апострофов на число k:

str(564,s); (s='564').

4. Процедура преобразования строки из цифр в число - val(s,k,i). Число i=0, если в строке s нет символов, отличных от цифр, в противном случае i=позиции первого символа, отличного от цифры:

val('780',k,i); (k=780; i=0).

Рассмотрим несколько программ, в которых используются строковые величины.

1. Составить программу, определяющую количество гласных в русском тексте, содержащем не более 100 символов.

Здесь удобно определить констант-строку, состоящуюиз всех 18 строчных и заглавных русских букв, и в цикле проверить, будет ли очередной символ заданного текста элементом констант-строки.

*Программа 11*

program vowel;

**const** с:зtring[18]='аеиоуыэюяАЕИОУЫЭЮЯ' ;

var a :string[100]; k,n:integer;

begin

writeln('введите текст'); readln(a);n:=0;

for k:=l to length(a) do

if pos(a[k],c)>0 then n:=n+l;

writeln('кол. гласных=',n) end.

2. Заменить в арифметическом выражении функцию sqr на ехр. Замена выражения sqr на ехр достигается последовательным применением процедур delete и insert:

*Программа 12*

program stroka;

var a,b:string[40]; k:integer;

begin

writeln('введите строку <= 40 символов');

readin(a);b:=a;

repeat k:=pos('sqr',b);

if k>0 then

begin

delete(b,k,3);insert('ехр',b,k) ;

end

until k=0;

writein('старая строка=',a); writein('новая строка"',b);

end.

3. Ввести и упорядочить по алфавиту 10 латинских слов. В программе определим массив **из** 10 элементов-строк и упорядочим его элементы методом пузырька.

*Программа 13*

program order;

const s=10;

type word=string(20] ;

**var** i, j, k **: 1. . s;**

b:word; p:boolean; list :array[l..s] of word;

begin

clrscr;writeln<'введите список слов');

for i:=l to s do readln(list[i]);

repeat p:=true;

for i:=s downto 2 do

if list[i]<list[i-l] then

begin

b:list[i);list[i]:=list[i-l]; list[i-l]:=b;p:=false end

until p=true;

writein('упорядоченныйсписок слов:');

for i:=l to s do writeln(list[i])

end.

*Множество* в Паскале имеет такой же смысл, как и в алгебре - это неупорядоченная совокупность отличных друг от друга однотипных элементов. Число элементов множества не должно превышать 255. В качестве типа элементов может быть любой скалярный тип, кроме типа integer и его интервалов, содержащих числа > 255. Тип элементов множества называется базовым. При описании множественного типа используются служебные слова set и of. Задание конкретного множества определяется правилом (конструктором) - списком элементов или интервалов, заключенным в квадратные скобки. Пустое множество обозначается двумя символами [].

Множественный тип можно определить в разделе описания типов по схеме:

type <имя> = set of <тип элементов>

Например:

type t=set of byte;

var a:t;

Можно совместить описание множественного типа и соответствующих переменных:

var code: set of0..7;

digits: set of'0'..'9';

Можно описать переменную множественного типа и задать ее первоначальное значение в разделе описания констант, как константмножество. Тип множества можно описать ранее, например,

type up=setof'A'..'Z';

low=set of'a'.. "z";

const upcase : up=['A'. . 'Z'];

vocals :low=['a', 'e', 'i', 'o', "u", 'y'];

delimiterset of char=[ '..' /',':'..' ? '];

Для данных множественного типа определены операции объединения, пересечения и дополнения множеств, обозначаемые в Паскале соответственно знаками +, \* и -, а также отношения равенства множеств (А=В), неравенства (А<>В), включения (А<=В,А>=В).

Логическая операция принадлежностих in А принимает значение true, если элемент х принадлежит множеству А и false в противном случае. Так как к элементам множества прямого доступа нет, то операция in часто используется для этой цели.

Заметим, что операции отношения на множествах выполняются быстрее, чем соответствующие операции на числах, поэтому их выгодно применять в программах.

*Пример:* составить программу, анализирующую латинский текст и печатающую в алфавитном порядке все найденные в нем буквы, а затем все ненайденные.

Пусть alfa - множество всех букв латинского алфавита. Будем вводить заданный текст с клавиатуры символ за символом, одновременно формируя множество Е -множество латинских букв текста. В конце текста введем символ \*. Затем с помощью операции in будем проверять, какие буквы алфавита имеются во множестве Е. Множество N - ненайденных букв в тексте - определяется оператором: N := alfa - Е .

*Программа 14*

program search;

const alfa:set of char=['a' **. .**'z'];

var c:char;E,N:set of char;

begin

cirscr; E:=[]; writeln('введите текст, конец ввода -\*'); read(c);

while c<> '\*' do

begin

if с in alfa then E:=E+[c]; read(c)

end;

writeln;

if E=alfa then writeln('найдены все латинские буквы')

else begin

N:=alfa-E;

writeln('найдены:');

for c:='a' to 'z' do if с in E then write(c);

writeln; writeln('не найдены:');

for c:='a' to 'z' do if с in N then write(c);

writeln

end

end.

Переменные множественного типа удобно применять в задачах, где порядок данных не имеет значения, например при моделировании случайных событий.

*Пример:* составить программу «спортлото 5 из 36», которая позволяет человеку ввести с клавиатуры пять натуральных чисел из интервала 1..36, затем генерирует случайным образом пять различных чисел из того же интервала и объявляет величину выигрыша по правилу: если угаданы человеком 0, 1 или 2 числа, объявляется проигрыш; если угаданы 3 числа, объявляется выигрыш 3 рубля; если угаданы 4 числа, объявляется выигрыш 100 рублей; если угаданы 5 чисел, объявляется выигрыш 1000 рублей.

В программе используются обозначения: mn — множество натуральных чисел из интервала 1 . . 36, а - множество чисел, задуманных человеком, х - множество чисел, . генерируемых компьютером, z=a\*x - пересечение множеств а и х; i, k, s - переменные, значения которых принадлежат интервалу 1..36. Случайное число из этого интервала генерируется оператором: s:=random(35)+l. Программа сначала выводит на экран сообщение о выигрышных номерах, затем определяет величину выигрыша.

*Программа 15*

program lottery;

type mn = set of 1 . . 36;

var x**,a,** z: nm; i, k, s: 0 . . 36;

begin

writeln; a:=[];

for i:=l to 5 do

begin write('введите ',i,' -тое число '); readin(k);a:=a+[k] end;

randomize;

k:=0; x:=[];

whilek<5 do

begin

s:random (35) +1;

if not(s in x) then begin k:=k+l; x:=x+[s]

end

end; writeln;

writeln('выигрыш выпал на следующие номера : ');

for i:=l to 36 do if (i in x) then write(i,' ');

writeln; z:=a\*x; k:=0;

for i:=l to 36 do if (i in z) then begin writeln('угадано: ',i); k:=k+l

end;

case k of

0, 1, 2 : writeln('вы проиграли ');

3 : writeln('получите 3 руб') ;

4 : writeln('получите 100 руб');

5 : writein('получите 1000 руб')

end

end.

**Записи** (комбинированный тип) - одна из наиболее гибких и удобных структур данных, применяющихся при описании сложных объектов, которые характеризуются различными свойствами, а также при создании различных информационных систем. Запись - это последовательность, состоящая из фиксированного числа величин разных типов, называемых полями или компонентами записи. Так же, как и массив, запись содержит ряд отдельных компонент, но компонентами записи могут быть данные различных типов. Например, адресные данные (индекс, город, улица, номер дома, квартиры) можно представить как запись (record):

type address = record

index : string[6];

city: string[20];

street : string[20];

haus.ilat: integer

end;

Из примера видно, что тип «запись» описывается по схеме

type имя типа записи = record

имя поля 1 : тип;

имя поля 2 : тип;

имя поля N: тип

end;

Как и при описании массивов, можно совместить описание типа записи и соответствующих переменных. Например, данные о двух студентах можно описать так:

fio :string[20];

fas :string[10]

grup :string[8]

end

Переменную типа «запись» и ее первоначальное значение можно определить как констант-запись в разделе констант по схеме

const **<имя>:** <имя типа> = <константное значение>

Константное значение - это список имен полей и соответствующих значений, заключенный в круглые скобки. Элементы списка разделяются знаком «точка с запятой». Например, запись о начале координат можно определить так:

type point = record

х, у, z: integer

end;

const o: point = (x:0; y:0; z:0);

С компонентами записи можно обращаться как с переменными соответствующего типа. Обращение к компонентам записи осуществляется с помощью указания имени поля через точку. Пусть, например, переменная х имеет тип address, т.е. в программе имеется описание var x: address. Тогда допустимы следующие присваивания:

x.haus := 52; х.street:='пр.Мира'; x.city:= 'Красноярск';

x.flat:= 135; x.index :='б60049'

Проиллюстрируем работу с записями на задаче, в которой требуется найти сумму и произведение двух комплексных чисел:

zl=al+i\*blиz2=a2+i\*b2.

*Программа 16*

program cornpl;

type compi =record

re : real;

im : real

end;

var zl,z2,s,p : compl;

begin

writeln('компл.число a+i\*b вводите двумя числами а и Ь: 'it-write('введи 1 число: '); readln(zl.re,zl.im);

write('введи 2 число: '); readin(z2.re,z2.im);

s.re := zl.re + z2.re;

s.im := zl.re + z2.im;

p.re *:=* zl.re \* z2.re - zl.im \* z2.im;

p.im := zl.re \* z2.im + z2.re \* zl.im;

writeln('s=',s.re:4:2,' + i \*'*,s.*im:4:2);

write('p=',p.re:4:2,' + i \*',p.im:4:2)

end.

Громоздкость обозначений в программе компенсируется большей наглядностью алгоритма за счет структуризации данных. Во многих случаях, если требуется производить операции с полями фиксированной записи, можно для сокращения обозначений использовать оператор присоединения with. Его структура такова:

with <имя записи> do <оператор>;

В этом случае в операторе, написанном после служебного слова do, имена полей указанной записи описываются без имени записи и точки. Например, печать суммы s в предыдущем примере можно организовать с использованием оператора with так:

with s do writeln('s=',re:4:2,'+i\*',im:4:2);

В операторах присваивания разрешается использовать не только имена полей, но и имена записей.

Тип поля может быть записью.

Например:

man = record

fio:record

fam, im,otch **:** string[10];

end;

data : record

day: 1..31;

mes:1..12;

god:integer

end;

pol: char;

telef: record

dom,rab : string[10];

end;

end;

В Паскале разрешается использовать тип «запись» при описании другихсоставных типов данных, например, можно построить массив записей.

Рассмотрим пример эффективного использования записей в программе начисления стипендии студентам по шаблону:

N ФИО Эк1 Эк2 ЭкЗ Балл Сумма Проф Итого

1 Васнецов Н.В. 4 4 3 11 50.00 0.25 49.75

Предположим, что вводится список группы с соответствующими оценками за экзамены. Графа «Балл» вычисляет суммарную оценку за семестр. Графа «Сумма» определяет размер стипендии по упрощенному правилу: если нет двоек и балл равен 15, то стипендия 75 руб.; при условии, что 12 < «Балл» < 15 стипендия 62 руб 50 коп., а если 9 < «Балл» < 12 , то - 50 руб. (в других случаях сумма равна нулю). В графе «Проф» указывается профсоюзный взнос в размере 0,5% от стипендии, а графа «Итого» определяет сумму денег к выдаче.

В программе перед распечаткой итоговой ведомостиможно предусмотреть упорядочение записей по убыванию в графе «Балл».

*Программа 17*

program spisok;

type stud = record

fio :string[20];

ex1, ex2, ex3 : 2 . . 5;

bal :6 . . 15;

sum :real;

nalog :real;

itog :real;

end;

var x : array[1..30] of stud;

i,k,m,n :integer;

у : 6..15;

z : stud;

begin write('введи число студентов: '); readln(n);

for i:= 1 to n do with x[i] do

begin write('введи ФИО ',i,'-ro студента: ');

readln(fio); write('Bведи его три оценки: ');

readin(exl,ex2,ex3);

end;

for i:= 1 to n do with x[i] do

begin bal:=exl+ex2+ex3;

if (exl=2) or (ex2=2) or (ex3=2)

then sum:=0

else if bal=15 then sum:=75

else if bal>12 then sum:=62.5

else if bal>9 then sum:=50

else sum:=0;

nalog:=sum\*0.005; itog:=sum-nalog;

end;

for k:= 1 to n-1 do

begin y:=x[k].bal; m:=k;

for i:=k+l to n do if y<x[i].bal then

begin y:=x[i].bal; m:=i

end; z:=x[k]; x(k]:=x[m]; x(m]:=z;

end;

writeln; writeln ('СТИПЕНДИАЛЬНАЯ ВЕДОМОСТЬ ');

for i:=l to 64 do write('-'); writeln;

write ('N | ФИО | эк1 | эк2 | эк3 | балл | сумма | проф | итого |') ;

for i:=l to 64 do write('-'); writeln;

for i:=l to n do with x[i] do

begin write(i:3,fio:20,exi:4, ex2:4,ex3:4);

writeln(bal:5,sum:9:2,nalog:8:2,- itog:7:2);

end

end.

***Контрольные вопросы и задания***

1. Как определяется перечислимый тип данных?

2. Для чего может понадобиться интервальный тип данных?

3. Как вводятся и используются в программах массивы?

4. Какие действия возможны над величинами строкового типа?

5. Какие операции допустимы над множествами?

6. В чем принципиальные различия между одномерными массивами и записями?

7. В усеченном конусе длина диагонали осевого сечения равна d, образующая составляет с плоскостью основания угол х и равна а. Вычислите площадь боковой поверхности конуса.

8. Вычислите объем призмы, боковые грани которой квадраты, а основанием служит равносторонний треугольник, вписанный в круг радиуса г.

9. Числа а и b выражают длины катетов одного прямоугольного треугольника, числа с и d - другого. Определите, являются ли треугольники подобными.

10. Напечатайте числа а, b, с в порядке возрастания.

11. Определите все пары двузначных чисел, обладающих свойством: (20+25^2 = = 2025.

12. Вычислите в числовом массиве а1,а2,...,аn суммы положительных и отрицательных элементов.

13. Вычислите скалярное произведение двух десятимерных векторов Х и Y.

14. Упорядочите массив х1,х2,...,хn по неубыванию, используя метод сортировки вставками: пусть первые k элементов уже упорядочены по неубыванию; берется (K+1)-й элемент и размещается среди первых k элементов так, чтобы упорядоченными оказались уже (k+1) первых элементов.

15. Составьте программу решения треугольной системы уравнений порядка n.

16. Замените в заданном арифметическом выражении все вхождения sin на cos и sqrt на abs.

17. Для заданного текста определите длину содержащейся в нем максимальной последовательности цифр 0, 1,2,..., 9.

18. Дан текст из латинских букв и знаков препинания. Составьте программу частотного анализа букв этого текста, т.е. напечатайте каждую букву с указанием количества ее вхождений и процента вхождений.

19. Найдите и напечатайте в порядке убывания все простые числа из диапазона [2..201].

20. Опишите тип «запись» для следующих данных:

а) адрес(город, улица, дом, квартира);

б) дата(число,месяц.год);

в) студент(фио,факультет,курс,группа).

21. Заданы N точек на плоскости. Найдите точку, ближайшую к началу координат. Используйте тип «запись».

### 3.4. ПРОЦЕДУРЫ И ФУНКЦИИ

**Описание и вызов.** В Паскале подпрограммы называются процедурами и функциями и описываются в разделе с тем же названием.

**Процедура** имеет такую же структуру, как и программа, но с двумя отличиями:

• заголовок процедуры имеет другой синтаксис и включает служебное слово procedure;

• описание процедуры заканчивается точкой с запятой (а не точкой). Все имена, описанные в программе до процедуры, действуют во всей программе и в любой ее подпрограмме (если они там не описаны заново). Они называются глобальными, в отличие от локальных имен, описанных в процедуре и действующих лишь в ней.

Данные для обработки могут передаваться процедуре через глобальные имена или через аргументы процедуры. В процедуре каждый аргумент имеет свое имя -формальный параметр, описываемый в заголовке процедуры по схеме

procedure <имя> (<список описаний формальных параметров>) Описание формальных параметров может иметь вид

<список имен>: <тип> или var <список имен>: <тип>

В первом случае говорят о параметрах-значениях, во втором - о параметрах-переменных. В простейшем случае заголовок процедуры может содержать только имя процедуры.

Оператор вызова процедуры имеет вид

<имя процедуры> (<список выражений>);

Указанные выражения называют фактическими параметрами. Их список должен точно соответствовать списку описаний формальных параметров процедуры. Во время вызова процедуры каждому параметру-значению присваивается значение соответствующего фактического параметра и поэтому их обычно используют для передачи входных данных.

Параметры-переменные следует использовать для представления результатов процедуры.

*Пример:* составим программу, которая с помощью строки символов разделит экран на части, где напечатает таблицу квадратных корней для чисел 1, 2,..., 10 и таблицу натуральных логарифмов для чисел 1, 2,..., 5.

Печатьстроки символов оформим как процедуру. Так как никакую информацию передавать из процедуры в программу не надо, то аргументы процедуры (вид и количество символов) будут описаны как параметры-значения.

Заметим, что процедура в программе выполняется пять раз.

*Программа 18*

program section;

var x:integer;

procedure line(a:integer;c:char) ;

var j:integer;

begin

for j:=l to a do write (c);

writeln

end;

begin

line(35,'-'); writeln('таблица квадратных корней');

line(35,'-');

for x:=l to 10 do writeln(x:8,sqrt(x):8,4);

line (35,'-'); writein('таблица натуральных логарифмов');

line(35,'-');

for x:=l to 5 do writein(x:8,In(x):8:4);

line(35,'\*')

end.

Функция - это подпрограмма, определяющая единственное скалярное, вещественное или строковое значение. Отличия подпрограммы-функции от процедуры:

• заголовок функции начинается со служебного слова function и заканчивается указанием типа значения функции:

function <имя> (список описаний формальных параметров): <тип>;

•раздел операторов функции должен содержать хотя бы один оператор присваивания имени функции;

• обращение к функции - не оператор, а выражение вида

<имя функции> (<список фактических параметров>).

Функции (и процедуры) могут использовать свое имя в собственном описании, т.е. могут быть рекурсивными.

*Пример:* составим программу, которая для заданных четырех натуральных чисел а, b, с, d напечатает наибольшие общие делители первой и второй пар чисел и сравнит их по величине.

В программе определим **рекурсивную функцию** nod(x,y) по формулам

**|** x, если у = 0

nod(x,y) = | nod(y.x). если х < у

| nod(x mod у,у), если х > у

Применяя эти формулы к числам 21 и 15, последовательно находим nod(21,15) = nod(6,15) = nod(15,6) = nod(3,6) = nod(6,3) = nod(0,3) = nod(3,0) = 3.

*Программа 19*

program four;

var a,b,c,d,m,n:integer;

function nod(x,у:integer):integer;

var h:integer;

begin

if y=0 then h:=x

else if x<y then h:=nod(y,x)

else h:=nod(x mod у, у);

nod: =h end;

begin

writeln('введите 4 натуральных числа');

read(а,Ь,с,d); writeln;

m:=nod(a,b); n:=nod(c,d);

writeln('нод(',а,',',b,*'*)=',m);

writeln('нод(',c,',',d,')=',n);

if m>n then writeln('первый > второго')

else if m<n then writeln ('первый < второго')

else writeln('нод пар равны') end.

**Внешние библиотеки.** Как известно, подпрограммы (процедуры и функции) используются в программах с целью их структурирования, а также при многократных повторениях некоторых частей программы. Процедуры и функции описываются в программных единицах в разделе описания подпрограмм. Они являются внутренними для этих программных единиц.

Бывают случаи, когда одни и те же подпрограммы могут использоваться в различных программах одного и даже нескольких пользователей. В подобных ситуациях целесообразно создавать внешние подпрограммы, которые можно в необходимый момент подключать в любые программы. Как правило, внешние подпрограммы объединяют в отдельные пакеты, так называемые, библиотеки внешних подпрограмм. Могут создаваться личные библиотеки, специализированные библиотеки коллективного пользования и др. С одной из таких библиотек - встроенной библиотекой стандартных подпрограмм - пользователи имеют дело практически всегда. В состав этой библиотеки входят процедуры и функции вычисления значений ряда элементарных функций: синуса, косинуса, экспоненты и т.д., процедуры и функции обработки символьных величин, процедуры ввода-вывода и др. (список их приведен в конце § 3). Встроенная библиотека подключается к любой программе автоматически при компиляции. Поэтому откомпилированный файл с расширением .corn (иногда называемый «комовским»), как правило, занимает в 8 -10 раз больше места в памяти, чем исходный текст.

Внешние подпрограммы создаются обычным образом в виде отдельного файла или файлов. Для подключения внешних подпрограмм в программе пользователя в разделе описания ставится директива $I имя файла. С этого момента все процедуры и функции внешнего файла становятся внутренними для программы, и на все входящие в него процедуры и функции распространяется правило локальных и глобальных переменных. В этой связи, директива подключения внешнего файла должна размещаться после описания всех ею используемых глобальных параметров, процедур и функций.

*Пример.* Создадим внешнюю библиотеку из двух процедур и одной функции. Первая процедура программы 20 очищает экран, выдает приветствие, затем после нажатия клавиши <Пробел> снова очищает экран. Вторая процедура возводит число а в степень b. Третья подпрограмма-функция вычисляет значение экспоненты с некоторым грубым приближением на основе ряда Тейлора.

*Программа 20*

procedure PRIVET;

var a: char;

begin

cirscr; gotoxy(20,10) ;

write('здравствуйте , желаю успехов !') ;

repeat (цикл позволяет)

gotoxy(35,50);write('пробел'); (сменить экран}

read(kbd,а); (по нажатию клавиши)

until а=' **'; (\*** 'пробел' )

cirscr;

end;

procedure STEPEN(a,b:real;var y:real);

begin

y:=exp(b\*ln(a)) ;

end;

function MEXP(x:real):real;

begin

mexp:=l+x+x\*x/2+x\*x\*x/6+x\*x\*x\*x/24;

end;

Пусть представленные три подпрограммы записаны в файл с именем lab.pas. А теперь составим программу, использующую созданную внешнюю библиотеку.

*Программа 21*

program primeri;

($i lab) (директива подключения библиотеки} var a,b : real;

begin

PRIVET;

STEPEN(2,4,a); writeln('2 в степени 4 =',a); b:=MEXP(l);

write('машинная exp(1)=',EXP(1):6:4,' моя exp(1)=',b:6:4);

end.

В программе используется стандартная функция - экспонента ЕХР(1) и наша подпрограмма МЕХР(1).

**Модули** используют в более поздних версиях Паскаля для создания библиотек и разделения больших программ на логически связанные независимые друг от друга составные части. В состав модуля входят следующие разделы: заголовок, интерфейс, реализация, инициализация. Заголовок необходим для ссылок на модуль. Интерфейс содержит объявления, включая процедуры и функции, представленные списком заголовков и доступные пользователям в теле основной программы. Раздел «реализация» содержит тела процедур и функций, перечисленных в интерфейсной части модуля. Раздел «инициализация» содержит операторы, необходимые для инициализации модуля. Таким образом модуль - это набор констант, типов данных, переменных, процедур и функций. Каждый модуль компилируется отдельно; результат компилляции - файл с расширением .tpu (Turbo Pascal Unit). Каждый элемент модуля можно использовать в программе пользователя без дополнительного объявления, для чего достаточно записать имя используемого модуля в директиве Uses в начале программы после его заголовка.

В Турбо-Паскале версии 5.0 и выше применяют стандартные модули CRT, GRAPH и др. В этих модулях содержатся сервисные процедуры и функции по работе с экраном дисплея, с клавиатурой, графическими примитивами и т.п. Модули подключаются к программе путем специальной команды, размещаемой сразу после заголовка:

uses <имя модуля>

Программист может сам создать модуль. Ниже приведен пример с соответствующими комментариями.

*Пример.* Создать модуль, дополняющий математические возможности Паскаля арифметическими действиями над комплексными числами.

Будем представлять комплексные числа парами действительных: (а, b). Как известно, действия над ними выполняются по правилам

(a,b) + (c.d) = (a+c,b+d),

(a,b)-(c,d)=(a-^,b-d),

(a,b) \* (c,d) = (a\*c-b\*d , a\*d+b\*c),

(a,b) / (c,d) = ( (a\*c+b\*d)/(c\*c+d\*d), (b\*c-a\*d)/(c\*c+d\*d)).

Создаваемый модуль будет включать четыре процедуры: Sum - сумма,Raz -разность, Proiz - произведение, Chastn - частное.

Этот модуль может быть отдельно откомпилирован. После этого любая программа, написанная на Паскале, может получить доступ к интерфейсным объектам (в данном случае - процедурам) этого модуля с помощью директивы Uses CompChisla.

Обратим внимание, что в интерфейсной части модуля от процедур присутствуют лишь заголовки, а в части «реализация» от заголовков процедур остаются лишь их имена.

*Программа 22*

unit CompChisla;

interface

procedure Sum(a,b,c,d: real; var **x,y:** real);

procedure Raz(a,b,c,d: real; var **x,y:** real);

procedure Proiz (a,b,c,d: real; var **x,y:** real);

procedure Chstn(a,b,c,d: real; var **x,y:** real);

implementation

procedure Sum;

begin **x:=a+c;** y:=b+d end;

procedure Raz;

begin **x:=a-c; y:=b-d end;**

procedure Proiz;

begin x:=a\*c-b\*d; y:=a\*d+b\*c

end;

**procedure** Razn;

var z:real;

begin z:= c\*c+d\*d; x:=(a\*c+b\*d)/z; y:=(b\*c-a\*d)/z end;

end.

***Контрольные вопросы***

1. Какова структура процедуры? функции?

2. Какие параметры называют формальными и какие - фактическими?

3. В чем различие между локальными и глобальными переменными?

4. В чем сходство и различие между процедурой и модулем?

### 3.5. РАБОТА С ФАЙЛАМИ

Файл (последовательность) - это одна из наиболее фундаментальных структур данных. Программная организация компьютеров,их связь с внешними устройствами основаны на файловой структуре.

Файлы позволяют решить две проблемы:

1) возможность формирования и сохранения значений для последующего использования другими программами (например, в программах многократной обработки информационных систем, таких как платежные ведомости, различные АСУ, базы данных, необходимость длительного хранения информации очевидна);

2) взаимодействие программ с внешними устройствами ввода-вывода: дисплеем, принтером, АСП и т.п.

В Паскале эти проблемы снимаются с помощью структурированных данных файлового типа.

Файловый тип данных в программе задается следующем образом:

type <имя файлового типа> = file of <тип компонентов>

В качестве типа компонентов файла разрешается использовать любой тип данных, кроме файлового. Например:

type

intfile = file of integer;

refile = file of real;

chfile = file of char;

ran = 1 .. 10;

st = set of ran;

vector = array[ran] of real;

comp = record

re,im : integer;

end;

setfile=file of st;

vecfile = file of vector;

compfile= file ofcompt;

Описание файловой переменной задается обычным способом в разделе описаний. Например:

var f: intfile; или var f: file of integer;

Файловая переменная является буфером между Паскаль-программой и внешним устройством и должна быть логически с ним связана. Связь осуществляется оператором языка Паскаль:

assign (<имя файловой переменной>,'<имя устройства>')

Как правило, файлы для хранения данных связаны с устройством внешней памяти на магнитных носителях (дисковод) и носят название **внешние файлы**. Если, например, файл с именем primer, dat логически связан с дисководом А:, то все данные, помещаемые в файл, будут храниться на этом дисковом накопителе, aустановка «окна» между программой и файлом будет определяться через файловую переменную f оператором

assign (f, 'primer.dat')

Если внешним устройством является принтер, то связь осуществляется оператором assign(f, '1st:'). Здесь 1st - логическое имя печатающего устройства. Ниже приведены логические имена внешних устройств ввода-вывода:

con - консоль; trm - терминал; kbd - клавиатура; 1st - принтер; aux - буфер сети;

usr-драйвер пользователя.

После осуществления связи файловая переменная f отождествляется с соответствующим файлом.

Для работы с файлом его необходимо открыть, а по окончании работы — закрыть Файл открывается для чтения оператором reset(f), для записи - оператором rewrite(f).

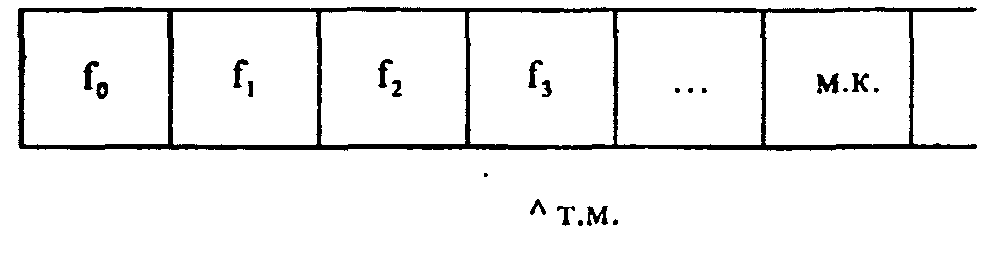
Чтение и запись данных осуществляется известными командами read/write, только в начале списка помещается имя файловой переменной:

read (f, <список ввода>); readln (f, <список ввода>);

write(f, <список вывода>); writeln(f, <список вывода>).

Закрытие файла осуществляется командой close(f).

условно файл можно представить в виде ленты, у которой есть начало, а конец не фиксируется. Компоненты файла записываются на эту ленту последовательно, друг за другом:

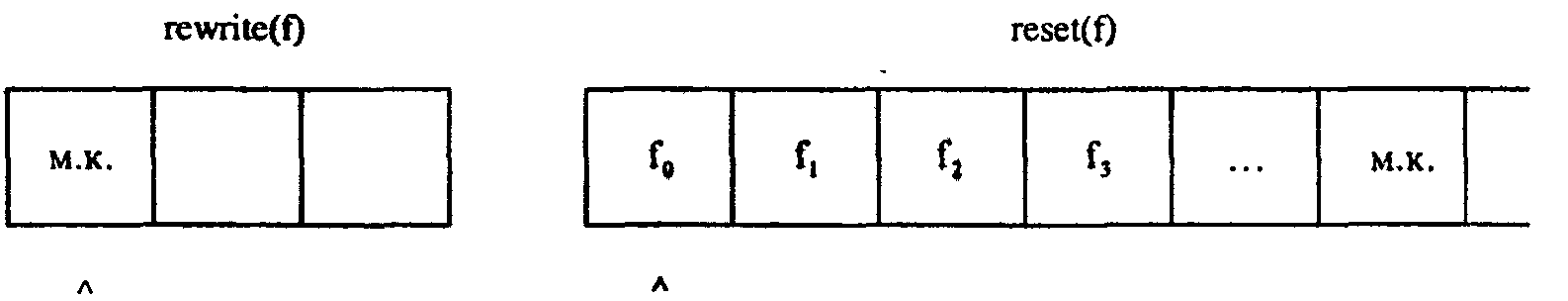


Здесь т.м. - текущий маркер, указывающий на рабочую позицию (окно) файла; м.к. (маркер конца файла) - специальный код, автоматически формируемый вслед за последним элементом файла.

Такого рода файлы называются **файлами последовательного доступа**. В исходной версии Паскаля **файлов прямого доступа**, для которых можно непосредственно «достать» любую компоненту, не предусмотрено; однако, в Турбо-Паскале элементы прямого доступа есть (например, через функцию seek, см. ниже).

Команда rewrite(f) - открыть файл для записи - устанавливает файл в начальное состояние режима записи; текущий маркер устанавливается на маркер конца файла. Если в файле f до этого была информация, то она уничтожается.

В открытом для чтения командой reset(f) файле текущий маркер устанавливается на нулевое состояние, однако содержимое файла не утрачивается.



Команда закрытия файла close (Q обязательна, поскольку эта команда формирует маркер конца файла, что в большинстве случаев является необходимым нaпpимep, для работы с функцией eof(f), см ниже).

В системе Турбо-Паскаль предусмотрены встроенные функции по работе с файлами:

filesize(f) - текущее количество компонент открытого файла;

filepos(f) -номер текущей позиции маркера;

геnamе (f,имя) - переименование файла, связанного с f;

erase(f) -уничтожение файла;

execute(f) - выполнение СОМ-файла;

chain(f) -выполнение CHN-файла;

seek(f,N) - устанавливает маркер на позицию N;

eof(f) - возвращает TRUE, если найден конец файла;

ealn(f) - возвращает TRUE, если найден конец строки.

На практике широко используются **текстовые файлы**, которые состоят из литерных (логических) строк. Поэтому в языке Паскаль предусмотрен стандартный файловый тип TEXT (он не является file of char, скорее всего, это - file of string[n]). Логические строки бывают разной длины, в том числе и нулевой. В конец каждой строки помещается специальный символ «конец строки» (eoln - «end of line»). В качестве печатного символа конца строки используют литеру #. Текстовый файл (text) является строго последовательным, к нему не применимы некоторые встроенные функции, в частности, seek. В отличие от типизированных файлов, с текстовым файлом нельзя одновременно проводить операции чтения (read) и записи (write). Однако, допустимы операторы writein и readln. Числовые данные, целые и вещественные, в текстовом файле должны записываться через пробел.

Ниже приведены несколько примеров, иллюстрирующих работу с файлами. *Пример 1.* Вывод данных на печатающее устройство - принтер (1st:).

*Программа 23*

program print;

var

fal :text; **x :real;** *name :*string[25];

begin

assign (fal, ' 1st:'); rewrite (fal); x:2.5; name:"'Слава';

writeln(fal,x:8:2) ;

writeln(fal,'Привет, '.name); close(fal) :

end. (Здесь файловая переменная fal связывается с принтером 1st:, и запись в файл fal практически означает вывод на печать)

*Пример 2.* Создание и сохранение в файле «xxx.dat» последовательности целых чисел от 10 до 20.

*Программа 24*

program zapis;

var

f: file of integer; i: integer;

begin

assign(f,'xxx.dat'); rewrite(f);

for i:=10 to 20 do write(f,i); close (f);

end. (После выполнения программы формируется внешний файл xxx.dat)

*Пример 3.* Считывание первых пяти компонент из существующего файла «xxx.dat» и вывод на дисплей квадратовэтих значений.

*Программа 25*

program read;

var ff: file of integer; j,i : integer;

begin

assign(ff,'xxx.dat'); reset(ff) ;

for j:=l to 5 do begin read(ff.i); writeln(i\*i);

end;

close(ff);

end.

*Пример 4.* В текстовом файле (text) «slov.txt» содержится русский текст. Определить сколько гласных букв в тексте.

*Программа 26*

program text;

var

ft : text; n : integer; ch : char; st : set of chart begin

assign (ft,'slov.txt'); reset(ft);

st *:=* ['A'.'E','И','О','У,'Ы','Э','Ю','Я'];

st := st+['a','e','и','о','у','ы','э','ю','я'];

n: =0;

while not eof(ft) do begin

read(ft, ch); if ch in st then n:=n+l;

end;

close(ft);

writeln; write('кол-во гласных букв =',n);

end.

Поскольку длина текста (файла) неизвестна, то в цикле «пока» используется логическая функция eof(f), которая возвращает значение TRUE, если найден конец файла.

*Пример 5.* Шифрование и дешифрование текста.

Программа шифрования заданного текста (sekret) использует следующее правило шифровки: каждая буква в тексте заменяется на букву, расположенную на n позиций вправо от искомой в русском алфавите. Причем последним символом алфавита является пробел '', а далее алфавит продолжается циклически.

Значение смещения n находится во внешнем файле 'n.key', который формируется программой key. Зашифрованный текст выводится во внешний файл с именем «шифр.eхe» , а также на дисплей.

Программа дешифровки (retsek) считывает шифрованный текстиз файла «шифр.tхt» и выводит на экран дисплея искомый текст.

*Программа 27 (а)*

program key;

var

n : integer; f : file of integer;

begin , .

assign (f,'n.key'); rewrite(f); write\('Bвeди ключ(смещение): ');

readln(n);

write(f,n); close(f);

end.

*Программа 27 (б)*

program sekret;

var

| slovo,anslovo: string[100];

alfavit : string[33];

n, i, k, p : integer;

fkl : file of integer;

fs : text;

begin

alfavit:='абвгдежзийклмнопрстуфхцчшщъыьэюя ';

assign(fkl,'n.key'); reset(fkl); read(fkl,n); close(fkl);

writeln; write('введи текст: ');

readln(slovo); anslovo:='';

for k:=l to length(slovo) do begin for i:=l to 33 do

if slovo[k]=alfavit[i] then begin p:=i+n;

if p >33 then p:=p mod 33;

anslovo:=anslovo+alfavit[p];

end;

end;

assign(fs,'шифр.txt'); rewrite(fs); write(fs,anslovo);close(fs);

writeln; write(ansiovo) ;

end.

*Программа 27 (в)*

program retsek;

var slovo, anslovo : string[100];

alfavit : string[33];

n, i, k, p : integer;

fi : file of integer;

f : text;

begin alfavit:='абвгдежзийклмнопрстуфхцчшщъыьэюя ';

assign(fi,'n.key'); reset(fi); read(fi.n); close(fi);

assign(f,'шифр.txt'); reset(f); read(f,anslovo); close(f)

slovo:='' ;

for k:=l to length(anslovo) do

begin for i:=l to 33 do

if anslovo[k]=alfavit[i] then

begin p:=i-n; if p < 1 then p:=33-p mod 33;

slovo:=slovo+alfavit[p];

end;

end;

writeln; write('TeKCT шифровки: ',slovo);

end.

***Контрольные вопросы и задания***

1. Какие проблемы решаются при использовании файлов?

2. Какие операции и функции используются при работе с файлами?

3. Найдите площадь выпуклого четырехугольника со сторонами х, у, z, t и одной

из диагоналей d.

4. Заданы три комплексных числа. Найдите их сумму,

5. Имеется внешний файл записей «lab.zap», содержащий сведения об ученика.' школы. Файл формируется приведенной ниже программой.

Составьте программу, в которой необходимо сделать следующее:

а) упорядочить файл по признаку «class» в порядке возрастания;

б) упорядочить файл по признаку «god» в порядке убывания;

в) упорядочить файл в алфавитном порядке.

program lab;

type shcoo 1= record

fiо : string[20], class : 1 .. 11; god : integer; pol: char;

end;

var x:array[1..100] ofshcool; n,i:integer; f:file of school;

begin

write ('введите кол-во учеников:'); readln(n); assign(f,'lab.zap'); rewnte(f);

for i:=l to n do with x[i] do

begin \write('введи Ф.И.О.',1,'-ого ученика:'); readln(fio);

write('клacc:'); readln(class);

write('гoд рождения:-'); readln(god); write('пол(M/Ж):-');

readln(pol); write(f,x[i])…;

end; close(f);

end.

6. Задание рассчитано на двух студентов, использующих «электронную почту» на одном компьютере. Первый студент составляет программу, в которой формирует внешний файл «письмо». Второй студент должен «прочитать» файл и сформировать «ответ».

ПИСЬМО ОТВЕТ

а) последовательность целых чисел до 100 | квадраты этих чисел;

б) простые числа в интервале от 1 до 200 | сумма этих чисел

### 3.6. ДИНАМИЧЕСКИЕ ИНФОРМАЦИОННЫЕ СТРУКТУРЫ

**Динамические переменные и указатели.** До сих мы рассматривали **статические** переменные. Такие переменные автоматически порождаются при входе в тот блок, в котором они описываются, существуют на протяжении работы всего блока и уничтожаются при выходе их этого блока.

Обращение к статическим переменным производится по их именам, а тип определяется их описанием. Вся работа по размещению статических объектов в памяти машины выполняется на этапе трансляции. Однако использование только статических переменных может вызвать трудности при составлении эффективной машинной программы. Во многих случаях заранее неизвестен размер той или иной структуры данных, или структура может изменяться в процессе выполнения программы. Одна из подобных структур - последовательный файл - уже ранее рассматривалась.

В языке Паскаль предусмотрена возможность использования **динамических** величин. Для них выделение и очистка памяти происходит не на этапе трансляции, а в ходе выполнения самой программы. Для работы с динамическими величинами в Паскале предусмотрен специальный тип значений - *ссылочный.* Этот тип не относится ни к простым, ни к составным. Переменные ссылочного типа, или *указатели,* являются статическими переменными. Значением переменной ссылочного типа является адрес ячейки - места в памяти соответствующей динамической величины, Свое значение ссылочная переменная получает в процессе выполнения программы, в момент появления соответствующей динамической величины.

Переменные ссылочного типа (указатели) вводятся в употребление обычным путем с помощью их описания в разделе переменных, а их тип, указывающий на тип создаваемых в программе соответствующих динамических величин, тоже определяется либо путем задания типа в описании переменных, либо путем указания имени ранее описанного типа.

Значением указателя является адрес ячейки, начиная с которой будет размещена в памяти соответствующая динамическая величина. Задание ссылочного типа выполняется по схеме

type <имя ссылочного типа> = ^ <имя типа динамической величины>

(значок ^ указывает на то, что величина является динамической).

Например:

type

р = ^ integer;

q = ^ record

х:integer;

у: string [20]

end;

Объявлены два ссылочных типа - р и q. Первый - указатель на целочисленные значения, второй - на двухполевую запись.

Связь указателя с динамическим объектом можно изобразить следующим образом:

Указываемый объект

\*

Р

На этой схеме р - имя указателя; звездочкой изображено значение указателя, а стрелка отражает тот факт, что значением указателя является адрес объекта (ссылка на объект), посредством которого объект и доступен в программе.

В некоторых случаях возникает необходимость в качестве значения указателя принять «пустую» ссылку, которая не связывает с указателем никакого объекта. Такое значение в Паскале задается служебным словом nil и принадлежит любому ссылочному типу. Результаты выполнения оператора p:=nil можно изобразить

следующим образом:

\*

Р

Имея объявленные типы, можно обычным образом описывать переменные этих типов.

Например,

var i: р; zap: q;

Динамические переменные базовых типов можно вводить прямо в разделе «описания переменных»:

var i: ^integer;

Описание указателя еще не резервирует память для значения соответствующего динамического объекта. Так, каждое вышеприведенное описание выделяет в памяти два байта для записи адреса \* - значения указателя, но никакой величины типа р или q в памяти не образуется и даже адреса \* еще нет.

Для порождения динамического объекта, на который указывает ссылочная переменная i, сложит стандартная процедура new(i), где new - «новый» - имя процедуры, i - имя указателя, я

Процедура new(i) выполняет две функции:

1) резервирует место в памяти для размещения динамического объекта соответствующего типа с именем i;

2) указателю i присваивает адрес динамического объекта i.

Однако, узнать адрес динамической переменной с помощью процедуры writeln(i) нельзя.

Динамические объекты размещаются по типу стека в специальной области памяти — так называемой «куче», свободной от программ и статических переменных. Как уже было отмечено, символ ^ после имени указателя означает, что речь идет не о значении ссылочной переменной, а о значении того динамического объекта, на который указывает эта ссылочная переменная. Так, если в программе имеется описание переменной

var:^integer

то при выполнении процедуры new(i) порождается динамическая переменная типа integer. Если затем будет выполнен оператор присваивания

i^:= 58

то этой динамической переменной будет присвоено значение 58.

Имя ссылочной переменной с последующим символом ^ называют «переменной с указателем». Именно она синтаксически выполняет роль динамической переменной и может быть использована в любых конструкциях языка, где допустимо использование переменных того типа, что и тип динамической переменной.

Так, ecли j - статическая переменная типа integer, то допустимы операторы присваивания

j:=j+i^2; i^:i^div 3+4; i^sqr(i^) и т.д.

Если ссылочная переменная b указывает на массив

type mas = array[l... 100] of integer, ,

var b:^mas

то в этом случае переменные с указателем могут выглядеть, например, так:

b^[2], b^[k+5]*.*

Если в процессе выполнения программы некоторый динамический объект *р^,* созданный в результате выполнения оператора new(p), становится ненужным, то его можно уничтожить (очистить выделенное ему место в памяти) с помощью стандартной процедуры dispose(p). В результате выполнения оператора вида dispose(p) динамический объект, на который указывает ссылочная переменная р, прекращает свое существование, занимаемое им место в памяти становится свободным, а значение указателя р становится неопределенным (но не равным nil).

Если до вызова процедуры dispose(p) имел пустое значение nil, то это приведет к «зависанию» программы.

Если же до вызова этой процедуры указатель р не был определен,то это можетпривести к выходу из строя операционной системы.

Значение одного указателя можно присвоить другому указателю того же типа. Можно также указатели одинакового типа сравнивать друг с другом, используя отношения «=» или «<>».

Стандартные процедуры new и dispose позволяют динамически порождать программные объекты и уничтожатьих, что дает возможность использовать память машины более эффективно.

**Связанные списки данных.** Несмотря на богатый набор типов данных в Паскале, н не исчерпывает всего практически необходимого для разработки многих классов программ. В частности, из разнообразных *связанных структур данных* в языке стандартизированы массивы и файлы, а кроме них могут потребоваться и схожие с ними, но иные структуры. Для них характерны, в частности, следующие признаки:

а) неопределенное заранее число элементов;

б) необходимость хранения в оперативнойпамяти.

Средство для реализации таких структур дает аппарат динамических переменных.

Простейшей из обсуждаемых структур является однонаправленный список. Он строится подобно очереди на прием к врачу: пациенты сидят на любых свободных местах, но каждый из них знает, за кем он в очереди (т.е. данные размещаются на свободных местах в памяти, но каждый элемент содержит ссылку на предыдущий или следующий элемент). Поскольку количество пациентов заранее не очевидно, структура является динамической.

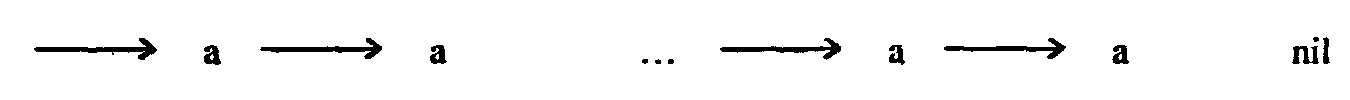
Другая подобная структура, с упоминанием о которой мы уже не раз встречались - **стек**. Его моделью может служить трубка с запаянным концом, в которую вкатывают шарики. При этом реализуется принцип «последним вошел - первым вышел». Возможное количество элементов в стеке не фиксировано.

Остановимся на примере стека и покажем его программную реализацию. Технически при этом следует решить ряд задач, из которых наиболее специфическими являются

а) связывание последующих компонентов стека;

б) смещение ссылок при каждом движении по стеку.

Из-за необходимости хранить не только значение каждого элемента, но и соответствующую ссылку на последующий элемент, каждый из элементов будем хранить в виде двухполевой записи, в которой первое поле - значение элемента, а второе -ссылка на следующий элемент. Схематически эту структуру можно описать следующим образом



(элементу, который пришел первым, ссылаться не на что, о чем свидетельствует «пустая ссылка» nil).

Пусть для конкретности элементы стека - действительные числа, и при последовательном заполнении состояния стека будут

1,75

35,7 1,75

-6,94 35,7 1,75

Приведенная ниже программа включает две процедуры: добавления очередной компоненты к стеку и изъятия ее. Программа формирует указанный стек и производит контрольное извлечение из него. Каждое действие снабжено комментарием, который поможет разобраться в программе. Следует обратить особое внимание на организацию смещения ссылок.

*Программа 28*

program stack; **(формирование динамической структуры - стека}**

type s^StackComp; StackComp=record

b:real;

p;s;

end;

var a:s; k:real;

procedure Dobavl(k:real); (процедура добавления к стеку числа «к»}

var j : s ;

begin new(j); (создание новой динамической записи}

j^.b:=k; (внесение следующей компоненты стека}

j^.p;=a; (смещение указателя} a:=j (формирование новой записи)

end;

procedure Vzjat(var k:real); (процедура чтения из стека)

var j :s;

begin j:=а;

k:j^b; (k-значение последней компоненты}

a:=j^p; (перенастройка указателя)

dispose(j) (ликвидация использованной записи)

end;

begin (пример записи и чтения)

a:=nil; Dobavl(1.75) ;

Dobavl(35.7);

Dobavl(-6.94); ( сформирован стек: -6.94, 35.7, 1.75)

Vzjat(k); writeln(k); (контрольное взятие из стека)

Vzjat(k); writeln(k); (еще одно взятие из стека)

Vzjat(k); writeln(k) (еще одно взятие из стека)

end. (результат работы программы: -6.94 35.7 1.75}

### *Контрольные вопросы*

1. В чем принципиальное различие статических и динамических переменных?
2. 2 Как реализуется связь указателя с динамическим объектом?
3. 3. Что такое «связанный список данных»?

### 3.7. РАБОТА С ГРАФИКОЙ

**Машинная (компьютерная) графика** - одно из важных направлений в современной прикладной информатике. В отличие от базового Паскаля, современные версии содержат мощные средства разработки графических программ. Рассмотрим часть соответствующих возможностей Турбо-Паскаля, в котором они реализованы с помощью стандартного модуля Graph.

Модуль представляет собой мощную библиотеку графических подпрограмм универсального назначения, рассчитанную на работу с наиболее распространенными графическими адаптерами CGA, EGA, VGA, SVGA IBM-совместимых персональных компьютеров.

Подключение модуля Graph, tpu к программе выполняется директивой

uses graph;

Инициализация графического экрана осуществляется с помощью процедуры Initgraph. Драйвер поддерживает тот или иной режим экрана, табл. 3.1.

Таблица 3.1

**Некоторые сведения о драйверах н определяемых ими режимах**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Адаптер | Драйвер | Режим (Номер, имя) | Разрешимость | Число страниц |
| EGA | EGA | OEgalo | 640х200 | 4 |
| 1 Egahi | 640х350 | 2 |
| VGA | VGA | OVgalo | 640х200 | 4 |
| IVgalo | 640х350 | 2 |
| 2Vgalo | 640х480 | 1 |

Процедура инициализации в Турбо-Паскале имеет три аргумента:

Initgraph(<драйвep>, <режим>, '<путь к драйверу>').

Она может быть выполнена так:

uses graph;

var gd, gm: integer; {переменные gd и gm определяют драйвер и режим}

begin

gd:=vga; gm:=vgahi;

initgraph(gd,gm.'d:\tp551);

Первые две команды можно заменить одной:

gd:=detect

Целая константа detect=0 в модуле Graph автоматически распознает драйвер и устанавливает режим максимального разрешения для данной машины.

Процедура closegraph освобождает память от драйвера и устанавливает режим работы экрана, который был до инициализации графики.

Для обнаружения ошибок в графике применяются функции graphresult и grapherrormsg (код ошибки). Последняя выдает строку сообщения о характере ошибки, соответствующей коду. Инициализация графического режима с проверкой ошибок может быть выполнена в программе следующим образом:

uses graph; var gd, gm, errorcod: integer;

begin

gd:=detect; initgraph(gd,gm,");

errorcod'=graphresult;

if errorcod <>grok then

**begin**

writeln('ошибка графики');

writeln(grapherrormsg(errorcod));

halt

end;

Процедура Halt останавливает выполнение программы и возвращает управление операционной системе.

Для формирования палитры используется система смешения красного, зеленого и синего цветов и изменения яркости луча. Цвет задается номером из списка цветов палитры в интервале 0 .. 15.

Процедуры sе1со1ог(<цвет>) и setbkcolor(<цвет>) устанавливают текущий цвет рисунка и цвет фона. При инициализации графики по умолчанию устанавливается черный фон и белый цвет рисунка.

В табл. 3.2 указаны основные процедуры для модуля Graph, применяющиеся для построения простейших геометрических примитивов.

Координаты точек воспринимаются в «экранной» системе координат, в которой начало - верхний левый угол экрана, ось «х» направлена вниз, ось «у» -направо. Максимальные значения координат определяются разрешимостью экрана (см. табл. 3.1).

Первый аргумент процедуры setlinestyle(a,b,t) а - стиль линии второй параметр b -«образец» - имеет значение 4, если а=4, в остальных случаях b=0; третий параметр t -толщина линии - может иметь значение 1 (нормальная толщина) или 3 (жирная линия).

Таблица 3.2

Основные процедуры модуля Graph

|  |  |
| --- | --- |
| Заголовок процедуры | Геометрический смысл |
| putpixel(x,y,c) | Построить точку (х,у) цветом с |
| setlinestyle(a,b,t) | Установить стиль, образей и толщину линий |
| line(xl,yl,x2,y2) | Соединить две точки отрезком |
| rectangle(xl,yl.x2.y2) | Построить прямоугольник с заданными концами диагонали и |
|  | сторонами, параллельными осям координат |
| circle(x,y,r) | Построить окружность с центром (х,у) и радиусом г |
| arc(x,y,a,b,r) | Построить дугу окружности: a,b - начальный и конечный угол в |
|  | Градусах |
| ellipse(x,y,a,b,rx,ry) | Построить эллиптическую дугу: rx, rу - полуоси эллипса |
| setfillstyle(t,c) | Установить стиль закраски и ее цвет |
| fillellipse(x,y,rx,ry) | Построить закрашенный эллипс, используя цвет рисунка |
| floodfill(x,y,cg) | Закрасить фигуру до границы с цветом cg; (x,y) - внутренняя |
|  | точка фигуры |
| bar(xl,yl,x2,y2) | Построить столбец, используя тип и цвет закраски |
| pieslice(x,y,a,b.r) | Построить и закрасить сектор круга |
| sector(x,y.a,b,rx,ry) | Построить и закрасить эллиптический сектор |
| settextsiyle(f,n,d) | Установить шрифт, направление вывода и размер символа текста ' |
| outtextxy(x,y,st) | Вывести строку st, начиная с точки (х,у) |
| outtext(st) | Вывести строку, начиная с точки расположения текущего указателя |

Первый аргумент процедуры setfillstyle(t,c) - тип закраски t - принимает значения из интервала 0..12. Наиболее употребителен тип t = 1 - заполнение фигуры текущим цветом.

Для вывода текста на графический экран сначала выполняется процедура settextstyle(f,n,d), устанавливающая шрифт f, направление вывода п и размер символов (параметр d). При f = 0 используется стандартный точечный шрифт, встроенный в систему Турбо-Паскаль. С использованием других шрифтов познакомимся ниже. Направление вывода п принимает значения 0 (горизонтальный вывод) и 1 (вертикальный вывод).

Размер букв определяется параметром d, принимающим значения из интервала 1..10. Если d = 1 и f = 0, то каждый символ занимает квадрат 8\*8 точек, при d > 1 сторона квадрата умножается на d.

Далее, с помощью процедуры outtextxy(x,y,st) строка st выводится на экран, начиная с точки (х,у).

Например:

settextstyle(0,0,2); оuttехtху(100,200,'горизонтальная строка');

оuttехtху(100,230,'размер увеличен вдвое');

**Примеры графических программ**

*Пример 1:* программа рисует звездноенебо с 400 «звездами», вспыхивающими постепенно, и полную желтую луну.

*Програлша 29*

program sky;

uses crt,graph;

var k,gd,gm:integer;

begin gd:=detect;

initgraph (gd, gm, ' '); randomize;

for k:=l to 400 do

begin putpixel(random(640),random(480),random(15)+1); delay(10);

end;

setfillstyle(l,14); setcolor(14);

circle(550,80,30) ; floodfill(550,80,14) ;

repeat until keypressed; closegraph

end.

*Пример 2****.*** *Узор.* Используя простейшие геометрические образы строят замечательные графические изображения. Ниже приведена программа изображения муарового узора, полученного пересечением двух семейств расходящихся отрезков прямых. Качество и изображение получаемого узора зависит в основном от трех параметров: kl, k2 - расстояний между отрезками слева и справа ; h - смещения вниз (вверх) всего семейства.

*Программа 30*

program uzor;

uses crt, graph;

var gd,gm,errCode,i,kl,k2,h; integer;

begin

kl:=8; k2:=3; h:=110; gd:=Detect; InitGraph(gd,gm.'') ;

errCode:=GraphResult ;

if errCode = grOk then begin

setcolor(green); \*

for i:=l to (420 div kl) do

begin line(0,i\*kl,640,i\*k2+h); line(0,i\*k2+h,640,i\*kl);

end;

repeat until keypressed; CloseGraph;

end else writeln('errCode=',errCode)

end.

Процедуры построения прямоугольных фигур удобно использовать, в частности, при построении схем, диаграмм.

*Пример 3:* программа 31 строит столбчатую диаграмму, наглядно отражающую числовую информацию о населении 6 крупных городов мира: Токио, Гамбурга, Москвы, Бангкока, Мехико и Парижа.

*Программа 31*

program colon;

uses crt,graph;

const m:array[l..6] of real=(11500,2300,9700.5100,12400,8200) ;

name:array[I..6] of

string =('Токио','Гамбург','Москва','Бангкок','Мехико','Париж');.

var gd,gm,k,n,s:integer; st:string[6];

begin

gd:=detect;

initgraph(gd,gm,' ');

setcolor(15);

setlinestyle(0,0,1);

line(60,400,620,400);

line(60,400,60,100) ;

settextstyle(0,0,1);

for k:=l to 12 do begin n:=1000\*k; str(n.st);

outtextxy(10,400-20\*k-4,st);

line(60,400-k\*20,65,400-k\*20)

end;

setcolor(14); settextstyle(0,0,2);

outtextxy(120,20, 'Население городов (тысяч)'); settextstyle(0,0,1);

for k:=l to 6 do begin setfillstyle(1,k+2);

bar(100+(k-l)\*70,400,100+k\*70,round(400-m[k]/1000\*20)) ;

outtextxy(100+(k-l)\*70+4,450,name[k]) ;

end;

repeat until keypressed;

closegraph

end.

Процедура bar3d(xl,yl,x2,y2,d,top) рисует трехмерный столбец, глубина которого определяется параметром d. Последний параметр процедуры top - логического типа. Если top=true, рисуется верхнее основание столбика, в противном случае оно не изображается, что позволяет рисовать столбики один над другим. Диаграммы с трехмерными столбиками красивы, но их труднее создавать.

**Построение графиков функций**. Построение графиков функции - неотъемлемая часть большинства программ, предназначенных для обучения математике, физике. Ниже представлен пример программы построения графика функции y=x\*x\*sin(l/x). на произвольном отрезке [а, b]. Количество точек графика (параметр п) также задается произвольно (точка х=0 исключается, так как в ней функция не определена). В программе также определяются величины tl=(xl-x0)/(b-a) и t2=(yl-y0)/(2m), которые означают масштабы по осям Х и Y соответственно.

График рассматриваемой функции представлен на двух отрезках [а, b] и [-0.1,0.1]. Чтобы построить график другой функции, достаточно задать ее аналитический вид в описании функции (function f).

*Программа 32*

program **grafik;**

uses crt, graph;

var gd,gm, errCode : integer; a,b : real; n : integer;

function f(x:real):real;

begin if **x<>0** then **f:=x\*x\*sin(l/x);**

end;

procedure grafun(xO,xl,yO,yl,n :word;a,b;real);

var h,m,x, tl,t2 : real; i, u,v,xv,yv : word;

begin

h:=(b-a)/n; (поиск максимума f(x)} m:=abs(f (a));

for *i:=l* to n do if m<abs(f(a+i\*h)) then m:=abs(f(a+i\*h)) ;

tl:=(xl-x0)/(b-a);

t2:=(yl-y0)/(2\*m); (построение координатных осей)

setfillstyle(l,15); bar(x0-5,y0-5,xl+5,yl+5) ;

xv:=round(x0-a\*tl); yv:=round((yO+yl)/2) ;

setcolor(l); line(xv,y0,xv,yl);

line(xO,yv,xl.yv) ;

Moveto(xO,yv-round(f(a) \*t2)); (установка курсора в начало графика}

setcolor(3); (построение графика) for i:=l to n do begin x:=a+i\*h; u:=x0+round((x-a)\*tl);

v:=yv-round(f(x)\*t2); lineto(u,v);

end;

end; (конец процедуры } begin

clrscr; write ('введи a,b и n : '); readln(a,b,n); gd^Detect;

InitGraph (gd,gm, ' ' ) ;

errCode:=GraphResult ;

if errCode = grOk then begin

grafun(100,500,50,300,n,a,b);

grafun(550,620,10,100,200,-0.1,0.1);

repeat until keypressed; CloseGraph;

end else writeln( "ezzCode=' ,errCode)

end.

Для изображения поверхностей, определяемых функцией двух переменных z=f(x,y), можно использовать разные способы. Одним из них является метод построения семейства одномерных графиков функции z=f(x.y) от одной переменной х или у различных фиксированных значениях другой. Это может быть хорошей тренировкой для самостоятельной работы по освоению графики.

**Построение движущихся изображений.** Особую ценность в графике представляет организация движения фрагментов рисунка. Наиболее просто это сделать по следующему плану:

• нарисовать фрагментв нужном месте экрана;

• стереть фрагмент, рисуя его цветом фона или используя процедуру cleardevice;

• снова нарисовать фрагмент в другом месте экрана.

Такой способ осуществлен в программе billiard, где два шарика радиусом 5 пикселей разных цветов двигаются с одинаковой скоростью внутри зеленого прямоугольника, построенного с помощью процедуры bar.

Процедура blou измеряет смещение центра шарика от сторон прямоугольника по каждой оси и, если это смещение на следующем шаге цикла станет меньше радиуса, изменяет направление движения, моделируя поведение упругого шара при ударе о стенку.

В программе также рассмотрена ситуация соприкосновения шариков во время их движения. Она решается примитивно просто: каждый шарик меняет направление своего движения на противоположное.

*Программа 33*

program billiard;

uses crt,graph;

var х,y,dx,dy,gd,gm:integer; xl,yl,dxl,dyl:integer;

procedure blow(a,b:integer; var c,d:integer);

begin if (a<107) or (a>523) then c:=-c;

if (b<107) or (b>363) then d:=-d;

end;

begin

gd:=detect; initgraph(gd,gm,''); setcolor(14);

setlinestyle(0,0,1); rectangle(99,99,531,371) ;

setfillstyle(l,3); bar(100,100,530,370) ;

x:=320; y:=240; dx:=2; dy:°2;

xl:=320; yl:=200; dxl:=-2; dyl:=-2;

repeat circle(x,y,5); setcolor(4); circle(xl,yl,5);

blow(x,y,dx,dy); blow(xl,yl,dxl,dyl); delay(10);

if (abs(x-xl)<=10) and (abs(y-yl)<=10) then begin

dx:—dx; dy:=-dy; dxl:=-dxl; dyl:=-dyl; delay(300)

end;

setcolor(3); circle(х,у,5);

x:=x+dx; y:=y+dy;

setcolor(3); circle(xl,yl,5) ;

xl:=xl+dxl; yl:=yl+dyl; setcolor(14)

until keypressed;

closegraph

end.

Еще один способ организации движения на экране, широко применяющийся в компьютерных играх, связан с использованием нескольких экранных страниц. В режиме Vgamed их две, а в режиме Vgalo - четыре. Страницы имеют номера: 0,1,... В любой момент времени одну из страниц можно сделать видимой и посмотреть ее содержимое на экране с помощью процедуры setvisualpage(номep). Визуальная страница обычно пассивна, т.е. на ней нельзя выполнять графические процедуры. Другую страницу можно объявить активной с помощью процедуры setactivepage(номep). Активная страница невидима для пользователя. На ней можно подготовить другой рисунок. В следующий момент можно поменять роли страниц, т.е. визуальную сделать активной и невидимой и на ней рисовать следующий кадр, а бывшую активную сделать визуальной и показать объект в новом месте экрана.

В программе helicopter с помощью страниц моделируется вращение винтов вертолета. Две процедуры verti и vert2 подготавливают рисунки вертолета с разным положением винтов. Изображения выводятся на разные страницы, которые потом по очереди становятся то видимыми, то активными.

В программе организовано также движение вертолета по эллиптической орбите. Центр орбиты - точка (х0,у0) - располагается в центре экрана, числа а=250 и b=130 -горизонтальная и вертикальная полуоси эллипса. Положение вертолета на орбите 'вычисляется по формулам

u=x0+round(a\*cosp), t=y0+round(b\*sinp),

где р - чекущии угол, образованный радиус-вектором точки эллипса с осью абсцисс.

*Программа 34*

program helicopter;

uses crt,graph;

const step=0.01;

var gd,gm,u,t,z,k,xO,yorc,ac,vi:integer; p,a,b:real;

procedure vertl(x,у:integer) ;

begin

cleardevice; setcolor(14);

setlinestyle (0,0,1); setfillstyle(1,3) ;

line(x+12,y,x+36,y);line(x+24,y,x+24,y+8) ;

fillellipse(x+24,y+14,12,7); moveto(x+18,y+20);

lineto(x,y+20); lineto(x,y+14);

line(x+18.y+28,x+30,y+28); line(x+24,y+27,x+24,y+21) ;

end;

procedure vert2(x,у:integer);

begin

cleardevice; setcolor(14);

setlinestyle(0.0,1); setfillstyle(1,3) ;

line(x+22,y,x+26,y); line(x+24,у,x+24,y+8) ;

fillellipse(x+24,y+14,12,7); moveto(x+18,y+20) ;

lineto(x,y+20); lineto(x,y+14) ;

line(x-3,y+14,x+3,y+14); line(x+24,y+27,x+24,y+21) ;

line(x+18,y+28.x+30,y+28);

end;

begin

gd:=vga; gm:=vgamed; initgraph(gd,gm,' ');

x0:=getmaxx div 2; y0:=getmaxy div 2;

z:=l; p:=pi; a:=250; b:=130; ac:=0; vi:=l;

for *k:=l* to 800 do

begin u:=x0+round(a\*cos(р)); t:=y0-round(b\*sin(p));

• setactivepage(ac); setvisualpage(vi);

if z=l then vertl(u,t) else vert2(u,t); z:=-z;

delay(50); p:=p+step; c:=ac; ac:=vi; *vi:=c:*

end; closegraph

end.

### *Контрольные вопросы и задания*

1. Какие бывают режимы графического экрана?

2. Охарактеризуйте возможности процедур модуля Graph.

3. Какие есть способы построения движущихся изображений?

4. Постройте трехмерные столбчатую и круговую *анаграммы* для примеров, рассмотренных в данном параграфе.

5. Изобразите поверхность функции z=Sin(x)+Cos(y), предусмотрев удаление «невидимых линии».

6. Создайте демонстрационную модель идеального газа в замкнутом объеме.

### 3.8. ТУРБО-ОБОЛОЧКИ. ВЕРСИИ ПАСКАЛЯ

**Система** **программирования Турбо-Паскаль 3.0.** После запуска программы turbo.exe на экране появится меню системы

|  |
| --- |
| Logged drive : A  Work fi1е:  Main file:  Edit Compile Run Save eXecute Dir Quit compiler Options  Text: 0 bytes (7BF5-7BF5)  Free: 19472 by tes (7BF6-C806) > |

*Рис. 3.2.* Главное меню Турбо-Паскаля 3.0

Соответствующие команды выполняютсяпри нажатии выделенной большой буквы.

Logged drive: команда «L» осуществляет выбор дисковода: на запрос

(выбор устройства) NEW DRIVE ввести одну из букв от А до Р, oбозначающую номер дисковода;

Work file: команда «W» осуществляет выбор рабочего файла, на   
(рабочий файл) запрос Work file name ввести имя вашего файла;

Main file: команда «М» выбирает основной файл для программ,   
(основной файл) использующих директивы $I: аналогична команде «W»;

Edit команда «Е» вызывает текстовый редактор; если не

(редактор) выбран файл, то будет запрос; инструкцию по работе с редактором см. ниже;

Compile команда «С» осуществляет компиляцию программы,

(компиляция) компиляция проводится в память, режим компиляции (в СОМ или CHN файлы) меняют по команде «О» (см. compiler Options );

Run команда «R» запускает откомпилированную или   
(запуск) неоткомпилированную программу;

Save команда «S» осуществляет запись рабочего (сохранить)  
(сохранить) файла с сохранением предыдущей версии с расширением .ВАК;

EXecute команда «X» осуществляет вызов и выполнение любого

(выполнять) СОМ-файла;

Dir команда «D» осуществляет просмотр директории   
(оглавление файлов);

Quit команда «Q» осуществляет выходиз системы; если файл  
 (прервать) не записан, то будет запрос;

compiler Options команда «О» выбирает режим компиляции   
(опции компилятора) с помощью меню



(команды «М», «С» и «Н» осуществляют выбор режима компиляции в память, в corn-файл и в chn-файл (откомпилированный файл без библиотеки), **команда «Q»** возвращает в основное меню)

Описание языка Паскаль, проведенное в §3, соответствует версии 3. Исключение составляет раздел работы с графикой, в котором использован универсальный модуль Graph для версий 5 и выше. Таким образом, практически все приведенные в главе примеры могут быть реализованы в среде Турбо-Паскаль 3 (и, разумеется, в более поздних версиях)

**Система программирования Турбо-Паскаль 5.5.** Ниже перечисляются основные расширения возможностей версии Турбо-Паскаля 5.5:

•допускается изменение структуры Паскаль-программы, заголовок программы (program) не обязателен, разделы глобальных описаний могут следовать в любом порядке;

• расширен синтаксис описания констант, разрешено определять типизированные константы и производить их инициализацию;

• введены шестнадцатиричные константы;

• расширен список арифметических функций;

• введены типы данных Word, Byte,

• значительно расширен список процедур и функций для работы с файлами и модулями.

Начиная с версии 4.0 в Турбо-Паскале введено понятие модуля (Unit), определены его составляющие части, а также разрешена раздельная компиляция модулей. Имеются стандартные модули Crt, Svstem, dos, Graph, Turbo3. Graph3. Последние модули призваны осуществлять совместимость с предыдущими версиями.

В системе программирования Турбо-Паскаль версии 5.5 создана интегрированная среда разработчика, т е ряд специализированных средств и возможностей, объединенных оконным турбо-меню (контекстное меню). Главное меню интегрированной среды содержит следующие пункты:

File Edit Run Compile Options Debug Break/watch

Система вступает в диалог с пользователем и предоставляет возможность обратиться к контекстной помощи в любой момент.

Как правило, работа в среде начинается с загрузки текста программы или с его формирования в окне редактирования. После редактирования программы осуществляется компиляция и выполнение, которые выполняются при нажатии клавиш Ctrl+F9 (эквивалентно выбору команды run из меню Run).

Во время компиляции программы на экране появляется специальное окно, в котором информируется процесс компиляции. В случае обнаружения ошибки компиляция прерывается и активизируется встроенный редактор, в котором курсор указывает место ошибки. Клавиша F1 в этот момент выводит информацию из службы помощи по диагностике ошибки. Исправив ошибку, можно опять отправить программу на компиляцию. После удачной компиляции программа выполняется и результаты ее работы можно просмотреть нажатием клавиш Alt+F5. В процессе компиляции выявляются лишь синтаксические ошибки. Правильность работы программы необходимо проверять специальными приемами. В интегрированной среде разработчика предусмотрен отладчик, позволяющий осуществлять трассировку программы или ее отдельных участков, прерывать выполнение в заданных точках, следить за изменениями значений различных переменных. Трассировкой называют пошаговое исполнение программы, при котором за один шаг выполняются все операторы одной строки с последующим ожиданием. Очевидно, что трассировка эффективнее работает для исходного текста, в котором в строке помещают не более одного оператора. При этом появляется возможность анализировать результаты работы каждого оператора в отдельности.

Сеанс отладки обычно начинается с команды Trace into (F\*) или Step over (F8) из меню Run. Выборочная трассировка отдельных участков исходного текста осуществляется с помощью команд Toggle breakpoint из меню Break/Watch и Go to cursor меню Run, Для наблюдения за значениями переменных и выражений существует окно просмотра Watch (F6) в меню Debug. В нем отображаются текущие значения выбранного выражения. Сеанс отладки программы необходимо завершать командой Program reset (Ctrl+F2) из меню Run.

**Система программирования Турбо-Паскаль 6.0.** Главное меню интегрированной среды содержит следующие пункты. Е - системное меню;

File - работа с файлами (сохранение, загрузка, связь с операционной системой);

Edit -редактирование текущего файла (стандартные возможности встроенного текстового редактора);

Search - поиск и замена фрагментов текста;

Run -запуск программы на выполнение;

Compile -компиляция программы:

Options - установка опций интегрированной среды;

Debug - установка параметров отладки программы;

Window - работа с окнами;

Help - система помощи и подсказок.

Версия 6.0 является развитием и усовершенствованием версии 5.5. Существенно переработана интегрированная среда пользователя. В ней появилась возможность работать в многооконном режиме, редактируя несколько файлов одновременно. Допускается работа с «мышью». Существенно улучшен сервис для работы с окнами и с системой помощи Help, которая контекстно вызывается из любого окна, а в окне Help допустимы некоторые команды редактирования. Увеличился максимальный размер редактируемого файла и имеется возможность менять цветовую палитру самой интегрированной среды.

Еще одна возможность, появившаяся в Турбо-Паскале 6.0 - объектно-ориентированное проектирование программ. Подобная технология стала возможной благодаря новому типу данных «объект». Объект - это структура данных, содержащая поля данных (подобно записи) различных типов и заголовки методов. Метод - это процедура или функция, объявленные внутри объявления элемента типа «объект». Большую популярность среди программистов приобрела объектно-ориентированная библиотека TurboVision, включающая объекты, управляющие

• перекрывающими окнами с изменяемыми размерами;

• выпадающими меню;

• диалоговыми окнами;

• работой мыши;

• кнопками,полосами скроллинга.

Следует отметить, что сама интегрированная среда Турбо-Паскаля 6.0 разработана с использованием библиотеки TurboVision.

**Система программирования Турбо-Паскаль 7.0.** Главное меню интегрированной среды Турбо-Паскаль 7.0 в дополнение к меню Турбо-Паскаля 6.0 содержит

Tools - инструментальные программные средства.

По сравнению с предыдущей версией в интегрированной среде появилась возможность настройки компилятора на работу в трех режимах: обычном режиме MS DOS (Real), защищенном режиме (Protected) и в режиме операционной среды Windows (Windows). Введена новая команда Object Browser - просмотр модулей, объектов и символов при редактировании исходного текста. Появилось удобное средство - синтаксическая подсветка, которая обеспечивает подсвечивание управляющих структур, зарезервированных слов, идентификаторов, строк и т.п. Новая секция меню Tools предназначена для передачи управления внешним программам и создания собственных инструментальных программных средств. Нажатие клавиш Alt+Fl (или правый щелчок мыши) активизирует локальные меню, чувствительные к контексту (Browse, Edit, Help, Message, Watch).

В версии 7.0 расширена библиотека стандартных модулей. Помимо известных в версии 6.0 модулей Crt, Graph, Graph3, Overlay, String, System, Turbo3 появились WinAPI, WinCrt, WinDos, WinPrn, WinTypes, WinProcs.

### *Контрольные вопросы*

1. В чем состоят основные различия версий Турбо-Паскаля?

2. Какие возможности версии Турбо-7.0 отсутствуют в предыдущих версиях?

### 3.9. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЮ ТУРБО-ПАСКАЛЯ

СТАНДАРТНЫЕ ПРОЦЕДУРЫ И ФУНКЦИИ

Принятые обозначения:

type -любой тип;

string - любой строковый тип;

file -любой файловый тип;

scalar - любой простой тип;

pointer -любой ссылочный тип.

*Замечание:* если в списке параметров не указан спецификатор типа, то в качестве данного параметра процедура допускает параметр любого типа; в некоторых процедурах допускаются несколько вариантов заголовков процедур.

ПРОЦЕДУРЫ ВВОДА-ВЫВОДА

Процедура **Read**

•

Read(var F:file of type; var v:type);

Read(var F:text; var I':intnger);

Read(var F:text; var R:real);

Read(var F:text; var C:char);

Read(var F:text; var S:string);

где F - имя переменной файлового типа, a var I,R,C,S - одна или несколько переменных, совпадающих по типу с типом элементов файла.

Процедура **Readln**

Readln(var F:text; var varl,var2,...varN);

Readln(var varl ,var2,...varN);

Readln(var F:text);

Readln;

где F - имя переменной типа text, a varl...varN - переменные типа Char. Если опущено имя файла, то по умолчанию принимаетсяимя типа INPUT. Если опущея список переменных, то происходит переход на другую строку.

Процедура **Write**

Write(var F:file of type; var v:type);

Write(var F:text; var I:integer);

Write(var F:text; var B:boolean);

Write(var F:text; var C:char);

Write(var F:text; var S:string).

Процедура **Writeln**

Writeln(var F:text; var varl ,var2,... varN);

Writeln(var var varl ,var2,... varN);

Writeln(var F:text);

Writeln.

СТАНДАРТНЫЕ ФУНКЦИИ

Функции арифметические

Abs(x); Arctan (x); Cos (x); Exp (x); Ln (x); Sin (x); Sqr (x); Sqrt (x)

имеют стандартный смысл. Функция

fruc(R:Real):Real;fruc(I:Integer):Real

возвращает значение дробной части своего аргумента.

Функция

int (R:real):real; int (I:integer):real;

возвращает значение целой части своего аргумента.

Функции скалярные

Odd(x); Pred(x); Succ(x)

имеют стандартный смысл.

Функции преобразования

Chr (x); Ord (x); Round (x); Trunc (x) имеют стандартный смысл.

СТРОКОВЫЕ ПРОЦЕДУРЫ И ФУНКЦИИ

Процедура **Delete**(var S:Strina;Pos, Len: Integer) убирает из строки S Len символов, начиная с Pos. Если Len больше длины S, то ничего не происходит. Если Pos+Len больше длины строки, то удаляются все символы до конца строки.

Процедура **Insert**(S:String; var D:String;Pos:Integer) вставляет строку S в строку D перед символом с номером Pos. Если Pos больше длины D, то S и D соединяются. Если длина результата при этом больше максимальной длиныD, то в D записываются только самые левые символы.

Процедура **Str**(I:Integer; var S: String); **Str**(R:Real; var S:String) преобразует I или R из числа в строку и записывает эту строку в S, причем R и I могут записыватьсяформально, т.е. как в процедурах Write и Writeln.

Процедура **Val**(S:String; var R:Real; var P:Integer); **Val**(S:String; var I,P:Integer) преобразует строковую переменную S в число (тип зависит от типа переменной I или R соответственно). Если в строке S ошибок нет. то Р равно 0, иначе значение R (или I) не определено, а Р присваивается номер первого ошибочного символа.

Функция **Concat**(Sl,S2,....Sn:string):String возвращает строку, полученную конкатенацией строк Sl,...,Sn. Если длина результата больше 255, то возникает ошибка времени выполнения.

Функция **Copy**(S:String; Pos,Len:Integer):String возвращает строку, полученную из Len символов строки S, начиная с позиции Pos. Если Pos больше, чем длина строки S, то возвращается пустая строка. Если Pos+Len больше, чем длина строки S, то возвращаются только символы, принадлежащие строке S.

Функция **Length**(S:Strina):Integer возвращает длину строки S.

Функция **Pos**(Pattern,Sourse:String):Integer возвращает номер символа, начиная с которого Pattern входит в Sourse. Если вхождения нет, то возвращается 0.

ПРОЦЕДУРЫ И ФУНКЦИИ ДЛЯ РАБОТЫ С ФАЙЛАМИ

Процедура **Assign**(var F:File: Name:String); Name - имя файла на диске, F – имя файловой переменной. После выполнения этой процедуры Name и F отождествляются. Assign не должна применяться к файлу, находящемуся в употреблении.

Процедура **BlockRead**(var F:File; var Dest:Type; Num:Integer)

F -нетипизируемый файл, Dest -переменная,

Num - количество 128-баитных записей, которые должны быть перенесены из дискового файла в переменную.

Процедура **BlocKWrite**(var F:File; var Dest:Type; Num:Integer) I Num - количество 128-баитных записей, которые должны быть перенесены из переменной в дисковый файл.

Процедура **Chain**(var F:File) активизирует CHN-файл, с которым предварительной процедурой Assign было отождествлено имя F.

Процедура **Close**(var F:File); дисковый файл, отождествленный с переменной F, закрывается и в директорию диска вносятся необходимые изменения.

Процедура **Erase**(var F:File); дисковый файл, отождествленный с переменной F, уничтожается.

Процедура **Execute**(var F:File) загружает и выполняет файл с диска, с которым предварительно процедурой Assign было отождествлено имя F.

Процедура **Rename**(var F:File; Name:String); дисковому файлу, отождествленному с переменной F, присваивается новое имя Name. Rename не должна применяться к уже открытому файлу.

Процедура **Reset**(var F:File) открывает файл F для чтения и ставит указатель на начало файла.

Процедура **Rewrite**(var F:File) открывает файл F для записи, чистит его и ставит указатель на начало файла.

Процедура **Seek**(var F:Fileof type; Pos:Integer) устанавливает указатель файла F на Pos элемент файла.

Процедура **Flush**(var F:File) очищает буфер дискового файла, отождествленного с переменной F. Гарантирует, что следующая операция чтения будет произведена непосредственно с диска. Должна применяться к уже закрытым файлам.

Функция **Eof**(var F;File):boolean возвращает TRUE, если найден конец файла.

Функция **Eoln**(var F:Text):boolean возвращает TRUE, если найден конец строки или конец файла F.

Функция **Filepos**(var: F: File of Type):Integer; **Filepos**(var: F:File):Integer возвращает текущую позицию указателя файла.

Функция **Filesize**(var: F: File ofType):Integer;Filesize(var: F:File):Integer возвращает текущее количество элементов файла.

ПРОЦЕДУРЫ И ФУНКЦИИ ДЛЯ РАБОТЫ СО ССЫЛОЧНЫМИ СТРУКТУРАМИ

Процедура **Getmem**(var P:Pointer; I:Integer) отводит под область памяти, на которую будет ссылаться переменная Р, 1 байт памяти.

Процедура **Mark**(var P:Pointer) запоминает текущий указатель «кучи» в переменной Р.

Процедура **New**(var P:Pointer) отводит место в памяти под переменную того типа, на который указывает Р.

Процедура **Release**(var P.Pointer) ставит указатель «кучи» на значение переменной Р. Переменная Р должна быть предварительно обработана процедурой Mark. Уничтожаются все динамические переменные, расположенные в памяти после последнего вызова процедуры Mark.

Функция **MemAvai**:Integer возвращает размер доступной для динамических переменных памяти.

Функция **Ord**(P:Pointer):Integer возвращает адрес переменной ссылочного типа Р.

Функция **Ptr**(I:Integer):Rointer возвращает ссылку на адрес памяти I.

ПРОЦЕДУРЫ ДЛЯ РАБОТЫ С ТЕКСТОВЫМ ЭКРАНОМ

Процедура **CrtExit** - выдача на экран строки сброса экрана, определенной в инсталляторе.

Процедура **Crtlnit** - выдача на экран строки инициализации экрана, определенной в инсталляторе.

Процедура **CIrEol** уничтожает символы строки от курсора до крайнего правого символа без перемещения курсора.

Процедура **CIrScr** очищает экран.

Процедура **DelLine** - удаление строки, на которой находится курсор, и сдвиг всех строк, находящихся ниже этой строки, на одну вверх.

Процедура **GotoXY** (X,Y:Integer) - позиционирование курсора по координатам **XnY.**

Процедура **InsLine** - вставка пустой строки на место курсора и сдвиг всех строк, находящихся под этой строкой на одну вниз.

ПРОЧИЕ ПРОЦЕДУРЫ И ФУНКЦИИ

Процедура **Bdos**(func,param:Integer) - вызов функции Bdos с номером func, которая заносится в регистр С, и параметром param, который заносится в регистровую пару DE.

Процедура **bios** (func,param:Integer) - вызов функции Bios с номером func и параметром param. который заносится в регистровую пару ВС.

Процедура **Delay** (Ms:Integer) - задержка программы на Ms миллисекунд.

Процедура **FillChar(var** dest; length:Integer;data:Char); **FillChar(var** dest;

length:Integer;data:Byte) - заполнение length байт памяти величиной data. Начало участка памяти - адрес переменной dest.

Процедура **Halt** - аварийный выход из программы.

Процедура **Move**(var sourse.dest:length:Integer) - пересылка length байт, расположенных в памяти, начиная с первого байта переменной sourse, в участок памяти длины length, начинающийся с первого байта переменной dest.

Процедура **Randomise**- инициализация генератора случайных чисел.

Функция **Addr**(var variable):Integer; **Addr**(<function identifier>):Integer;

Addr(<procedure identifier>):Integer возвращает адрес первого байта своего аргумента.

Функция **Bdos**(funk,param:Integer):Byte аналогична соответствующей процедуре. Возвращает значение регистра А.

Функция **BdosHL**(funk,param:Integer):Integer аналогична функции Bdos. Возвращает значение регистровой пары HL.

Функция **Bios**(funk, param :Integer):Byte аналогична соответствующей процедуре. Возвращает значение регистра А.

Функция **BiosHL**(func.param:Integer):Integer аналогична функции Bios. Возвращает значение регистровой пары HL.

Функция **Hi**(I:Integer):Integer возвращает число, младший байт которого содержит старший байт параметра. Старший байт результата равен 0.

Функция **IOresult**:Integer работает после директивы компилятору $1-. Значение 0 соответствует отсутствию любой ошибки ввода-вывода. В противном случае возвращается произвольное ненулевое число.

Функция **KeyPresscd**:Boolean возвращает TRUE, если нажата любая клавишанаклавиатуре, и FALSE в противном случае.

Функция **Lo**(I:Integer):Integer возвращает число, младшийбайт которого содержит младший байт параметра. Старший байт результата равен 0.

Функция **Random**(I:Integer):Integer; Random:Real возвращает случайное число. Для первого варианта вызова это число лежит в интервале от 0 до 1, для второго -от 0 до 1.

Функция **Size0f**(var variable):Integer; **Size0f**(<type identifier>):Integer возвращает размер (в байтах) своего аргумента.

Функция **Swap**(I:Integer):Integer возвращает число, у которого по сравнению с аргументом поменялись местами первый и второй байты.

Функция **UpCase**(Ch:Char):Char возвращает верхнийрегистр данного символа. Если такового нет, то возвращается аргумент.

Функция **ParamCount**:Integer; возвращает число параметров в строке вызова СОМ-файла.

Функция **ParamStr**(n:Integer):String[14]; возвращает n-параметр из строки вызова СОМ-файла.

Процедура **Exit** - выполнение этой процедуры эквивалентно передачи управления на последний END процедуры, функции или основной программы, где выполнена процедура **Exit.**