Лекции
по программированию

Turbo Pascal

Ст. преподаватель Невельская О.В

Содержание лекций

Алфавит языка Turbo Pascal

1 бит – минимальная единица хранения данных

Слово = 16 бит

64 Кб – максимальный размер массива

Набор символов:

1. Прописные – A – Z, строчные – a – z;
2. Цифры 0 – 9, A – F (или a – f) для представления чисел в 16 с/с;
3. Знаки +, -, *,/,=, <, >, [ ], ( ), { }, **:** , **;**, **.**, тире, **‘;**
4. Зарезервированные слова: end, else, if, begin, and …

Строка – набор символов в коде ASCII

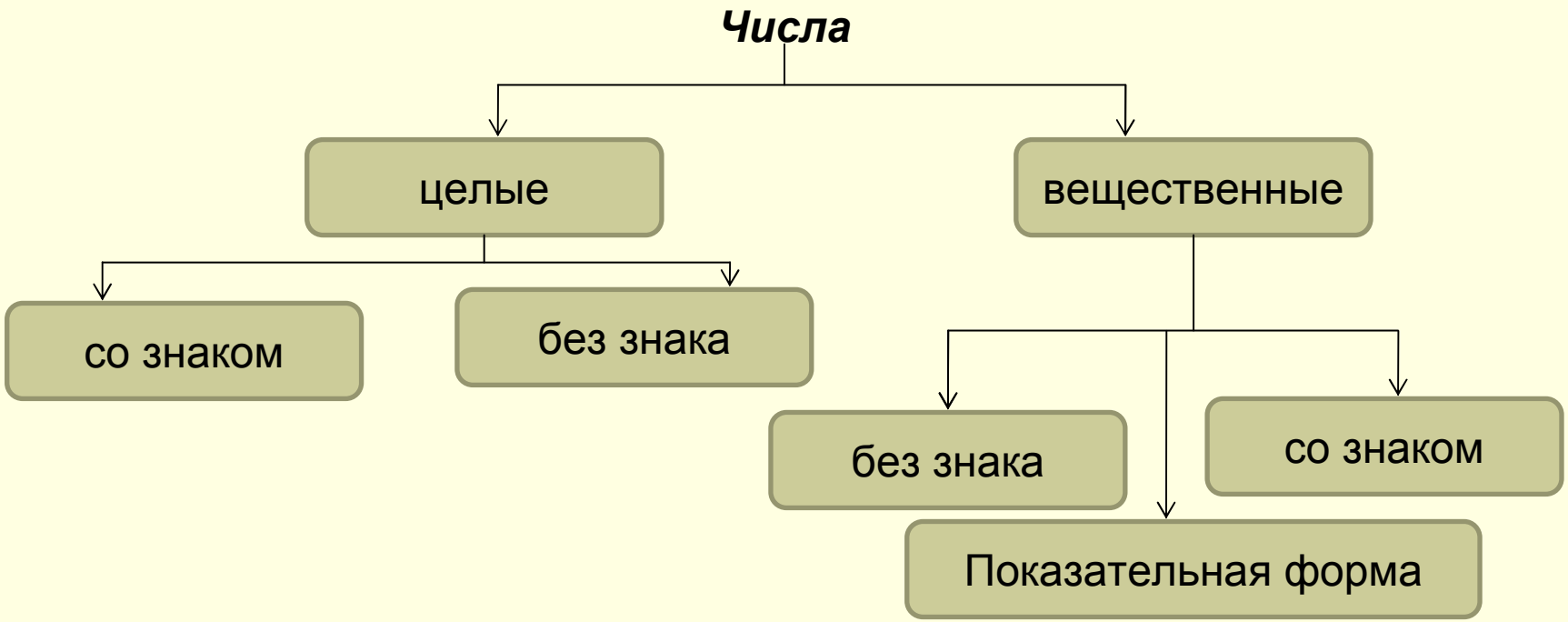
Комментарий – фрагмент текста программы, заключенный { } или (* *).

Разделители: пробел (код 32), знак табуляции (код 09), возврат каретки и перевод строки (код 13 10), управляющие символы 0 – 31.

Алфавит языка Turbo Pascal

Лексема – минимально значимые единицы текста программы (зарезервированные слова).

Идентификаторы – последовательность букв, цифр и знаков подчеркивания (не содержит пробел).



Структура программы

Program [имя программы]

Uses [имя модуля]

Label [описание меток]

Const [описание констант]

Type [описание типов]

Var [описание переменных]

Procedure [имя процедур]

Function [имя функций]

Exports [описание экспортируемых имен]

Begin

[тело программы]

end.

Структура программы

Модуль ***uses***

Uses system – загружается автоматически, включает в себя: файловый ввод/вывод, обработка строк, операции с плавающей точкой.

Uses graph – подключает специальные программы для работы в графическом режиме.

Uses crt – разрешает использование всех возможностей клавиатуры и дисплея (включая: управление экраном, цветом окна, звуковые сигналы).

Uses Dos – поддерживает функции MS DOS (установка тек. Времени, поиск по каталогам файлов).

Uses Printer – позволяет организовать доступ к устройству печати.

Label - (метка) - последовательность цифр 0 – 9999 или символов.

Пример label 10, err;

Begin 10: оператор1; err: оператор2; end.

Структура программы

Const

Константы:

• ***литеры*** – числа (целые, вещественные), символы и строки символов.

Вещественные (real)	Целочисленные (longint) [-2 147 483 648, 2 147 483 648]	Строковые (string)

• ***именованные константы*** – задание фиксированного значения при определении константы в начале программы с присвоением ему имени.
Const <имя константы> = <значение константы>;

Пример: const dr := 9.81; max := 1000; hd := ‘табл.5’; pr := ‘ ‘;

Константы, которые можно использовать без предварительного описания:
False ↔ ложь, ***True*** ↔истина, ***maxint*** ↔32767, ***pi***↔3.14159265358979, ***maxlongint***↔2147483647

Структура программы

•***Типизированные константы.*** Данные константы могут изменять свое значение, поэтому их используют для начального присвоения значений.

Const *<имя константы>* ***: <****тип константы****> = <****значение****>;***

Пример:

const num : integer := 2007; ing : real := 19.90; zag : string [7] :='массив';

-*Структурные типизированные константы* используют для описания массивов, записей, множества.

Пример *описания массивов*:

```
Const a: array [1…5] of integer = (0, 0, 1, 1);
Const mz: array [0…1,0…1] of integer ((0,1),(2,3));
```

Пример *описания записи*:

```
Type
Rec = record
R: real;
B:char;
C: integer;
End;
```

Структура программы

```
Recof = record
I :integer;
S:string;
Z:rec;
End;
Const
        recElem: Rec =(R:3.14; B: true; C:'*');
        RecEl : Recof = (I:32; S: 'Pascal'; Z:( R:3.14; B: true; C:'*'));
```

Переменные

1. Группа целых типов :
• shortint
• integer
• longint
• byte
• word
2. Группа вещественных типов:
• single
• real
• double
• extended
• comp

3. Группа булевых типов
• boolean
• bytebool
•Wordbool
•longbool

4. Символьный тип (char) значения из табл. ASCII
5. Строковый тип (string, pchar) н-р слов и символов + (возврат каретки и перевод строки) или (ноль)
6. Указательный (pointer) ссылка на адрес оперативной памяти
7. Текстовый тип (text) используют для описания текстовых файлов

Группа целых типов

Название типа	Идентификатор	Диапазон чисел	Размер памяти (байт)
Короткое целое со знаком	Shortint	-128 ÷ 127	1
Целое со знаком	Integer	-32768 ÷ 32767	2
Длинное целое со знаком	Longint	-2147483648 ÷2147483647	4
Короткое целое без знака	Byte	0 ÷255	1
Целое без знака	Word	0 ÷65535	2

Группа вещественных типов

Название типа	Идентификатор	Диапазон чисел	Размер памяти (байт)
Вещественное одинарной точности	Single	$1,5*10^{-45} \div 3,4*10^{38}$	4
Вещественное	Real	$2,9*10^{-39} \div 1,7*10^{38}$	6
Вещественное двойной точности	Double	$5,0*10^{-324} \div 1,7*10^{308}$	8
Вещественное повышенной точности	Extended	$3,4*10^{-4932} \div 1,1*10^{4932}$	10
Целое в формате вещественное	Comp	$-9,2*10^{18} \div 9,2*10^{18}$	8

Группа булевых типов

Идентификатор	Значение False	Значение True	Размер памяти (байт)
Boolean	0	Любое Число не 0	1
ByteBool	0		1
WordBool	00		2
LongBool	0000		4

Операции и выражения

Выражения служат для определения действий.

Выражения состоят из операций и операндов.

Делятся по количеству операндов на :
-Унарные X := -7
- Бинарные X := (5+7)*10-25

Выполнение операций зависит от 3-х факторов:
- Приоритета операций,
- Порядка расположения операций,
- Использование скобок.

По приоритету :
1.Унарные операции +, -, not;
2.Бинарные операции *, /, div, mod;
3.Операции с равным приоритетом (выполнение слева направо);
4.Операции низшего приоритета = , <, > , <>, <= , >=.

Арифметические операции

+, - , *, /

Переменная :=функция (выражение)

ABS (числ. Выражение)

EXP (числ. Выражение)

Ln(x)

MOD(x)

ArcTan(x)

Sqr(x) $\rightarrow x^2$

Sqrt (x) $\rightarrow \sqrt{x}$

Cos(x)

Sin (x)

Div (x)

Операции отношения

= <> < > <= >=

Результатом всех этих операций является булевое значение
TRUE(Истина) FALSE(ложь)

При использовании операций отношения для строковых значений, сравнение выполняется посимвольно слева направо согласно значениям кодов символов табл.ASCII.

Логические операции

Not (A) – логическое отрицание

Результат истинен тогда, когда **A** – ложно.

(A)And (B) – логическое умножение

Результат истинен тогда, когда **A** и **B** – истинны.

(A)Or (B) – логическое сложение

Результат истинен тогда, когда **A** или **B** – истинны.

(A)Xor (B) – исключающее или

Результат истинен тогда, когда значения **A** и **B** не совпадают.

Строковые операции

Конкатенация – сложение 2-х символьных строк «+»

Пример: st := ‘file’;

st1 := ‘name’;

st2 := st+st1; write (‘new’, st2);

Сравнение строк.

Если коды равны, то строки считаются равными.

Строка, символы которой имеют большие коды, считается большей.

Если одна строка короче другой, то она считается меньшей (если до этого места строки были равны).

При сравнении учитываются начальные и конечные пробелы.

Операторы передачи управления

1. Условные операторы;
2. Операторы цикла.

Условные операторы – передают управление по условию на ту или иную ветку.

1. If … then …else – результат условного выражения булевый тип False, True

Сокращенная форма:

If <условие> then оператор 1;

If <условие> then **begin**

оператор1;

……..

оператор n; **End**;

Условный оператор

Полная форма:

```
If  <условие>  then оператор 1
                    else оператор 2;

If  <условие>  then begin
                        оператор 1;
                        ……..
                         оператор n; End;
                    else begin
                        оператор 2;
                        ……..
                         оператор n; End;
```

Условный оператор

Расширенная форма:

```
If  <условие1>  then оператор 1
                                  else  if <условие2>  then оператор 2;

If  <условие>  then begin
                                                оператор 1;
                                                ……..
                                                 оператор n; End;
                                  else if <условие2> then  begin
                                                                  оператор 2;
                                                                  ……..
                                                                   оператор n;
End;
                                                          else   begin
                                                                  оператор 3;
                                                                  ……..
                                                                   оператор n;
End;
```

Оператор выбора ***Case***

Case *выражение* **of** *альтернатива*;
else оператор;
end;

Альтернатива
const .. const: оператор;
else оператор;

Case *переключатель of*
константа1: оператор1 ;
константа2: оператор2;
……..
*константа**N**:* оператор**N**;
else оператор к;

end;

Оператор выбора

Пример 1.

```
Begin
……..
        case arrow of
        #72: y:=y+1;
        #80: y:= y-1;
        #75: y:=y+2;
        end;
……..
end.
```

Пример 2.

```
Begin
……..
Case Sym of
‘0’..’9’: writeln(‘Это цифры’);
‘a’..’z’: writeln(‘Это строчные буквы’);
‘A’..’Z’: writeln(‘Это прописные буквы’);
#10, #13, #26: writeln(‘Управляющие символы’);
Else writeln (‘Другие символы’);
End;
……..
end.
```

Оператор выбора

Пример 3.

```
Begin
……..……
        case year of
        -500 .. -400: writeln(‘древнегреческий Аббак’);
        480..500: Begin
                        writeln(‘Первые записи’);
                        writeln (‘c/c’); end;
        1642: writeln(‘Машина Паскаля’);
        Else writeln(‘другие’);
        end;
        end.
```

Оператор выбора

В качестве переключателя используется выражение, которое находится между **CASE** и **OF.**

Результатом этого выражения может быть только значение порядкового типа, общее количество элементов которого не превышает 65535.

Логика работы оператора ***Case***:

вычисленное значение переключателя определяет какой оператор должен выполнится, а остальные опускаются, либо выполняется оператор после **else** , если ни какой оператор не выбран.

Операторы цикла

1. Цикл со счетчиком.

For i:=нач.зна-е **to** N(кон.зн-е) **do** оператор;

For i:=1 **to** N **do** begin
оператор;
……
оператор T;
End;

to →downto

For i:=кон.зна-е **downto** N(нач.зн-е) **do** оператор;

Переменная **I** – целое число.
Данную переменную нельзя использовать до конца цикла.

Оператор цикла со счетчиком

Пример 1.

```
Begin
……..
Sum:=0;
For i:=1 to 10 do sum:=sum+I;
……..
End.
```

Пример 2.

```
Begin
……..
Sum:=0;
For i:=1 to 10 do begin
                    sum:=sum+I;
                    writeln('Сумма=', sum);
                    end;
……..
End.
```

Пример 3.

```
Begin
……..
Sum:=0;
For i:=10 downto 1 do sum:=sum+I;
                    ……..
End.
```

Оператор цикла с предусловием

While <условие> do оператор;

While <условие> do begin
оператор;
…….
оператор n;
end;

Пример 1.
Begin
……
Sum:=0; i:=1;
While i<=10 do begin
sum:=sum+I; i:=i+1;
end;

Оператор цикла с предусловием

Логика работы оператора:

Проверяется результат условия. Если результат условия =False, то цикл не исполняется. Если результат условия = True, то происходит однократное исполнение цикла, а управление передается на оператор While и осуществляется проверка результата выполнения условия.

Оператор цикла с постусловием

```
Repeat
        оператор1;
        ……..
        оператор n;
Until <условие>;
```

Пример 2.

```
Begin
……..
Sum:=0; i:=1;
Repeat
                sum:=sum+I;    i:=i+1;
                Writeln (‘Сумма = ’ , sum);
                Until  i<=10
……..
        end.
```

Оператор цикла с постусловием

Оператор ***Repeat*** допускает написание более 1-ого оператора без использования конструкции ***begin … end;***

Тело цикла исполняется всегда и столько раз, пока результат условия = ***False*** и прекращает выполняться, когда результат условия = ***True.***

Массивы

Массив – это набор переменных, имеющих одинаковое имя (идентификатор) и упорядоченные по номерам (адреса элементов).

Максимальный размер массива – ***64 Кб***.

Доступ к отдельному элементу массива осуществляется путем указания после имени индексного выражения.

Массив характеризуется:

- именем;
- размером (сколько элементов хранится в массиве);
- типом данных (какие данные хранятся в массиве).

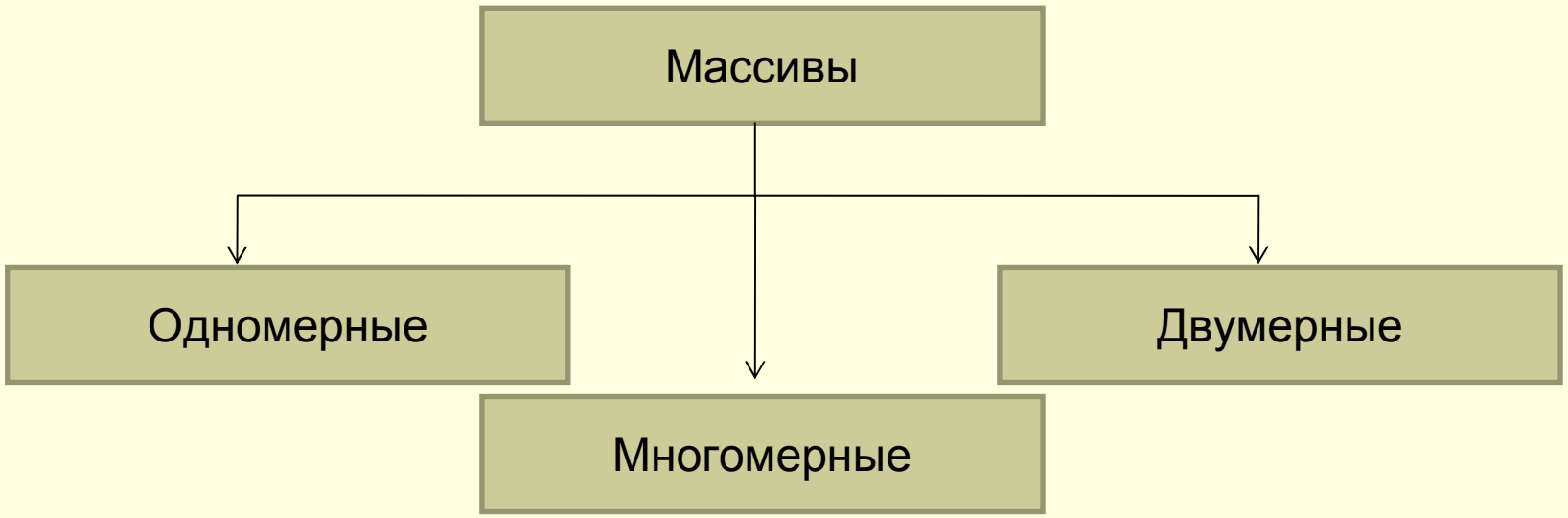
Для объявления массива используют оператор ***Array.***

Mas_name =array [1..n] of integer;
Mas_name1 = array [n, k] of string;

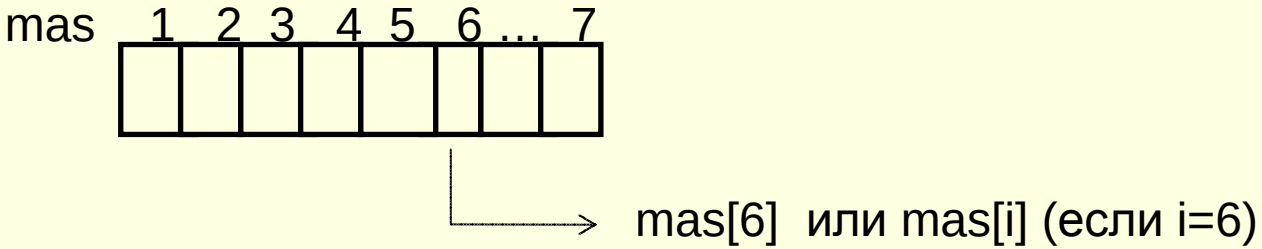
Массивы

Конфигурация массива фиксирована, поэтому к отдельному элементу обращаются с помощью индекса, который может быть записан в переменной.
При организации операции ввода-вывода с массивом следует иметь ввиду, что реальные размеры могут быть меньше, чем зарезервированный при описании размер массива.

При работе с массивами используют ***циклы***.



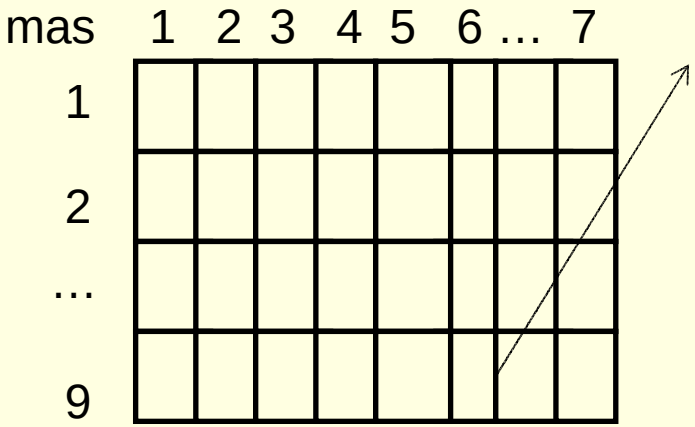
Массивы одномерные



Пример.
Вывести на экран содержимое одномерного массива.

Begin
……
For i:=1 to n do writeln (‘элемент массива mas[‘,I , ‘]=’, mas[i]);
end.

Массивы двумерные



Mas[6,9] или mas[n,k] (если n=6, k=9)

Пример.
Вывести на экран содержимое двумерного массива.

begin
……
For i:=1 to n do
for j:=1 to k do writeln (‘элемент массива mas[‘,I , ‘,‘, j,’]=’, mas[I,j]);
end.

Массивы

Заполнение массивов:
- с клавиатуры;
- с помощью случайных чисел;
- чтение из файла.

Заполнение массива с клавиатуры:
Пример. Заполнение одномерного массива.

```
Program massiv;
Type
Mas=array[n] of real;
Var
A: mas;          I, n: integer;
Begin
        Writeln('Задайте кол-во элементов в массиве');
        Readln(n);
        For i:=1 to n do begin
                Writeln('Введите эле-т массива',A[i] , ':');
                readln(A[i]);       end;
End.
```

Массивы

Пример. Заполнение двумерного массива с клавиатуры.

```
Program massiv_2;
Type
Mas=array[n, k] of real;
Var
A: mas;          I, n: integer;
Begin
        Writeln('Задайте кол-во элементов в строке');
        Readln(к);
        Writeln('Задайте кол-во строк в массиве');
        Readln(n);
        For i:=1 to n do begin
                for j:=1 to k do begin
                Writeln('Введите эле-т массива',A[I, j] , ':');
                readln(A[I, j]);     end;
                end;
End.
```

Массивы

Заполнение массива с помощью случайных чисел:

Пример. Заполнение одномерного массива.

```
 Program massiv;
Type
Mas=array[n] of real;
Var
A: mas;              i, n: integer;
Begin
        Writeln('Задайте кол-во элементов в массиве');
        Readln(n);
Randomaze;
        For i:=1 to n do A[i] := random(1)+100;
End.
```

Массивы

Пример. Заполнение двумерного массива с помощью случайных чисел.

```
Program massiv_2;
Type
Mas=array[n, k] of real;
Var
A: mas;              i, n: integer;
Begin
        Writeln('Задайте кол-во элементов в строке');
        Readln(к);
        Writeln('Задайте кол-во строк в массиве');
        Readln(n);
Randomaze;
        For i:=1 to n do
                for j:=1 to k do  A[i, j] := Random(1)+100;
End.
```

Датчик случайных чисел

Randomaze - процедура
Инициализирует встроенный генератор случайных чисел. Случайное значение получается от системного таймера.

Random – функция
Возвращает случайное число. Если диапазон не задан, то результатом будет вещественное число в интервале [0, 1].

Переменная := random(n)+k;

Строка

Строковая **константа** представляет собой произвольную последова-тельность символов, заключенную в одинарные кавычки, длиной до 32567 символов, например, *‘ Hello’, ‘Добрый день’*

Строковые **переменные** бывают переменной или фиксированной длины. Строка переменной длины (***String)*** представляет собой последовательность длиной до 32567 символов из таблицы ASCII. В памяти под такую символьную переменную отводится количество байт равное количеству символов переменной плюс 4.

Объявление строкового типа переменной длины:
Var Str: String;

Var Hello :String[12];

Строка

Строка представляет собой одномерный массив символов, который имеет фиксированную длину (количество элементов).
Строка имеет 2 разновидности длины:
• ***общая длина строки***, которая характеризует размер памяти, выделяемый строке;
• ***текущая длина строки –*** реальный размер строки.

2 способа реализации обработки строки:
• текущая длина строки указывается в 0-ом элементе строки;
• текущая длина строки фиксируется специальным символом «***конец строки***» и обозначается ***NUL***L , или ***\0***.

1-й способ реализации обработки строк

Каждый символ занимает 1 байт памяти, максимально допустимая длина строки 255 символов.
Простой доступ к значению текущей длины строки.
Var Str: String; - по умолчанию отводится 255 символов, или
Var Hello :String[12]; - отводится 12 символов.

Для получения расширенных возможностей реализации обработки строк, необходимо подключить модули:
- ***Crt;***
- ***String.***

Строковые процедуры и функции

Val(St) – преобразует строковое выражение в его численное представление.
Length(St) в качестве результата дает целое число, рав-ное длине строкового выражения ***St***.
Copy(St, n, m) в качестве результата дает фрагмент строки ***St***, длиной m, начиная с позиции n.
Str(числовое выражение) - преобразует числовое выра-жение в символьное. Если его значение положительно, то к полученной строке слева добавляется пробел.
Delete(St,n,m) удаляет из строки ***St*** подстроку длиной ***m***, начиная с позиции ***n.***
Pos(St,'stroka') производит поиск подстроки в строке и выдает № позиции, или **0** – если подсторка не найдена.
Insert(St, 'stroka',n) добавляет подстроку ***'stroka'*** в строку ***St,*** начиная с ***n.***

2-й способ реализации обработки строк

Текущая длина строки фиксируется специальным признаком, поэтому максимально допустимая длина строки установлена длиной сегмента памяти 65534 байт.

Пример.

```
Program char;
Uses Crt;
Const
str80:array[0, 80] of char = ‘’Cтрока типа PChar;
Var
        p:Pchar;
Begin
ClrScr;
P:=‘str80’;
Write( p);
end.
```

Функции модуля String

StrCat – добавляет одну строку к концу другой строки и возвращает указатель на результирующую строку.
StrCmp – сравнивает 2 строки ***C1*** и ***C2***. Если ***C1<C2***, то результат ***«-»***. Если ***C1 = C2***, то результат ***«0»***, ***C1>C2***, результат ***«+»***.
StrCopy – копирует значение 1-й строки в другую. Возвращает указатель на начало результирующей строки.
StrDispose – уничтожает строку, распределенную ранее с помощью функции ***StrNew***.
StrECopy - копирует значение 1-й строки в другую. Возвращает указатель наконец результирующей строки.
StrEnd – возвращает указатель на завершающий строку символ «***0***».
StrIComp - сравнивает 2 строки по аналогии ***StrCmp*** , но не различая регистры символов.
StrLCat – присоединяет исходную строку к концу целевой строки, при этом длина результирующей строки не превышает заданный максимум. Указатель переносится на конец строки-результата.
StrLCopy – копирует заданное число символов из исходной строки в результирующую и возвращает указатель на результирующую строку.

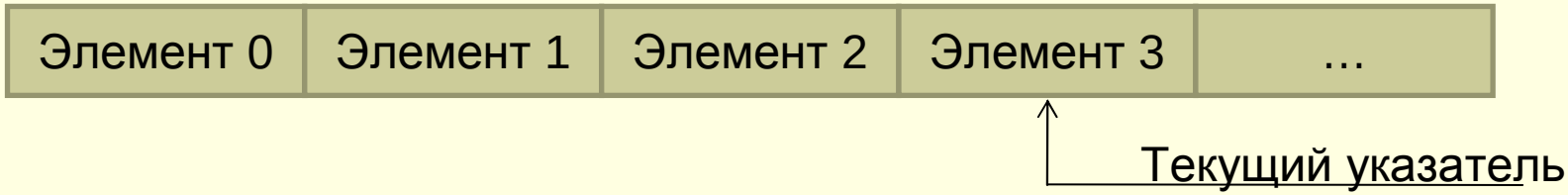
Функции модуля String

StrLen – возвращает длину строки.
StrLowel – преобразует строку в нижний регистр и возвращает указатель на нее.
StrPas – преобразует строку с завершающим символом «***0***» ***в строку-String.***
StrPcopy – копирует ***строку-String*** в строку с завершающим символом «***0***» и возвращает указатель на строку с завершающим символом «***0***».
StrPos – возвращает указатель на первое вхождение заданной подстроки в строке, или «0», если заданная подстрока не найдена.
StrRScan – возвращает указатель на последнее вхождение указанного символа в строке, или «0», если заданный символ не найден.
StrScan - возвращает указатель на первое вхождение указанного символа в строке, или «0», если заданный символ не найден.
StrUpper – преобразует строку в верхний регистр и возвращает указатель на нее.

Файлы

Файл – это совокупность данных, записанная во внешней памяти под определенном именем.

Все элементы файла пронумерованы, при этом начальный элемент имеет ***нулевой номер***.



В любой момент времени программе доступен только один элемент файла, на который ссылается текущий указатель.

Основные действия с файлом:

- чтение;
- запись.

Файлы

Применение файлов:
1.Ввод больших объемов данных, подлежащих обработки. Данный файл может быть подготовлен заранее и применяться многократно.
2.Файл может быть подготовлен другой программой, становясь средством связи программы с внешней средой.
3.Программа, использующая данные из файла, не требует присутствия пользователя в момент фактического использования.

По способу доступа к элементам различают:
1.Файлы последовательного доступа;
2.Файлы прямого доступа.

Файлы

Файлом последовательного доступа называется файл, к элементам которого обеспечивается доступ в той же последовательности, в какой они записывались.

Для поиска нужного элемента необходимо, начиная с нулевого элемента, перемещать указатель до тех пор, пока он не будет указывать на нужный элемент.

Файлом прямого доступа называется файл, к элементам которого осуществляется по адресу элемента.

Для поиска нужного элемента необходимо указать номер нужной позиции.

Файлы

Внимание : нельзя одновременно читать данные из файла и записывать данные в файл, так как для чтения некоторого элемента указатель обработки помещен на данный элемент, а для записи нового элемента этот указатель одновременно должен быть в конце файла.

Компилятор Turbo Pascal поддерживает обработку 3-х типов файлов:

- Текстовые;
- Типизированные;
- Нетипизированные.

Файлы

Ввод/ вывод данных их внешних файлов.

1. Необходимо объявить файловые переменные:
Пример***: Var f, ff: text***; ***text*** – файл из символов.

2. Записать связь логического и физического имени:
Пример: ***assign (идентиф-р файла, ‘ физ. имя‘);***
Где ***иденитф-р файла*** – идентификатор, который объявлен в ***Var*** с типом ***text***; ***физ.имя*** – полный путь и имя файла.

3. Открыть файл и указать для чего открыт:
- Для ввода данных – ***reset***;
- Для вывода данных – ***rewtite***;
- Для последующего вывода или корректировки – ***append***.

4. Закрыть файл **– *close(***идент-р);

Файлы

Пример:
Записать в файл 6 целых чисел.

```
Program test;
Var
f: integer;
……..
Begin
Assign (f, ‘Dant.txt’);
Rewrite(f); {открытие на запись}
……
For i:=1 to 6 do writeln(f,vrem);
……
Close(f);
End.
```

Файлы

Пример:
Открыть файл на чтение символьной информации.

```
Program test;
Var
f: text;
……..
Begin
Assign (f, ‘Dant.txt’);
Reset(f); {открытие на чтение}
……
While not eof(f) do read(f,x);
……
Close(f);
End.
```

Файлы

Внимание:

При чтении данных их файла применяют специальную логическую функцию: ***eof (f)*** – истинна, если указатель файла находится в его конце.

Для создания нового файла с целью записи в него данных применяют процедуру ***Rewrite(f)***;

Если на диске существовал файл с именем, которое указали в процедуре ***Rewrite(f)*** , прежний файл уничтожается.

Если Вы не закрыли файл, то при следующем запуске программы, чтение данных осуществляется с того момента, где остался указатель файла, а не с начала.

Сортировка

Сортировка – упорядочение элементов массива в определенном порядке.
Поиск – нахождение нужного элемента массива по заданным критериям отбора.
Поиск производится в отсортированном массиве.

Группы сортировки:
1.Сортировка с выборкой;
2.Сортировка вставками;
3.Обменная сортировка;
4.Сортировка с распределением;
5.Сортировка с подсчетом;
6.Сортировка слиянием.

Сортировка

1-ая группа сортировки:

Простейший способ сортировки заключается в том, что из входного массива выбирается наибольший элемент и записывается в выходной массив в конец.

Недостатки: большой расход памяти, а время обработки = N^2, где N – количество элементов в массиве.

Пример: 53961274 → ……9

5361274 → ……..79

Метод простого выбора сортировки заключается в том, что выбирается наибольший элемент и записывается на последнее место, а на прежнее место записывается элемент с последнего места.

Пример : 53961274 → 53461279 → 53461279→ 53421679

Сортировка

2-ая группа сортировки:
Метод пузырькового включения
Очередной элемент сравнивается с уже упорядоченными элементами, начиная с большего.
Если элемент < «большего», эти элементы сдвигаются $a_i \rightarrow a_{i+1}$
Если элемент > «большего», то элемент переносится в упорядоченное множество.
$T=N^2/4$

3-ая группа сортировки:
Метод пузырькового всплытия
Метод основан на перестановке соседних элементов в одном массиве.
$T=N^2$
53961274 → 35961274→ 35961274→ 35691274

Сортировка

Быстрая сортировка

Множество разбивается на 3 подмножества

A1 $[a_1,a_i]$ a_i A2$[a_i,a_j]$

a_i - произвольный элемент подмножества (пустое место), которое перемещается по подмножеству.

Неправильно расположенный элемент переносится ***на пустое место*** , а на его место помещается нужный элемент.

_3961274 →5

4396127_

43_61279

4-я группа сортировки

Распределение

Множество разбивается на подмножества так, чтоб любой элемент I-го подмножества > (или<) любого элемента I+1 множества.

Требуется много памяти

251 174 182 521 913 111 617 181 → 174 182 111 181 251 521 617 913

111 174 182 181 251 521 617 913

Сортировка

5-я группа сортировки

Подсчет

Используют вспомогательный массив с N мест. Сначала b_i=1. Далее происходит сравнение каждого элемента. Тот элемент исходного массива, который >, то его из b_i переносят в b_{i+1} при каждом сравнении. После всех сравнений элементы исходного массива переносятся согласно местам. В итоге b_i - отсортированный массив.

6-я группа сортировки

Слияние – объединение 2-х и более упорядоченных множеств в одно упорядоченное множество.

Простое слияние

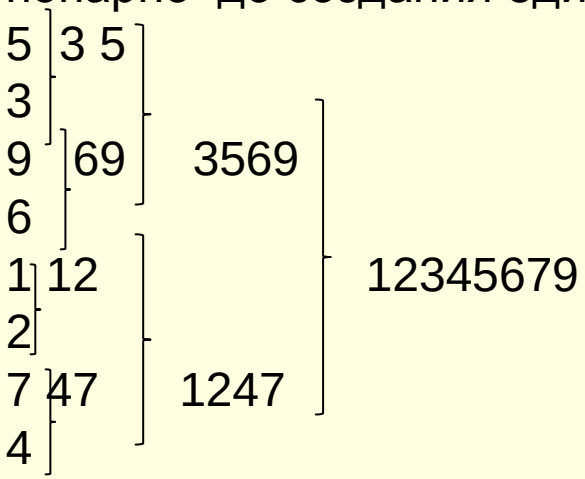
Сравнивают 2 элемента из обоих множеств. Тот что больше переносят 3-е множество, а его индекс увеличивают на 1.

359 ⎫ 1235679
1267 ⎭

Сортировка

Последовательное слияние

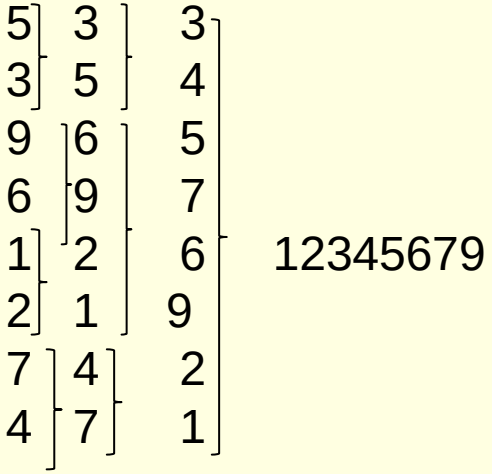
Исходное множество разбивают на N подмножеств. Далее сливаются попарно до создания единого множества. Требуется 2 массива.



Сортировка

Простое двухпутевое слияние

Исходное множество рассматривают с 2-х сторон. Выходное множество последовательно заполняется с 2-х сторон.



12345679

Сортировка

Пример: Блок сортировки одномерного массива

```
……….
{сортировка}
For i:=1 to n-1 do begin
                        min:=i ;
                        for j:=i+1 to n do if mas[j] < mas[min] then min:=j;
                       vr:=mas[i];
                        mas[i]:=mas[min];
                        mas[min]:=vr; end;
……..
```

Модульное программирование

Модуль - это отдельная, полностью независимая от других частей программа.

Каждый модуль имеет главную часть, в которой описываются:

• процедуры,

•функции, константы,

•переменные.

Главный модуль содержит «точку входа», с которой начинается выполнение программы.

В любой программе может быть только один главный модуль.

Преимущество модульного программирования:

1.Модули – это отдельные программы, собранные в одном месте, которые могут одновременно использоваться несколькими программами.

2.Если вносятся изменения в данные модули, то эти исправления автоматически становятся доступны всем программам, которые обращаются к ним.

3.Так как данные модули собирают в одном месте, то легко найти нужный и не будет дубликатов.

Модульное программирование

Внимание. При написании таких модулей необходимо помнить, что ими могут пользоваться другие программисты. При оформлении необходимо соблюдать *дружественный интерфейс*.

Дружественный интерфейс по оформлению программы:
1.достаточное количество поясняющих комментариев к вашему алгоритму,
2.соблюдать принцип «защита от незнающего»,
3.доходчивость,
4.оформление программы.

Основные требования к программе:
1.Дружественность пользователю,
2.Дружественность программисту,
3.Использование приемов управления персональным компьютером.

Модульное программирование

Дружественность пользователю:

1.Программа должна быть рассчитана на пользователя с минимальными знаниями,
2.Должен быть предусмотрен процесс обучения пользователя,
3.Простота в работе с программой,
4.Контроль за действиями пользователя,
5.Проектирование оптимального диалога компьютера с пользователем,
6.Защита программы от ошибок пользователя,
7.Разработка инструкции по использованию программы.

Дружественность программисту:

1.Отладка.
2.Сопровождение программы,
3.Компактность программы,
4.Наличие модулей (подпрограмм: функции, процедуры),
5.Качество документации.

Модульное программирование

Методы управления компьютером:
1.Отображение информации на экране,
2.Прием (ввод) данных с клавиатуры,
3.Выбор метода управления программой,
4.Методы соединения программ,
5.Создание и использование файлов.

Критерии разбиения на модули
•Частота использования фрагментов,
•взаимосвязь фрагментов.

Системная документация:
1.Комментарий в программе до 30%,
2.Документы: список функции, список переменных и их назначение, список массивов, список подпрограмм и их использование, описание использованных файлов,
3.Блок-схемы.

Модульное программирование

Методы управления программой:

1.Меню,
2.Простой выбор – предполагает выбор да/нет,
3.Команды – через операторы приема с клавиатуры имени программы.
4.Сочетание меню и команды.

Стратегия разработки программ:

1.Проектирование сверху вниз.
2.Модульное программирование.

Модульное программирование

Стандарты модульного программирования:

1.Программа делится на независимые части – модули.
2.Модуль выполняет только одну логическую функцию.
3.Размер модуля не должен превышать 100 операторов.
4.Модуль имеет одну входную и одну выходную точку.
5.Взаимосвязи модулей устанавливаются по иерархической структуре.
6.Каждый модуль должен начинаться с комментария.
7.Избегать использования оператора ***go to*** и ненужных меток.
8.Идентификаторы всех переменных и модулей должны иметь смысловые значения.
9.Родственные группы идентификаторов должны начинаться с одинакового префикса.
10.В одной строке записывать не более одного оператора.
11.Запись нескольких строк одного оператора начинается ***с отступа***.
12.Не допускать вложение ***if*** больше 3-х уровней.
13.Избегать использование конструкций с неочевидной семантикой.

Модульное программирование

Шаги программирования:

1.Определение пользователя.
2.Планирование процедур и объектов вывода.
3.Планирование процедур ввода.
4.Планирование модулей.
5.Планирование структур данных и файлов.
6.Выбор метода управления программой.
7.Подготовка системной документации.
8.Разработка документации пользователя.
9.Тестирование и оценка качества программы.

Модульное программирование

Этапы решения алгоритмических задач на компьютере:

1.Постановка задачи.
2.Решить вопросы:
•Как реализовать поставленную задачу,
•каковы способы и методы решения задачи,
•какой вид результата задачи.
3.Разработка алгоритма решения поставленной задачи.
4.Написать программу на языке программирования.
5.Ввод программы и ее отладка.
6.Тестирование программы.
7.Создание документации.
8.Передача документации в эксплуатацию.

Процедуры и функции

Процедуры и функции позволяют один раз записать какой-либо алгоритм и любое количество раз обращается к этому алгоритму из своей программы или других программ.

Процедуры и функции размещают в разделе описаний.

Синтаксис описания процедуры:
<описание процедуры> ::= <заголовок процедуры> <блок>;

Синтаксис описания функции:
<описание функции> ::= <заголовок функции> <блок>;

В блоках могут присутствовать все виды описаний, любые операторы. Завершает их ***end;***
В операторах разрешено использовать:
• глобальные переменные,
•Локальные переменные,
•Параметры.

Процедуры и функции

Синтаксис заголовка процедуры:
<заголовок процедуры> ::= procedure <идентификатор> [(<список формальных параметров>)];

Синтаксис заголовка функции:
<заголовок функции> ::= function <идентификатор> [(<список формальных параметров>)]: <тип функции>;

В функции результат присваивается ***идентификатору функции***, который располагается после списка формальных параметров.
В функции результатом может быть только «одиночное данное» любого типа.
Результат выполнения функции передается в основную программу как значение имени этой функции.
Обращение к функции осуществляется с помощью указателя функции.

Процедуры и функции

Синтаксис указателя функции:
<указатель функции> ::= <идентификатор> [(<список фактических параметров>)]

С помощью элементов из списка формальных параметров происходит описание алгоритма, а с помощью элементов из списка фактических параметров происходит замещение соответствующих формальных параметров и вычисления по данному алгоритму.

Соответствие между фактическими и формальными параметрами:
- Количество фактических и формальных параметров должно быть одинаковым,
- Порядок их следования должен быть идентичным,
- Тип фактических и формальных параметров должен быть одинаковым.

Процедуры и функции

<список формальных параметров> ::= ([var] идентификатор [,идентификатор]…: тип [; [var] идентификатор[, идентификатор]…: тип]…)

Служебное слово ***Var*** определяет способ замещения формального параметра, а именно :

•Наличие ***Var*** указывает ***на передачу по имени*** (для фактических и формальных параметров выделяется одна область памяти. Когда фактический параметр получает какое-либо значение, то формальному параметру передается адрес),

•Отсутствие Var указывает на ***передачу по значению*** (для фактических и формальных параметров выделяются две разные области памяти. Значения фактического параметра пересылаются в область памяти формального параметра Если формальный параметр изменит свое значение, то значение фактического параметра останется прежним).

Процедуры и функции

Внимание:

1.Следует передавать по «значению» параметры, которые задаются в виде констант или выражений,

2.Следует передавать по «имени» параметры, которые определяют результаты и файлы, массивы и записи.

Пример:

1.Заголовок функции ***function F(n: real) : real;***

2.Вызов функции ***per:=F(k);***

Процедуры и функции

Пример: ***составить программу вычисления факториала n!***

```
Program fakt;
Var
Fn, n: integer;
{***************функция  fakt_rs **************}
Fuction fakt_rs(k: integer):integer;
   Var
     p,i : integer;
        begin
             p:=1;
               for i:=1 to k do p:= p*i;
                        fakt_rs := p;
             end;
{************ конец функции ****************}
 begin
Write (‘ введите значение  n’);
Read(n);
Fn:=fakt_rs(n);  writeln(‘fn= ‘, fn:5); end.
```

Процедуры и функции

В процедуре результатом может быть любое количество параметров любого типа.
Результат выполнения процедуры передается в основную программу как значения параметров процедуры.
Обращение к процедуре происходит через оператор вызова процедуры.
Синтаксис оператора вызова процедуры:
<оператор вызова процедуры> ::= <идентификатор> [(< факт.парам.> [,<факт.парам.>]…)];
Все идентификаторы, содержащиеся в списке формальных параметров, являются локальными. Значения локальных переменных в момент входа в тело процедуры не определены, а создаются вновь при обращении к процедуре.

Процедуры и функции

Пример

1.Заголовок процедуры ***procedure prim(n: integer; x: real; var y:real);***

2.Вызов процедуры ***prim(A, B, Z);***

Внимание:

Любая процедура или функция может быть оформлена как самостоятельный внешний файл, который может использовать любой пользователь в своих программах.

Процедуры и функции

Пример: Составить программу вычисления a^m , причем:

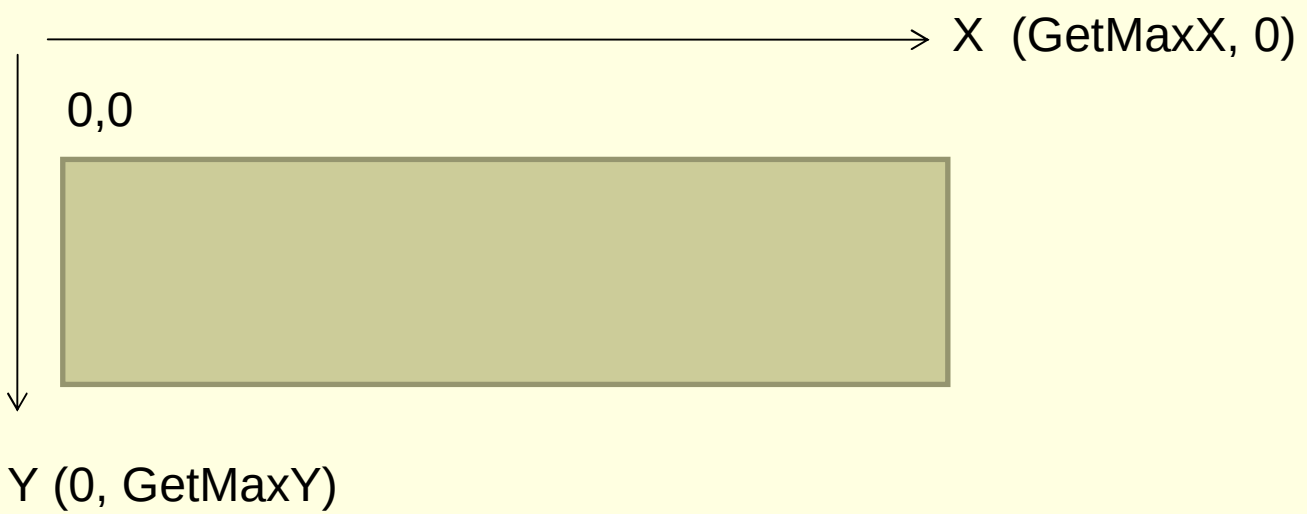
•Если m>0, тогда $y=a^m$. Если m=0, тогда y=1. Если m<0, тогда $y=1/a^m$

```
 program primer;
Var
m: integer;          a, z : real;
{***************процедура ****************}
Procedure step(n: integer, x: real; y: real);
Var
i: integer;
   begin
Y:=1;  for i:=1 to n do y:=y*x;     end;
{*************конец процедуры ***********}
Begin
Writeln ('Введите значения a, m');  readln (a, m);
If m=0 then z:=1
            Else if m>0 then step(m, a, z)
             else step(-m, 1/a, z);
Writeln (a, '  в степени ', m, '= ',z:4:2);      end.
```

Графика

GetMaxX, GetMaxY – стандартные функции модуля ***Graph,*** которые возвращают максимальные координаты по осям X и Y в зависимости от текущего режима видеоадаптера.

Текущий указатель = графический курсор = текстовый курсор



Графика

Инициализация графического режима *InitGraph*, *а закрытие графического режима* ***CloseGraph*.**
Формат ***InitGraph(*** *драйвер:* ***integer,*** *режим:* ***integer,*** *путь к файлу драйвера:* ***string);***

```
Program InitDemo;
Uses Graph;
Var
GraphDriver, GraphMode: integer;
Begin
GraphDriver:= Detect;
InitGraph(GraphDriver, GraphMode, ‘c:\bf\bgi’);
{графические операторы};
Line(0,0,GetMaxX, GetMaxY);
……..
CloseGraph;
End.
```

Графика

Если GraphDriver = 0, то происходит автоматическое определение графической среды, а переменным GraphDriver и GraphMode присваиваются величины, определяющие номер драйвера и номер режима графического экрана.

Detect=0 -автоматическое распознавание графической карты;

CGA=1 - карта CGA;

MCGA=2 - карта MCGA;

EGA=3 - карта EGA;

EGA64=4 - карта EGA64;

EGAMono=5 - карта EGAMono;

Reserved=6 - зарезервировано (не используется);

HercMono=7 - карта Hercules;

ATT400=8 - карта ATT400;

VGA=9 - карта VGA;

PC3270=10 - карта PC3270.

Графика

Если при инициализации графического режима произошла ошибка, то TPascal генерирует соответствующий код ошибки, который возвращается функцией ***GraphResult:***

grOk - нет ошибок

grNoInitGraph – *граф. Режим не инициирован*

grNoDetected - *граф. Средства не найдены*

grFileNotFound – *граф. Файл не найден*

grInvalidDriver – *недопустимый драйвер*

grNoLoadMem – *драйвер не загружен*

grFontNotFound – *шрифт не найден*

grInvalidMode – *недопустимый режим*

grError – *ошибка графики*

grIOError – *ошибка ввода/вывода графики*

Графика

Внимание. Необходимо оформить процесс инициализации отдельной процедурой ***MyGraphInit***, которая может включаться в каждую графическую программу.

```
Program GraphInitDemo;
Uses Graph;
Var
graphDriver, graphMode, ErrorCode: integer;
Procedure MyGraphInit;
Begin
GraphDriver:=Detect;
InitGraph (graphDriver, graphMode, 'c..\bp\BGI');
ErrorCode:= GraphResult;
If ErrorCode <> grOk then begin
Writeln ('Ошибка инициализации граф.режима:=' , GraphErrorMsg(ErrorCode));
Writeln (' Работа программы прервана ');  Halt(1);    end;
End;
……
Begin
MyGraphInit; Line (0,0, GetMaxX, GetMaxY);
CloseGraph;
End.
```

Графика

Присвоение выбранного цвета позиции цветовой гаммы производится с помощью процедуры ***SetPalette***.

Синтаксис: ***SetPalette (ColorNum, Color : word);***

Идентификатор цвета ***Color*** присваивается позиции ***ColorNum*** цветовой гаммы. Изменение цвета на экране будет обнаружено после выполнения процедуры ***SetPalette***.

Идентификаторы цветов могут быть выражены следующим образом:

Black=0 (черный);

Blue=1 (синий);

Green=2 (зеленый);

Cyan=3 (голубой);

Red=4 (красный);

Magenta=5 (лиловый);

Brown=6 (коричневый);

LightGray=7 (светло-серый);

DarkGray=8 (темно-серый);

LightBlue=9 (светло-синий);

LightGreen=10 (светло-зелен.);

LightCyan=11 (светло-голуб.);

LightRed=12 (светло-красн.);

LightMagenta=13 (светло-лилов.);

Yellow=14 (желтый);

White=15 (белый).

Графика

Выбор номера цвета для изображения объектов обеспечивает процедура ***SetColor***.

Синтаксис: ***SetColor(N : word);***

Процедура ***SetBkColor*** изменяет цвет фона на такой цвет, который соответствует позиции N текущей цветовой гаммы.

Синтаксис: ***SetBkColor(N : word).***

Графика

Рисование графических примитивов

Рисование точки (пикселя) с координатами (***X, Y***) цветом с номером ***Color*** выполняется процедурой ***PutPixel(X, Y: integer; Color: word).***

Рисование отрезков прямых линий можно выполнить одной из следующих трех процедур:

Line(X1, Y1: integer; X2, Y2: integer) – рисование отрезка прямой линии, соединяющего точки с координатами (***X1,Y1***) и (***X2,Y2***);

LineRel(dX, dY: integer) - рисование отрезка прямой линии от текущего положения графического курсора на расстояние ***dX*** по горизонтали и ***dY*** по вертикали;

LineTo(X, Y: integer) - рисование отрезка прямой линии от текущего положения графического курсора к точке, имеющей координаты (***X, Y***).

После установления графического режима по умолчанию графическим окном будет весь экран.

Левый верхний угол графического экрана имеет координаты (0,0), а правый нижний (***GetMaxX, GetMaxY***).

Графика

Графическое окно можно установить с помощью процедуры:

SetViewPort(X1, Y1: integer; X2, Y2: integer; Clip :boolean);

Процедура задает графическое окно в виде прямоугольника с координатами противоположных углов (***X1, Y1***) и (***X2, Y2***). ***CLIP*** определяет, должны ли обрезаться выходящие за пределы окна части рисунка: если этот параметр задается константой ***ClipOff*** (или False), то не обрезать, если же константой **ClipOn** (или ***True)***, то обрезать.

Все графические операции выполняются в текущем графическом окне, а координаты графического курсора отсчитываются всегда относительно левого верхнего угла окна.

Графический курсор невидим, но его текущие координаты могут быть определены с помощью процедур ***GetX и GetY***.

Синтаксис: ***GetX : integer; GetY : integer***.

Для очистки графического экрана используется процедура без параметров ***ClearDevice***.

После очистки экрана графический курсор будет установлен в верхнем левом углу текущего графического окна.

Процедура без параметров ***ClearViewPort*** очищает текущее графическое окно и заполняет его цветом первой позиции цветовой гаммы.

Графика

Рисование дуги эллипса с центром в точке (***X, Y***), полуосями ***XRadius*** и ***YRadius***, углами начала ***StAngle*** и конца ***EndAngle*** выполняется процедурой:

Ellipse(X, Y: integer; StAngle, EndAngle : word; Radius, YRadius : word).

Углы выражаются в радианах. Рисование осуществляется против часовой стрелки.

Для ***StAngle = 0*** и ***EndAngle = 2*** будет нарисован полный эллипс.

Изображение закрашенного сектора круга выполняется процедурой:

PieSlice(X, Y : integer; StAngle, EndAngle : word; Radius : word).

Процедурой ***Rectangle(X1, Y1: integer; X2, Y2: integer)*** будет построен прямоугольник, противоположные вершины которого имеют координаты (***X1, Y1***) и (***X2, Y2***).

Процедура ***Bar(X1, Y1: integer; X2, Y2: integer)*** в отличие от предыдущей процедуры строит закрашенный прямоугольник.

Графика

Рисуем линию.

Line ***(x,y, x_1, y_1);***

SetLineStyle ***(SolidLn, 0, NormWidth);*** **-**сплошная линия нормальной толщины

SetFillStyle ***(SolidFill, red);*** **-** закрасить сплошным красным цветом.

Рисуем прямоугольник.

SetColor (red);

SetLineStyle(SolidLn, 0, ThickWidth); - сплошная толстая линия

SetFillStyle(SolidFill, Yellow); - закрасить внутри желтым цветом

Rectangle (300, 10, 400, 70); - нарисовать прямоугольник

FloodFill (350, 50, red); -область 350, 50 внутри закрасить красным цветом

Рисуем пунктирную толстую линию зеленого цвета

SetColor (Green);

SetLineStyle (DashedLn, 0, ThickWidth);

Графика

Рисуем круг синего цвета и заштриховать красным цветом

```
SetColor (Cyan);
SetLineStyle(SolidLn, 0, NormWidth);
SetFillStyle (XHatchFill, Red);
Circle (500, 250, 50);
FloodFill (500, 250, Cyan);
```

Графика

Вывод текста в графическом режиме выполняется процедурами ***OutTex***t и ***OutTextXY*** по умолчанию растровым шрифтом 8x8.
SetColor (Green);
SetTextJustify (LeftText, TopText); **-** выравнивание по левому краю и по верхней линии
OutTextXY(0, GetMaxY div 7, 'DefaultFont, size 1');

Тестирование и отладка программ

Тестирование – важный этап в написании программы.

Тест - это некоторая совокупность исходных данных алгоритма и точное описание всех результатов, которые должны получить на них.

Цель тестирования – проверка и доказательство правильности работы программы.

Принципы тестирования:

1.Каждый линейный участок программы должен быть пройден при выполнении одного теста.

2.Необходимо проанализировать полученные данные после прогона программы на тестовых значениях.

3.Проверить работу программы на неправильных данных.

4.При планировании тестов исходим из предположения наличия ошибок в программе.

Тестирование и отладка программ

Проектирование тестов:
1.Тесты должны быть простыми.
2.Тесты должны проверить каждую ветку программы.
3.Тесты планируют во время разработки алгоритма.
4.Тесты должны быть целенаправленными и систематизированными.
5.Очередной тест должен проверить то, что не было проверено на предыдущем тесте.
6.Усложнение тестов должно быть постепенным.

Чем больше ошибок обнаружено в результате тестирования, тем больше вероятность правильности работы программы.

Хорошим называют тот тест, который обнаружил ошибки, а не удачным считают тот тест, который не обнаружил их.

Тестирование и отладка программ

Этапы тестирования:
1.Проверка работоспособности программы в реальных условиях ее функционирования.
2.Проверка работоспособности программы в экстремальных условиях ее функционирования (на малом количестве данных, на отсутствующих данных, на предельном объеме данных).
3.Проверка работоспособности программы в исключительных условиях ее функционирования (используют пограничные данные).

Ошибки, возникающие при написании программы:
1.Синтаксические – это ошибки при написании команды, вызов несуществующей функции.
2.Семантические – это ошибки, возникающие при неправильном использовании команд (деление на 0, применение операций сложения к символам).
3.Логические – это ошибки, возникающие в результате «правильной» работы программы.

Тестирование и отладка программ

Контроль правильности написания программы:
1.Просмотр текста программы на предмет обнаружения описок, неправильных команд, неправильных логических условий.
2.Проверка правильности вычислительного процесса.
3.Прокрутка. Пошаговое выполнение программы с проверкой .

Отладка – это процесс локализации и исправления ошибок в программе.

Локализация – нахождение места ошибки в программе.

Тестирование и отладка программ

Способы обнаружения ошибок:

1. ***Аналитический*** – просмотр текста программы без ее прогона.
2. ***Экспериментальный –*** запускаем программу на тестовых данных, печатаем и анализируем выходной результат.

Методы отладка:

1. ***Силовые методы*** – это методы применения отладочной печати, автоматических средств отладки программы, печати дампа памяти. и т.д.
2. ***Метод индукции*** – просматриваем программу по симптомам ошибок, определяем данные, имеющие отношения к ошибкам.
3. ***Метод дедукции*** – выдвигаем гипотезу, которая поможет обнаружить и исправить ошибки.
4. ***Обратное движение по алгоритму*** – отладка программы начинается с ошибки и возвращаемся назад в поиске причины ошибки.
5. ***Метод тестирования –*** разработка тестов и проверка работы программы на тестовых значениях.