

PASCAL

2006

**ВВЕДЕНИЕ В ЯЗЫК
ПРОГРАММИРОВАНИЯ**



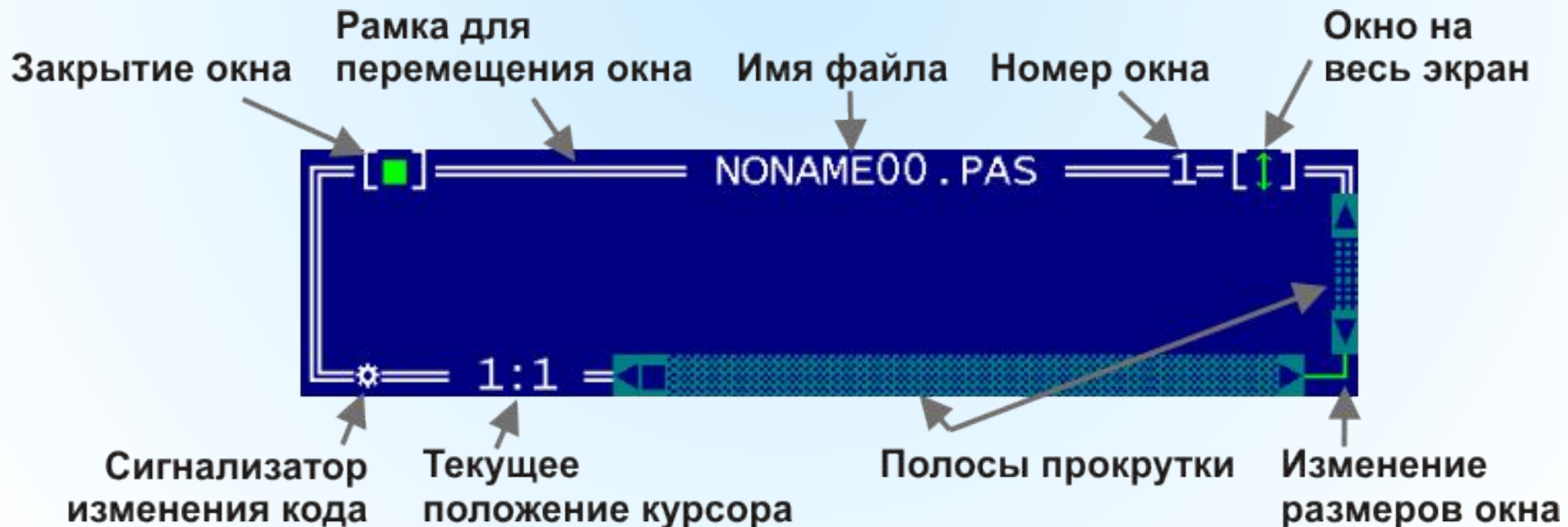
**ШКОЛЬНЫЙ
УНИВЕРСИТЕТ**

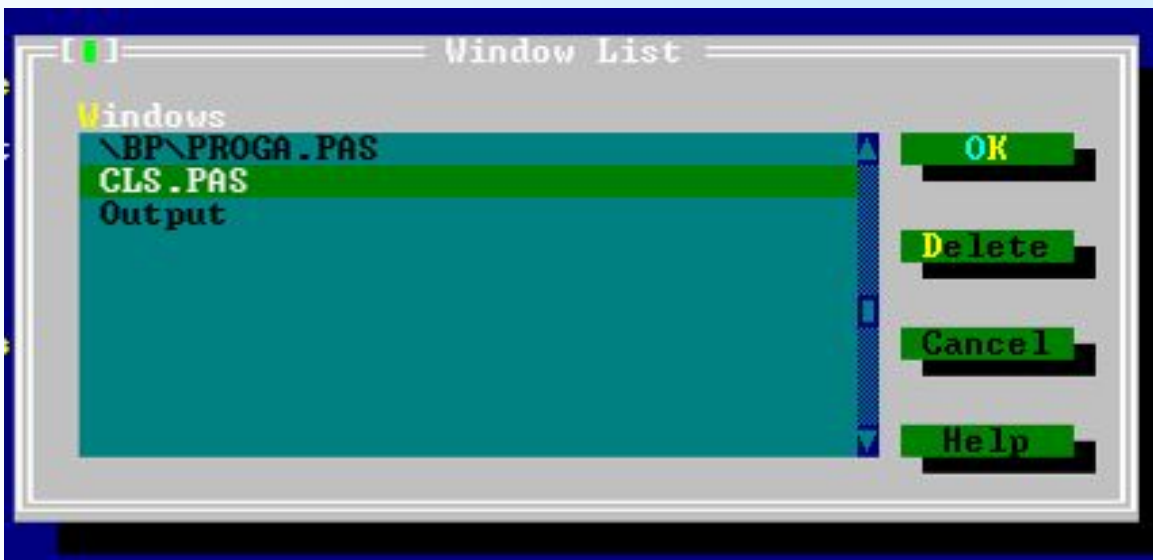
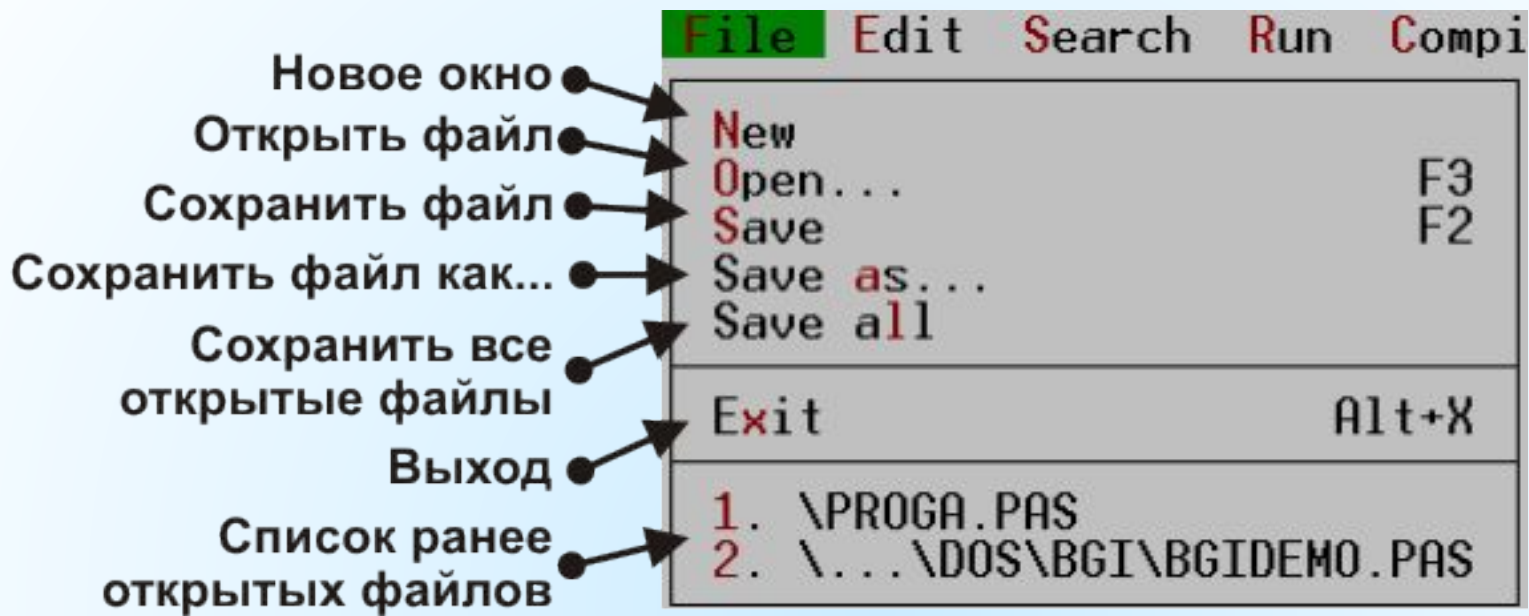
- интерфейс и основные определения;
- структура программы;
- вывод на экран текстовой информации;
- ввод данных с клавиатуры;
- подключение и использование системных модулей;
- работа с основными типами данных (целый, вещественный, строковый, массив, файловый)
- работа с графикой;
- использование подпрограмм (процедуры и функции);
- использование модулей.



Язык программирования Turbo Pascal 7.0 заключён в инструментальную оболочку. Она включает в себя:

- многооконный текстовый редактор;
- компоновщик программ;
- отладчик программ;
- систему помощи;
- компилятор.





Список открытых файлов:

Alt + 0

Быстрый доступ к открытым файлам:

Alt + <№ окна>

- **Зарезервированное слово** – это специальное слово, используемое языком программирования для отделения логических областей программы.
- **Оператор** – это команда языка выполняющая какое-либо действие (проверка условия, организация цикла и пр.)
- **Идентификатор** – это имя, свободно избираемое программистом для элементов программы.
- **Тип данных** – это характеристика идентификатора, определяющая множество значений, которые он может принимать (целые или дробные числа, строки символов, логически выражения и пр.).





Program *Имя_программы;*

Uses

раздел подключаемых модулей;

Label

раздел описания меток;

Const

раздел описания констант;

Type

раздел описания собственных типов данных;

Var

раздел описания переменных;

Begin

Основное тело программы;

End.

Минимальный код:

```
Program My_Program;
```

```
Begin
```

```
End.
```

Для вывода информации на экран, используются операторы: **Write** или **WriteLn**.

```

Program My_Program;
Begin
  Write<'Привет'>;
End.
    
```

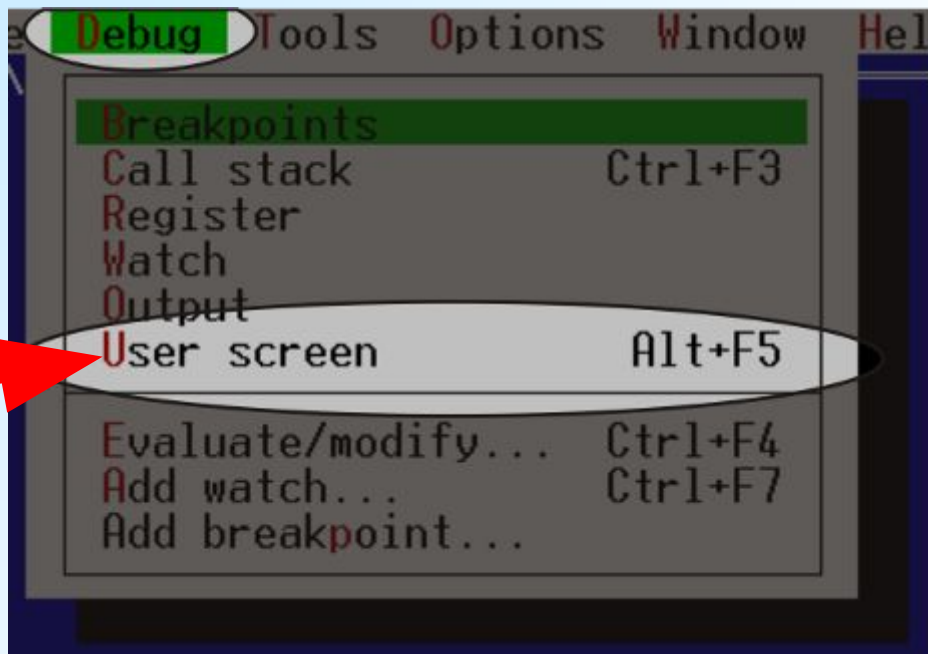
F9 – Компиляция + проверка
Ctrl+F9 – Запуск

F9 → **Ctrl+F9** – избежание ошибок

```

[ ]
Привет
    
```

Отображение результатов
 выполнения программы



Результат применения операторов:

WRITE

```


Program My_Program;
Begin
  Write('Message 1');
  Write('Message 2');
  Write('Message 3');
End.
    
```



WRITELN

```

Program My_Program;
Begin
  WriteLn('Message 1');
  WriteLn('Message 2');
  WriteLn;
  WriteLn('Message 3');
End.
    
```



«Пустой» оператор **WRITELN**
добавляет пустую строку

Выведите информацию на экран так, как показано ниже

```
Hello!  
Nice to see you!  
  
Call me computer.
```

Используемый материал:

Операторы вывода: **Write**, **WriteLn**
«Пустой» оператор **WriteLn**; вставляет пустую строку.



Дополнительные модули расширяют возможности ваших программ, путём введения дополнительных операторов. Модули подключаются в разделе **Uses**.

```
Program My;
```

```
Uses Модуль1,  
      Модуль2 ;
```

МОДУЛЬ 1

Набор
ресурсов 1

• • •

МОДУЛЬ N

Набор
ресурсов N

Модули:

- Системные
- Собственные

Подключённый модуль
с именем **CRT**

```
Program My_Program;
```

```
Uses Crt;
```

```
Begin
```

```
  ClrScr;
```

```
  Write('Hello!!!');
```

```
  ReadKey;
```

```
End.
```

Очистка текстового экрана

Ожидание нажатия на клавишу

GotoXY (X, Y : Integer)

Где X, Y – координата знакоместа на экране.

X может принимать значения от 1 до 80, а Y от 1 до 25.

Например:

```
Program My_program;  
  {Подключение модуля}  
Uses Crt;  
Begin  
  {Очистка экрана}  
  ClrScr;  
  {Вывод данных}  
  GotoXY(1, 1); write('█');  
  GotoXY(80, 1); write('█');  
  GotoXY(1, 25); write('█');  
  GotoXY(80, 25); write('█');  
  {Задержка экрана}  
  ReadKey;  
End.
```

Программа выводит по углам экрана символ “█” (код 177).

TextColor (Color);

Определяет цвет символов.

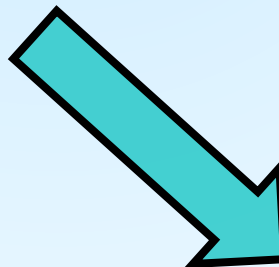
TextBackground (Color);

Определяет цвет знакоместа.

```

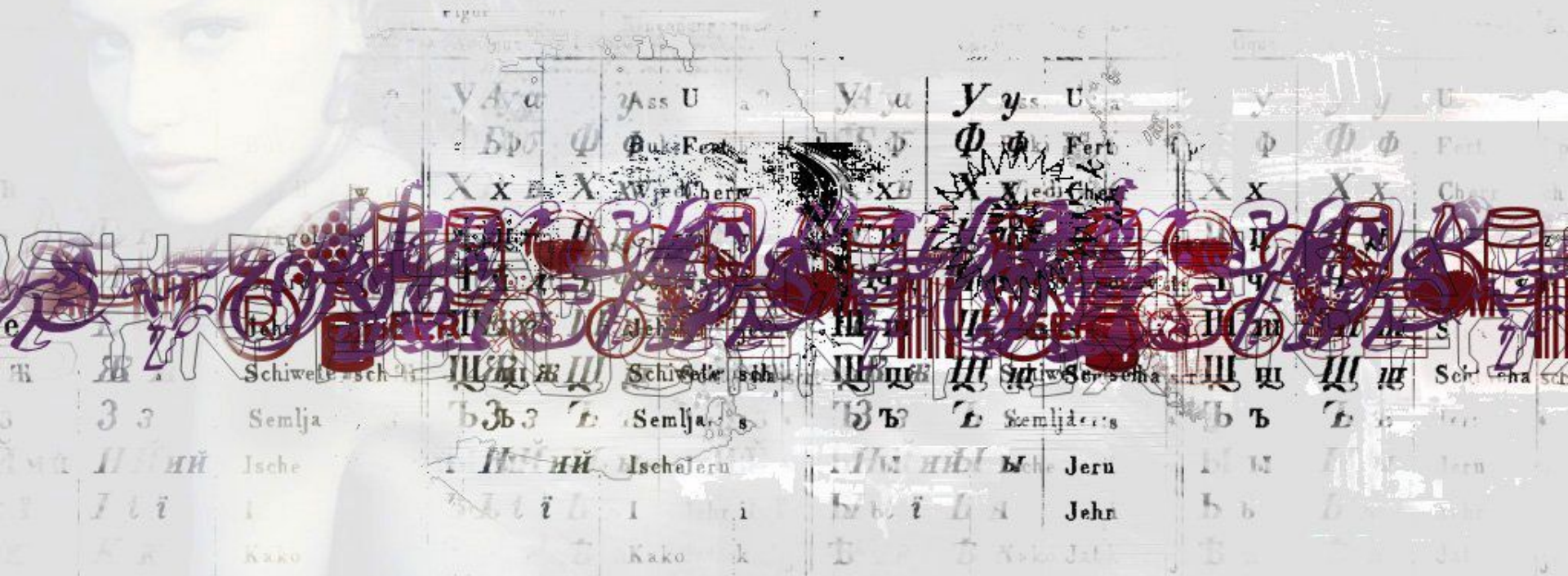
Program MyProgram;
Uses Crt;
Begin
  TextColor (Red) ;
  TextBackGround (Blue) ;
  Write('На дворе ');
  TextColor (White) ;
  Write('трава, ');
  TextColor (Green) ;
  TextBackGround (Yellow) ;
  Write('на траве ');
  TextBackGround (Magenta) ;
  Write('дрова. ');
End.
    
```

Цвета	
0	Black – чёрный
1	Blue – синий
2	Green – зелёный
3	Cyan – циановый
4	Red – красный
5	Magenta – сиреневый
6	Brown – коричневый
7	LightGray – светло-серый
8	DarkGray – тёмно-серый
9	LightBlue – голубой
10	LightGreen – светло-зелёный
11	LightCyan – светло-циановый
12	LightRed – розовый
13	LightMagenta – светло-сиреневый
14	Yellow – жёлтый
15	White – белый



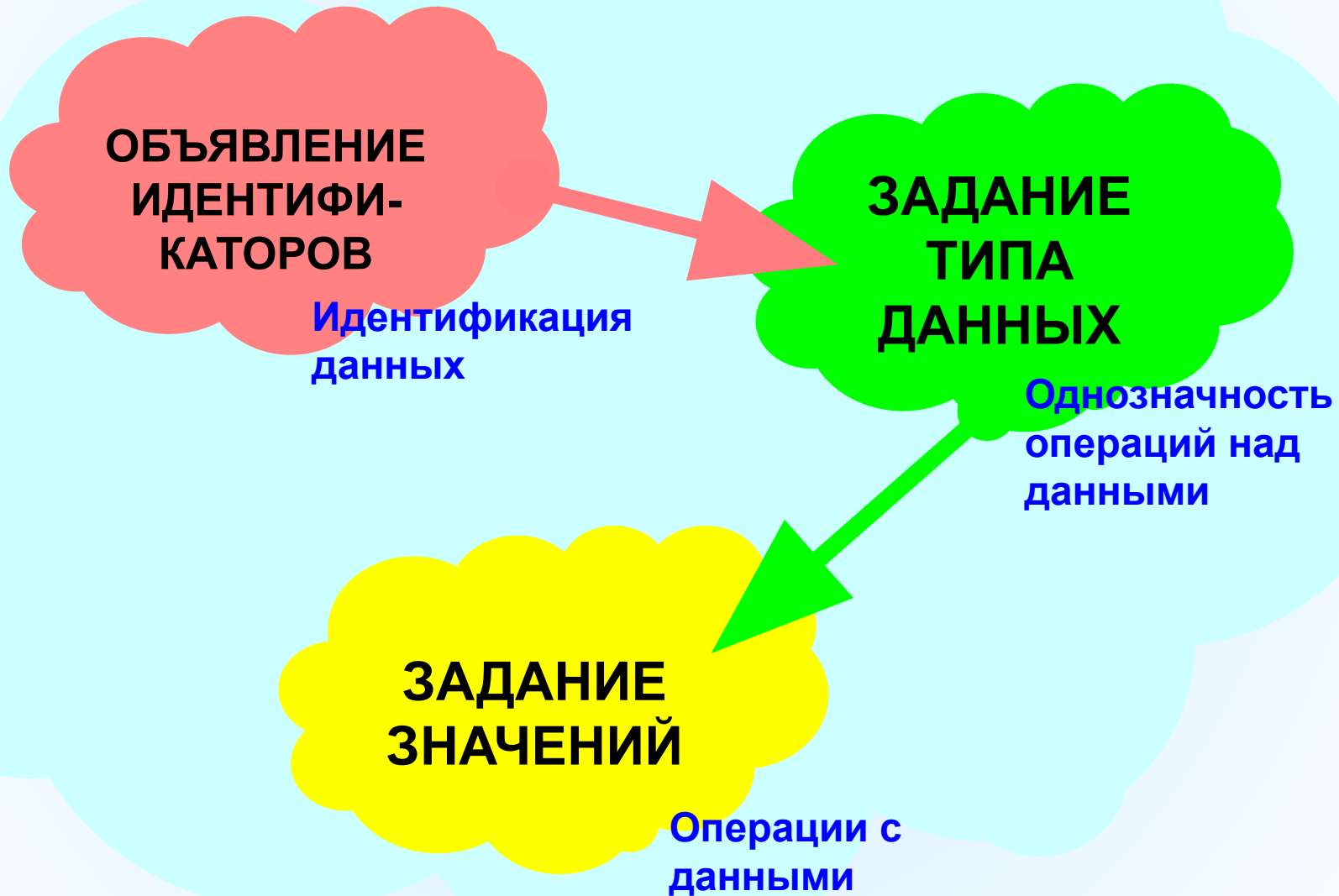
На дворе трава, на траве дрова.

РАБОТА С ДАННЫМИ



Образовательный центр

ШКОЛЬНЫЙ УНИВЕРСИТЕТ



Термин **идентификатор** применяется к константам, переменным, функциям, процедурам, компонентам и другим объектам, определяемым пользователем.

Разрешённые символы:

- латинские буквы;
- цифры;
- знак подчёркивания.

Ограничения:

- не должны начинаться с цифры, но могут начинаться со знака подчёркивания.
- не могут состоять из нескольких слов.
- не могут совпадать с каким-либо из ключевых слов.

В идентификаторах не учитывается регистр символов.

Ошибочные идентификаторы:

Данные { Используются русские символы }

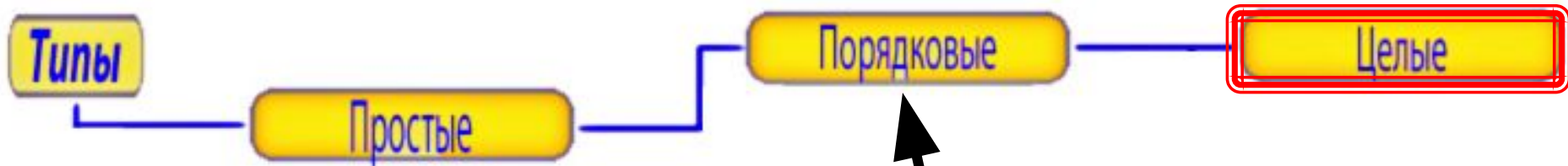
My name { Есть пробел }

2Array { Начинается с цифры }

Var { Совпадает с ключевым словом }

Тип данных – это характеристика идентификатора, определяющая множество значений, которые он может принимать (целые или дробные числа, строки символов, логически выражения и пр.).





Конечный набор возможных значений

ТИП	диапазон
byte	0..255
shortint	-128..127
word	0..65535
integer	-32768..32767
longint	-2147483648..2147483647

! Выход за пределы диапазона приводит к ошибке !

Переменная – это идентификатор, способный хранить какое-либо значение (число, строку и т.п.) и менять его в процессе работы программы.

Синтаксис:

Var *<имя переменной>* : *<тип переменной>* ;

Program *Имя_программы*;

Uses

раздел подключаемых модулей;

Var

раздел описания переменных;

Begin

Основное тело программы;

End.

Зарезервировано место в памяти компьютера под 3-и переменных

```
Program My_program;
```

```
Var A:Integer;  
    X,Y:Longint;
```

```
Begin
```

```
End.
```

Для задания значения переменной, необходимо воспользоваться оператором присваивания :=

Синтаксис записи: *<Переменная> := <Значение> ;*

```
Program My_program;  
  
Var A:Integer;  
    X,Y:Longint;  
  
Begin  
A := 3;  
  
End.
```

В переменную (целочисленную) с именем A заносится значение 3

```
Program My_program;  
Uses CRT;  
Var X, Y, S: Integer;  
Begin  
ClrScr;  
X := 3;  
Y := 6;  
S := X + Y;  
WriteLn (S);  
ReadKey;  
End.
```

Арифметические операции: + - * /
Стандартные операции:
div | mod | sqr

Нельзя использовать с
целыми типами

В переменную с именем S заносится
сумма значений, которые хранятся в
переменных X и Y

Значение, хранящееся в переменной с
именем S выводится на экран

[.]

9

-

1. Напишите программу, которая выводит на экран результат умножения чисел 15 и 20.
2. Напишите программу, которая выводит на экран значение функции $f = 2 \cdot x - 3 \cdot y$, при $x=11$, $y=3$

Используемый материал:

Переменные объявляются в разделе **Var**

Целый тип называется **Integer**

Синтаксис присвоения переменной значения:

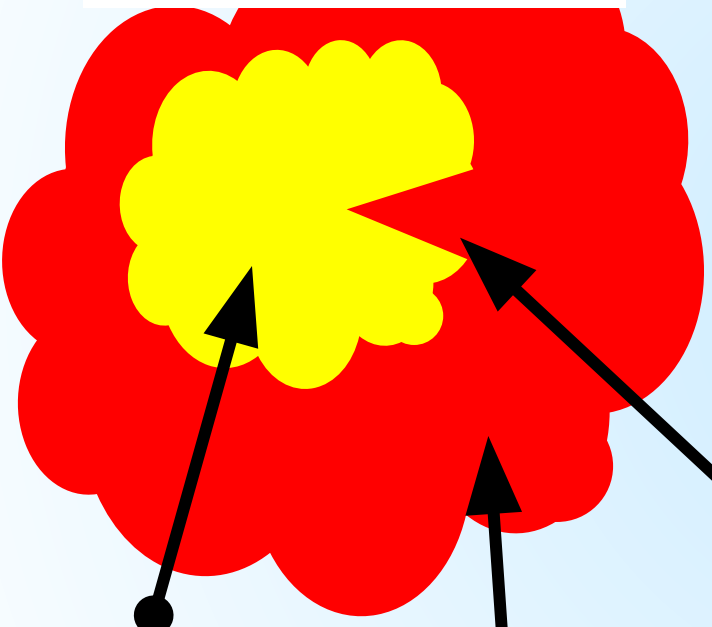
<Переменная> := <Значение> ;

После каждого оператора ставится знак **;** (кроме **begin** и последнего **end**)





Множество значений



Тип	диапазон
Real	2.9E-39 .. 1.7E+38
Single	1.5E-45 .. 3.4E+38
Double	5.0E-324 .. 1.7E+308
Extended	3.4E-4932 .. 1.1E+4932
Comp	E-263+1 .. E263-1

Односторонняя совместимость:
 Вещественный ► Целый
 Целый ✗ Вещественный

! Выход за пределы диапазона и несоблюдение правил совместимости приводит к ошибке !

```
Program My_Program;  
Uses CRT;  
Var X,Y,S:Real;  
Begin  
  ClrScr;  
  
  X := 3.576;  
  Y := 0.5;  
  S := X / Y;  
  
  Write(S:1:2);  
  
  ReadKey;  
End.
```

Стандартные операции:

pi | Sqrt | Sin | Cos | Abs | Exp | Ln
Round | Trunc (вещественный → целый)

Например (X, S – тип Real):

```
X := pi/3;  
S := Sin( X ) * 100;  
Write ( Round(S) );
```

В переменную с именем S заносится отношение значений, которые хранятся в переменных X и Y

Позиция числа и число символов в дробной части числа

E – обозначает степень числа.
 $5.6E-5 = 5.6 \cdot 10^{-5}$

7.15

Строки – упорядоченный набор символов.

Строки заключены в апострофы.

Строки не совместимы с целыми и вещественными типами.

Тип	Диапазон
String	255 символов

{ Основные операторы для строк }

- + { конкатенация }
- **Length (S)**; { длина строки }

Например:

```
X := 'Вася';
Write( 'В вашем имени',
      Length(X),
      'букв.' );
```

```
Program My_Program;
```

```
Uses CRT;
```

```
Var S,Z:String;
    X:Integer;
```

```
Begin
```

```
  clrscr;
```

```
  X := 10;
```

```
  S := 'Вася';
```

```
  Z := 'Меня зовут ' + S;
```

```
  writeln( Z );
```

```
  writeln( 'Мне ', X, ' лет' );
```

```
  ReadKey;
```

```
End.
```

```
= [■] =
```

```
Меня зовут Вася
Мне 10 лет
```

~~S := X;
X := S;~~

Напишите программу, которая выводит на экран результат деления чисел **12.89** и **0.22** с отображением только трёх значащих цифр после запятой в следующем формате:

первое число **разделить на** *второе число* = *результат*

Write (X:1:3, ' razdelit na ', y:1:3, ' = ' z:1:3)

Используемый материал:

- Строковый тип называется **String**
- Строки заключаются в опострафы
- Конкатенация строк происходит через знак **+**
- В операторе вывода на экран, разные типы отделяются запятой



Для того, чтобы ввести информацию с клавиатуры, необходимо воспользоваться оператором ввода: **Read** или **ReadLn**.

Синтаксис:

Read (N1, N2, ... Nn) ;

Где **N1, N2, ... Nn** – переменные (целые, вещественные, строковые)

В переменную **X**, заносится значение введенное с клавиатуры

- После ввода значения, необходимо нажать клавишу **Enter**
- Если переменных в операторе указано несколько, то они вводятся через **пробел**, либо через нажатия клавиши **Enter**

```
Program My_program;  
Uses CRT;  
Var X:Integer;  
Begin  
  clrScr;  
  Write ('Введите число: ');  
  ReadLn (X);  
  WriteLn ('Вы ввели число ', X);  
  
  ReadKey;  
  
End.
```

[■] _____ Outp
Введите число: 4
Вы ввели число 4

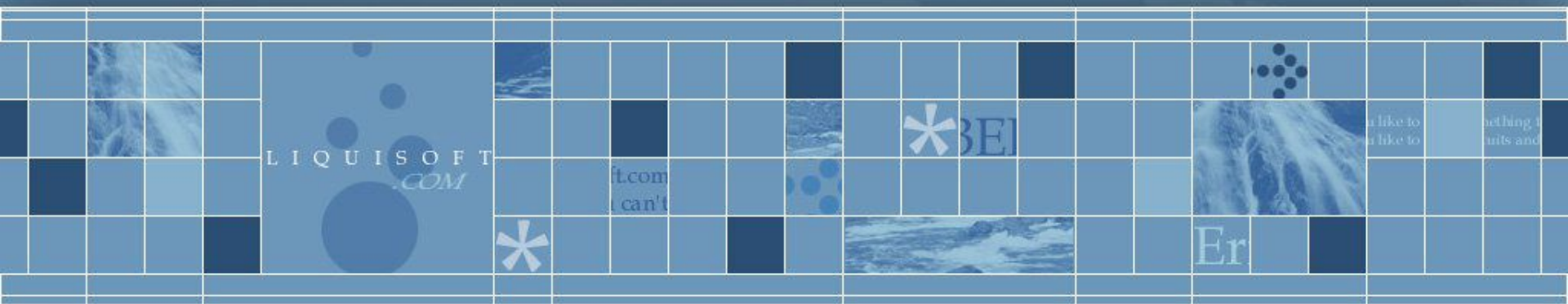
1. Напишите программу «мини-калькулятор», которая вычисляет сумму двух чисел, введенных с клавиатуры.
2. Напишите программу, которая спрашивает как Вас зовут, а после того как Вы напишите свое имя выводит приветствие. Например, если Вы ввели имя **Ваня**, то программа выведет фразу: **Привет, Ваня!!!**

Используемый материал:

Операторы ввода: **Read**, **ReadLn**



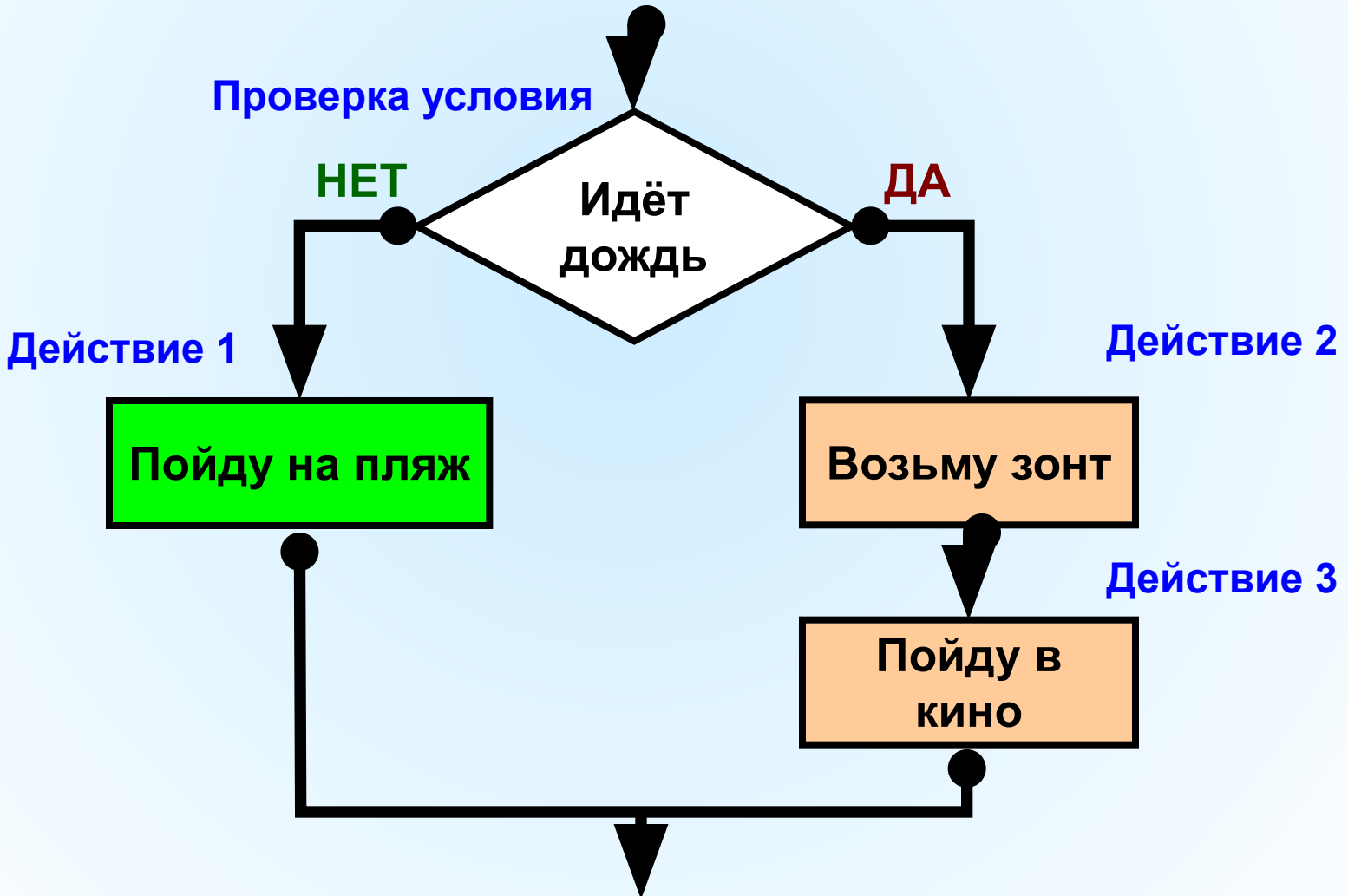
ОПЕРАТОРЫ



Образовательный центр

ШКОЛЬНЫЙ УНИВЕРСИТЕТ

Условный оператор реализует «ветвление», т.е. изменяет порядок выполнения операторов в зависимости от истинности или ложности некоторого условия.



Операции сравнения:

- > - больше
- < - меньше
- = - равно
- >= - больше или равно
- <= - меньше или равно
- <> - не равно

Логические операции:

- Not - Не
- And - И
- Or - Или

Например:

Если Мой рост > Роста Пети, **То** я выше Пети

Если (идёт дождь) **Или** (идёт снег), **То** я останусь дома

Если Не ((идёт дождь) **И** (ветер)), **То** я пойду гулять



При использовании логических операций, условия заключаются в скобки

Краткая форма:

If *<условие>* **then** *<оператор>*;

Например, если в переменной **X** значение меньше чем 0, тогда в эту же переменную записывается значение 1:

```
If x<0 Then x := 1;
```

После слов **Then** и **Else** можно использовать только один оператор

Полная форма:

If *<условие>* **then** *<оператор_1>*
else *<оператор_2>* ;

Перед словом **Else**, знак ; отсутствует

Например:

```
If x>5 Then x := x - 1  
Else x := x + 1;
```

1. Напишите программу, которая получает на вход сумму в долларах и сумму в евро, а затем выводит на экран в какой валюте сумма больше (1 доллар=28 руб., 1 евро=35 руб.)
2. Напишите программу, которая получает на вход три числа, а затем выводит на экран максимальное из них.

Используемый материал:

- Формы записи условного оператора:

```
if <условие> Then <оператор>;
```

```
if <условие> Then <оператор_1>  
    Else <оператор_2> ;
```

- Перед **Else** знак ; не ставится
- Операции сравнения: > < = <> >= <=
- Логические операции: **Not Or And**



Если после слов **Then** или **Else** необходимо записать несколько операторов, то их заключают в **операторные скобки (составной оператор)**.

Операторные скобки начинаются словом **Begin**, а заканчиваются словом **End;**

Например:

```
If Z > 0 Then Begin
    X := 1;
    Y := -1;
    WriteLn( 'Информация принята' );
End
Else
    WriteLn( 'Ошибка' );
```

Оператор выбора используется для замены конструкций из вложенных условных операторов.

Синтаксис записи

Case *<порядковая_переменная>* **of**

<значение_1> : <оператор_1> ;

<значение_2> : <оператор_2> ;

.....

<значение_N> : <оператор_N>

Else *<оператор_N+1> ;*

End;

В операторе выбора
можно использовать
операторные скобки

Не обязательная строка

Case *Рост ученика* **of**

16..30 : Вы ученик начальных классов ;

31,32,33 : Вы учитесь в классе 5-6 ;

34..50 : Вы старшеклассник

Else *Вы явно не ученик ;*

End;

Для перечисления значений используется **запятая**, для выбора диапазона – **двоеточие**

```
Case x of
```

```
  -128..-1: writeln( 'Отрицательные' );
```

```
  0:      writeln( 'Ноль' );
```

```
  1..127:  writeln( 'Положительные' )
```

```
Else WriteLn( 'Выход из диапазона' );
```

```
End;
```

Напишите программу, которая получив число, выводит на экран соответствующий день недели. Например, ввели число **2**, программа должна вывести **‘Вторник’**.

Если число выходит за пределы **1..7**, то должно быть выведено сообщение, что такого дня нет.

Используемый материал:

Оператор выбора:

```
Case <порядковая_переменная> of  
  <значение_1> : <оператор_1> ;  
  .....  
  <значение_N> : <оператор_N> ;  
Else <оператор_N+1> ;  
End;
```



Циклическими называются алгоритмы, у которых выполнение некоторых операторов осуществляется многократно с одними и теми же модифицированными данными (например, процедура умножения чисел через их многократное сложение).

В языке Паскаль имеются три оператора цикла:

- **For** (цикл с параметром или на заданное число повторений)
- **While** (цикл ПОКА)
- **Repeat** (цикл ДО)

Если **число повторений известно**, то лучше воспользоваться оператором цикла **с параметром**.

Цикл на заданное число повторений с *возрастающим* или *убывающим* значением параметра

```
For {парам} := {нач_зн} To  
    {кон_зн} Do  
    {оператор} ;
```

Тело цикла

Замечания

- Параметр – целый тип;
- В цикле можно использовать операторные скобки;
- В теле цикла нельзя менять параметр цикла;
- Параметр цикла увеличивается на единицу;
- Начальное значение > конечного, иначе тело цикла игнорируется;
- Для уменьшения параметра, за место **To**, использовать **DownTo**.

```
Program My_program;  
Uses CRT;  
Var f:Integer;  
Begin  
    clrScr;  
    For f := 1 to 4 do begin  
        Write (f, ' ');  
        WriteLn (f*f)  
    End;  
    ReadKey;  
End.
```

```
1 1  
2 4  
3 9  
4 16
```

Написать программу, которая запрашивает целые числа (a , b), причём $a < b$, а затем выводит на экран сумму чисел от a до b .

```
Program My_program;
Uses CRT;
Var a,b,s,f:Integer;
Begin
  ClrScr;

  Write ('Введите начальное число: ');
  ReadLn (a);
  Write ('Введите конечное число: ');
  ReadLn (b);

  S:=0;
  For f:=a to b Do
    S := S + f;

  Write ('Сумма чисел от ', a, ' до ', b, ' равна ', S);

  ReadKey;
End.
```

Обнуляем
переменную

В S записываем
её предыдущее
значение и
прибавляем к
ней значение
переменной f

```
[■] Output
Введите начальное число: 2
Введите конечное число: 10
Сумма чисел от 2 до 10 равна 54_
```

1. Написать программу, которая 15 раз выводит на экран ваше имя и фамилию (в столбик).
2. Написать программу, которая выводит на экран таблицу квадратов первых 10 чисел.

Число Квадрат

1 1

2 4

... ..

3. Написать программу, которая выводит на экран таблицу Пифагора

	1	2	3	4	5	6	7	8	9
1	1	2	3	4	5	6	7	8	9
2	2	4	6	8	10	12	14	16	18
...

Используемый материал:

Оператор цикла **For**:

For <парам> := <нач_зн> **To** <кон_зн> **Do** <оператор> ;

- Параметр – целый тип (обычно, **Integer**);
- В цикле можно использовать операторные скобки;
- Параметр цикла увеличивается на единицу



Цикл **While** сначала проверяет **условие**, и только если оно истинно выполняет тело цикла.

```
While {условие} do  
  {оператор};
```

- В теле кода, написанном ниже цикл не выполнится ни разу:

```
x:=1;  
While x>1 do  
  x:=x-1;
```

- Можно получить бесконечный цикл. Например:

```
x:=1  
While x>0 do  
  x:=x+1;
```

Программа, вывода на экран суммы чисел от **a** до **b**.

Цикл работает, пока изменяющаяся переменная **f** не станет больше значения **b**

```
Program My_program;
Uses CRT;
Var a,b,s,f:Integer;
Begin
  ClrScr;

  Write ('Введите начальное число: ');
  ReadLn (a);
  Write ('Введите конечное число: ');
  ReadLn (b);

  s:=0;
  f := a;
  While f<=b do Begin
    s := s + f;
    f := f + 1;
  End;

  Write ('Сумма чисел от ', a, ' до ', b, ' равна ', s);

  ReadKey;
End.
```

While not keypressed do begin

Delay(2000);

End;

[■] Output

```
Введите начальное число: 2
Введите конечное число: 10
Сумма чисел от 2 до 10 равна 54
```

1. Космонавты решили на луне садить берёзы, причем каждый год увеличивать число берёз в два раза, в первый год посадили 3 берёзы. Выведите на экран через, сколько лет число берёз превысит 10 000.
2. Напишите программу, которая определяет максимальное из введенных чисел с клавиатуры (ввод чисел заканчивается числом 0). Ниже представлен рекомендуемый вид экрана:

Введите числа. Для завершения ввода введите 0.

89

15

0

Максимальное число 89.

Используемый материал:

Оператор цикла **While**:

While <условие> **do** <оператор> ;

Цикл **While** сначала проверяет условие, и только если оно истинно выполняет основное тело цикла.



Цикл **Repeat** сначала выполняет тело цикла, а лишь затем проверяет условие

```
Repeat  
    {тело_цикла}  
Until {условие};
```

Нет необходимости в цикле **Repeat** использовать составной оператор, т.к. данная конструкция предусматривает выполнение не одного, а нескольких операторов, заключенных между словами **Repeat** и **Until**.

Программа, вывода на экран суммы чисел от **a** до **b**.

```
Program My_program;
Uses CRT;

Var a,b,s,f:Integer;

Begin
  ClrScr;

  Write ('Введите начальное число: ');
  ReadLn (a);
  Write ('Введите конечное число: ');
  ReadLn (b);

  s:=0;

  F := a;
  Repeat
    S := S + f;
    F := F + 1;
  Until f>b;

  Write ('Сумма чисел от ', a, ' до ', b, ' равна ', S);

  ReadKey;
End.
```

Цикл работает, пока изменяющаяся переменная **f** не станет больше значения **b**

[■] Output

```
Введите начальное число: 2
Введите конечное число: 10
Сумма чисел от 2 до 10 равна 54
```

For For f:=a to b Do
S := S + f;

While F := a;
while f<=b do Begin
S := S + f;
F := F + 1;
End;

Repeat F := a;
Repeat
S := S + f;
F := F + 1;
Until f>b;

Выбор модели цикла, зависит лишь от удобства его использования в конкретной ситуации.

Мы практически всегда можем вместо одного вида цикла воспользоваться другим

Вычислите значение функции $y = x^3 - x^2 + 16x - 43$ для x изменяющегося в диапазоне от -4 до 4 включительно с шагом 0,5.

Используемый материал:

Оператор цикла **Repeat**:

Repeat <тело_цикла> **Until** <условие>

- Цикл **Repeat** сначала выполняет тело цикла, а лишь затем проверяет условие



Для гибкого управления циклическими операторами используются процедуры:

- **Break** — выход из цикла;
- **Continue** — завершение очередного прохода цикла;

Примеры: Найти минимальное число i , для которого сумма чисел от 1 до i больше 100. Как только s (сумма чисел от 1 до i) становится больше 100 срабатывает оператор **break** и происходит выход из цикла.

```
s:=0;
for I := 1 to 100 do begin
  if s>100 then break;
  s := s + i;
end;
Writeln ('минимальное число i, такое, что ( 1+2+..+i )>100 равно ',i);
```

С клавиатуры вводятся 10 чисел и в цикле считается сумма только положительных. Если число отрицательное, то выполняется оператор **continue**, который начинает следующий проход цикла.

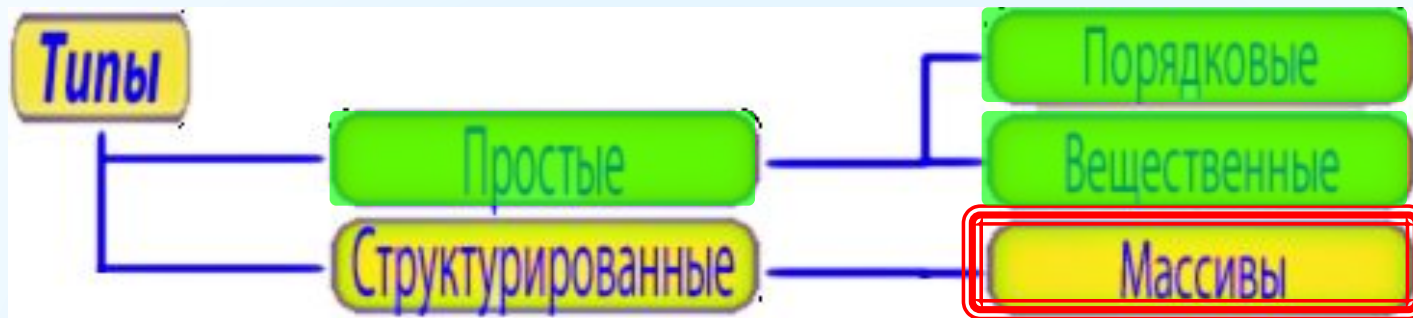
```
s:=0;
for I := 1 to 10 do begin
  Readln (k);
  if k<0 then Continue;
  s := s + k;
end;
Writeln ('Сумма положительных чисел равна ',s);
```


МАССИВЫ



Образовательный центр

ШКОЛЬНЫЙ УНИВЕРСИТЕТ



Простые типы: одна переменная – одно значение

Структурированные типы: одна переменная – несколько значений

Массив – это фиксированное количество значений одного типа.

Доступ к конкретному значению осуществляется через индекс.

Структура массива:



Доступ к массиву:

<Имя массива> [*<Индекс>*] A[1] := 7;

Массив объявляется в разделе **Var**:

```
{Имя} : Array [ {нач_зн} .. {кон_зн} ] of {тип} ;
```

Примеры объявления массивов:

```
Var A : Array [1..4] of String;  
    B : Array [0..662] of Real;  
    C : Array [1..10] of Integer;
```

Примеры заполнения массивов значениями:

```
A[1] := 'Вася' ; A[2] := 'Петя' ;  
A[3] := 'Маша' ; A[4] := 'Олеся' ;  
Write (A[3]);  
  
For f:=1 to 10 do C[f] := f*2;  
For f:=1 to 10 do WriteLn ( C[f] );
```

Напишите программу, которая запрашивает у пользователя 7 целых чисел и заносит их в массив.

В массиве находится максимальный элемент и отображается на экране. Например:

```
Введите 1 число: 4
Введите 2 число: 8
Введите 3 число: 9
Введите 4 число: 2
Введите 5 число: 4
Введите 6 число: 5
Введите 7 число: 0
Максимальное число: 9
```

Используемый материал:

Объявления массива:

```
<Имя> : Array [<нач_зн> ... <кон_зн> ] of <тип> ;
```

Доступ к массиву:

```
<Имя массива> [ <Индекс> ]
```



Для генерации в программе случайных чисел, используют следующие операторы:

Randomize – инициализация ГСЧ. Объявляется только в самом начале программы;

Random – генерирует случайное число от 0 до 1 (вещественный тип);

Random (N) – генерирует случайное число от 0 до N-1 (целый тип);

```

Program My_program;
Uses CRT;

Var n,f:Integer;
    A : Array [1..7] of Integer;

Begin
  ClrScr;
  Randomize;

  Write ('Случайные числа от 0 до ');
  Read (n);

  For f:=1 to 7 do begin
    A[f] := random (n+1);
    Write (A[f], ' ')
  end;

  ReadKey;
End.
    
```

 Output

```

Случайные числа от 0 до 10
7 2 7 3 0 7 6 _
    
```

```
{Имя} : Array [ {нач_зн} .. {кон_зн} ,
                {нач_зн} .. {кон_зн}
                ,
                {и т.д.}
                ] of {тип} ;
```

Список интервалов для каждой размерности массива

Пример объявления двумерного массива (матрицы, таблицы) на 4 строки и 6 столбцов:

```
Var A : Array [1..4,1..6] of Integer;
```

Пример заполнения массива:

```
For i:=1 to 4 do
  For j:=1 to 6 do
    A[i,j] := i+j;
```

$$A_{i,j} = \begin{bmatrix} 2 & 3 & 4 & 5 & 6 & 7 \\ 3 & 4 & 5 & 6 & 7 & 8 \\ 4 & 5 & 6 & 7 & 8 & 9 \\ 5 & 6 & 7 & 8 & 9 & 10 \end{bmatrix}$$

Напишите программу, которая заполняет двумерный массив случайными числами от -10 до 20 и выводит суммы элементов расположенных по диагонали таблицы.

Используемый материал:

Инициализация ГСЧ: **Randomize**

Случайное число от 0 до N-1: **Random (N)**



РАБОТА С ФАЙЛАМИ



Образовательный центр

ШКОЛЬНЫЙ УНИВЕРСИТЕТ



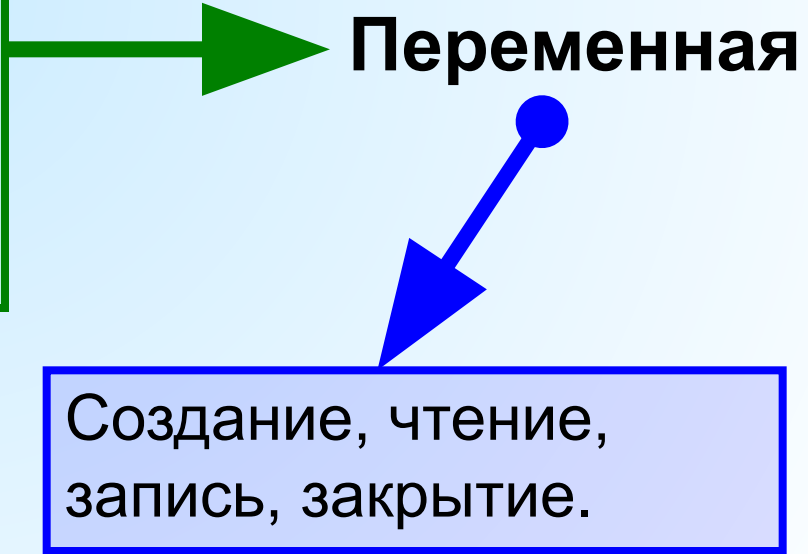
Файл:

Начало файла

Указатель

```
) a@$kuΔ | "Hd9v*9 (L*M=BYGRMxduB
8e*BOtCDrTVzHGJ1aBD>@Б\ (r8sE}
:wcJvAmRa'v/.Wai;$`SWI=y2]suB
?Hq>vF(LmBcV^Adz4P4.6b]o{QkB8
cu< 8Z':M^1;:8ANwak; ,b2-4...u5
2]suB?Hq>vF(LmBcAdz4wcP]o{QkB
8c8df]e"v,su>+),VAEFБjFV,W$Q-
y0G1GjN$-eБ|sqZ0`QnB%\БD%y
```

■ - признак конца строки
■ - признак конца файла



Последовательность действий, при работе с файлами:

1. Объявление файловой переменной (ФП);
2. Ассоциация ФП с файлом;
3. Открытие файла для чтения/записи;
4. Операции с файлом;
5. Закрытие файла (так же разрывается связь между файлом и ФП);

1. Объявление ФП

<Имя> : File of *<тип>* ;
<Имя> : Text;

```
Var f: Text;  
    h: File of Integer;
```

2. Ассоциация ФП с файлом

Assign (*<ФП>*, *<имя файла>*);

```
Assign (f, 'c:\my\Data.ghm')
```

3. Открытие файла для чтения/записи

Reset (<ФП>); - открывает файл для чтения

Rewrite (<ФП>); - открывает файл для записи

4. Операции с файлом

Read (<ФП>, <П1>, <П2>, ...); - считывает в переменные <П1>, <П2> и т.д. по одному элементу с позиции указателя.

Write (<ФП>, <П1>, <П2>, ...); - записывает в файл значения переменных <П1>, <П2> и т.д. по одному элементу с позиции указателя.

EoLn (<ФП>); - возвращает **True**, если достигнут конец строки.

EoF (<ФП>); - возвращает **True**, если достигнут конец файла.

5. Заккрытие файла

Close (<ФП>);

```
Program My_program;  
Uses CRT;  
Var i,j,i1,j1:Integer;  
    A : Array [1..100,1..100] of Integer;  
    f:Text;  
Begin  
  ClrScr;  
  
  Assign (f,'c:\Matrix.txt');  
  Reset (f);  
  j:=1;  
  Repeat  
    i:=1;  
    Repeat  
      Read (f,A[i,j]);  
      i := i+1;  
    Until EoLn(f);  
    j := j+1;  
  Until Eof(f);  
  Close(f);
```

Упростите программу, если известно, что матрица размером 5x5 (используйте цикл **For**)



```
Assign (f,'c:\Matrix1.txt');  
Rewrite (f);  
i1:=i; j1:=j;  
For i:=1 to i1-1 do begin  
  for j:=1 to j1-1 do  
    write (f,A[i,j],' ');  
  WriteLn(f);  
end;  
Close(f);  
End.
```

Напишите программу «ДОМ-3». В первом файле содержатся имена участников в именительном падеже. Во втором, те же самые имена, но в винительном. В третьем – список выражения чувств или какое-то действие (любит, не любит, целует, хочет укунить, обожает, уважает, ненавидит, хочет видеть, обнимает).

Программа должна случайно брать информацию из этих файлов и создавать новый по следующему принципу:

<имя в им. пад> <чувство/действие> <имя в вин. пад>

Ольга любит Сергей
Олег хочет видеть Романа
Катя уважает Настю
И т.д.

Используемый материал:

Название операторов: **Assign, Rewrite, Reset, Write, Read, Close.**



TECHNO

ГРАФИКА



Образовательный центр

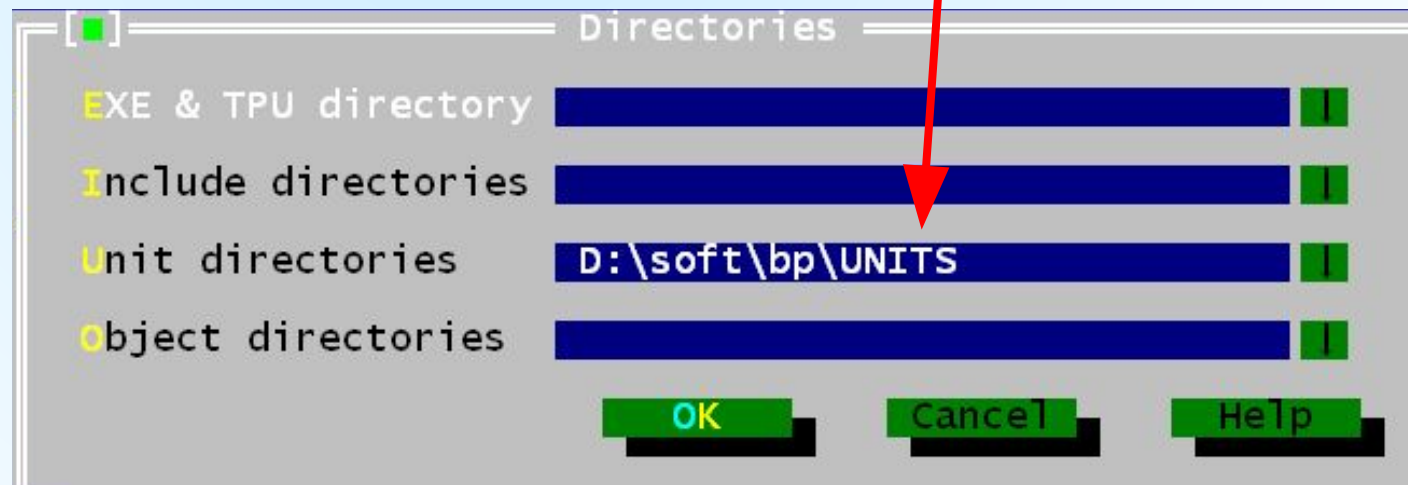
ШКОЛЬНЫЙ УНИВЕРСИТЕТ

Для работы с графикой в Pascal необходимы два файла – модуль **graph.tpu** и драйвер графического адаптера **egavga.bgi**. Первый находится в каталоге **\UNITS**, а второй – в **\BGI**.

! Драйвер **egavga.bgi. Необходим при работе exe файла !**

Чтобы рисовать, надо:

1. подключить модуль **GRAPH** (в разделе **Uses**);
2. инициализировать графику (**InitGraph**);
3. что-нибудь нарисовать;
4. закрыть графический режим (**CloseGraph**)

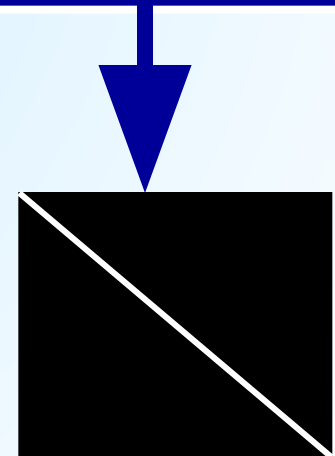


Инициализация
графического режима

Рисование линии.
Перо переходит из
точки (0,0) в точку
(639, 479).

```
PROGRAM Primer_1;  
Uses Graph, crt;  
Var Gd,Gm: Integer;  
BEGIN  
  Gd:=0;  
  InitGraph (Gd,Gm, 'd:\BP\bgi');  
  Line (0,0,639,479);  
  ReadKey;  
  CloseGraph;  
END.
```

Заккрытие
графического режима



1. Изобразить в центре экрана прямоугольный треугольник
2. Изобразить в центре экрана прямоугольник с сторонами в двое меньшими, чем соответствующие стороны экрана

Используемый материал:

Графический модуль: **Graph**

Инициализация графики: **InitGraph**

Закрытие графического режима: **CloseGraph**;



SetColor(Color: word);

Устанавливает цвет пера

GetColor: word;

Возвращает цвет пера

SetBkColor(color: word);

Устанавливает цвет фона.

GetBkColor: word;

Возвращает цвет фона.

```

Program My_Program;
Uses CRT,Graph;
Var g,b:Integer;
Begin
  ClrScr;
  Randomize;
  g:=0; InitGraph(g,b, 'd:\BP\BGI');
  SetBkcolor (Red);
  For g:=0 to 10 do Begin
    Line (320, 240, Random(640), Random(480));
    setcolor (Random(15));
  End;
  ReadKey;
  CloseGraph;
End.
    
```

Цвета

Black – чёрный

Blue – синий

Green – зелёный

Сyan – циановый

Red – красный

Magenta – сиреневый

Brown – коричневый

LightGray – светло-серый

DarkGray – тёмно-серый

LightBlue – голубой

LightGreen – светло-зелёный

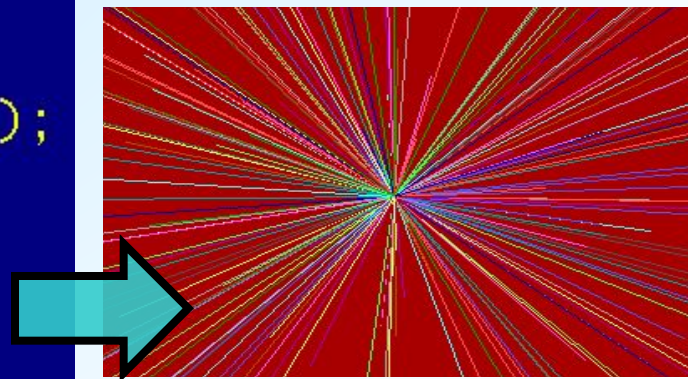
LightCyan – светло-циановый

LightRed – розовый

LightMagenta – светло-сиреневый

Yellow – жёлтый

White – белый



Line (x, y);

Чертит линию от текущей координаты пера до координат (x,y).

MoveTo (x, y);

Устанавливает перо в координаты (x,y).

PutPixel (x, y, Color);

Рисует точку с координатами (x,y) цветом Color.

GetPixel (x, y): word;

Возвращает цвет точки с координатами (x,y).

Rectangle (x1, y1, x2, y2);

Строит контур прямоугольника.

x1, y1



x2, y2

Circle (x, y, r);

Рисует окружность с центром в (x,y) и радиусом r.

SetLineStyle (Ln, 0, T)

Изменяет параметры контуров. Ln - стиль линии (0..3):

T - толщина линии: 1 = нормальная; 3 = толстая.

FillEllipse (x, y, Xr, Yr);

Рисует закрашенный эллипс с центром в (x,y) и радиусами Xr и Yr.

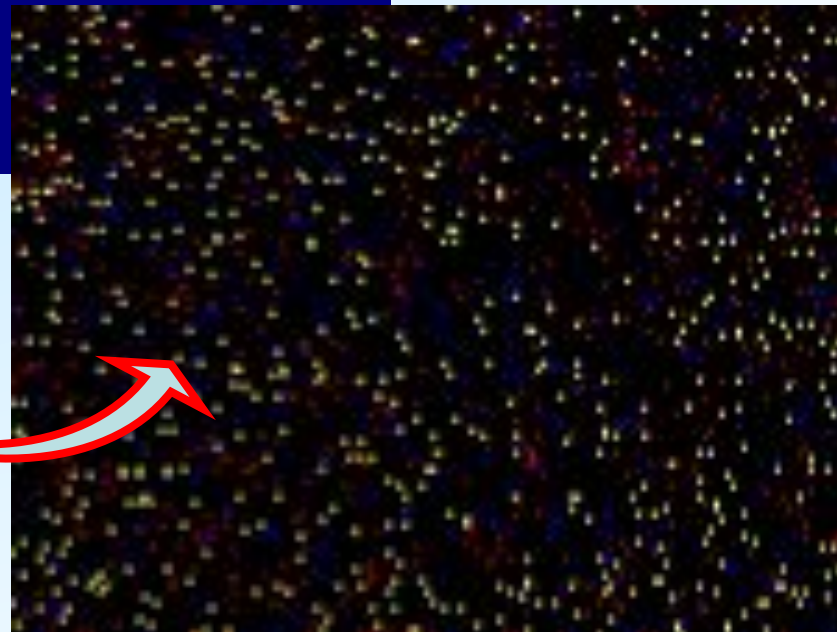
SetFillStyle (Type, Color);

Устанавливает тип (0..11) и цвет штриховки замкнутых фигур

ClearDevice;

Очищает графический экран закрашивая его в текущий цвет фона

```
Program My_Program;  
  Uses CRT,Graph;  
  Var color ,RColor ,g ,b ,maxX ,MaxY :Integer;  
Begin  
  ClrScr;  
  Randomize;  
  MaxX := 639;  
  MaxY := 479;  
  g:=0; InitGraph(g,b,'d:\BP\BGI');  
  For g:=0 to 10000 do Begin  
    PutPixel( Random(MaxX) , Random(MaxY) , Red);  
    PutPixel( Random(MaxX) , Random(MaxY) , Yellow);  
    PutPixel( Random(MaxX) , Random(MaxY) , Blue);  
  End;  
  ReadKey;  
  CloseGraph;  
End.
```



1. Напишите программу, рисующее на экране монитора домик;
2. Напишите программу, рисующая на экране монитора звёздное небо со звёздами случайного радиуса (от 1 до 5 пикс.) и случайным расположением. Цвет тоже случайный (белый, светло-серый, тёмно-серый);

Используемый материал:

SetColor (Color: word); - Устанавливает цвет рисования

SetBkColor (color: word); - Устанавливает текущий цвет фона.

Line (x, y); - Чертит линию от текущей координаты пера до координат (x,y).

MoveTo (x, y); - Устанавливает перо в координаты (x,y).

PutPixel (x, y, Color); - Рисует точку с координатами (x,y) цветом Color.

Rectangle (x1, y1, x2, y2); - Строит контур прямоугольника из линий текущего цвета.

Circle (x, y, r); - Рисует окружность с центром в (x,y) и радиусом r.

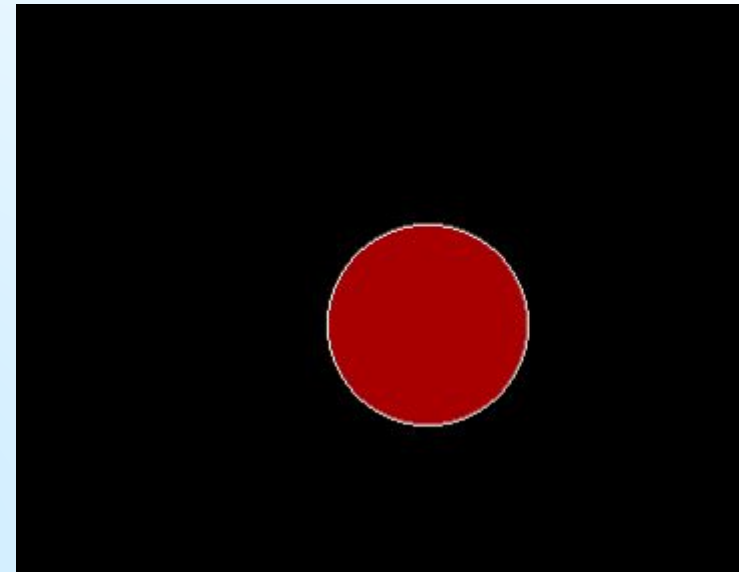
FillEllipse (x, y, Xr, Yr); - Рисует закрашенный эллипс с центром в (x,y) и радиусами Xr и Yr.



```

Program My_Program;
Uses CRT,Graph;
Var Vx,Vy,X,Y,g,b,maxX,MaxY:Integer;
Begin
ClrScr;
g:=0; InitGraph(g,b,'d:\BP\BGI');
MaxX := 639; MaxY := 479;
X := 100; Vx := 3;
Y := 100; Vy := 6;
Repeat
  {РИСОВАНИЕ}
  SetFillstyle (1,Red);
  SetColor (white);
  FillEllipse (X, Y, 50, 50);
  Delay (6000);
  {СТИРАНИЕ}
  SetFillstyle (1,Black);
  SetColor (Black);
  FillEllipse (X, Y, 50, 50);
  {ИЗМЕНЕНИЕ координат}
  X := X+Vx;
  Y := Y+Vy;

```

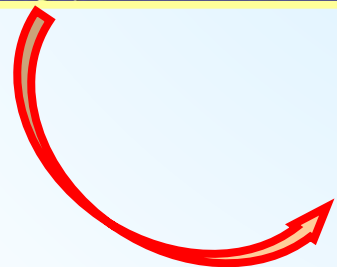


Пауза в мСек.

```

  {Проверка границ}
  If (Y+50>MaxY) or (y<50) Then vy := vy*-1;
  If (X+50>MaxX) or (x<50) Then vx := vx*-1;
  Until KeyPressed;
  CloseGraph;
End.

```



ПОДПРОГРАММЫ

DESIGN IN FUTURE



Образовательный центр

ТУСУР

ШКОЛЬНЫЙ УНИВЕРСИТЕТ

Подпрограммы позволяют выделять повторяющуюся часть кода в отдельные фрагменты и вставлять их в нужные места программы.

Процедуры

Функции

Функция, в отличие от **процедуры** возвращает результат вызова.

Подпрограммы:

- пишутся между ключевыми словами **Begin End**;
- идентифицируются по именам, которые пишутся по правилам задания идентификаторов;
- могут иметь входные и выходные параметры;
- полностью повторяют структуру основной программы.

Пример системных подпрограмм:

```
Write ("Ok"); // процедура с одним параметром
ClrScr; // процедура без параметров
Length (S); // функция с одним параметром
Random; // функция без параметров
```

Program My;

Подпрограмма 1
[Код подпрог-мы
1]

Подпрограмма 2
[Код подпрог-мы
2]

Begin

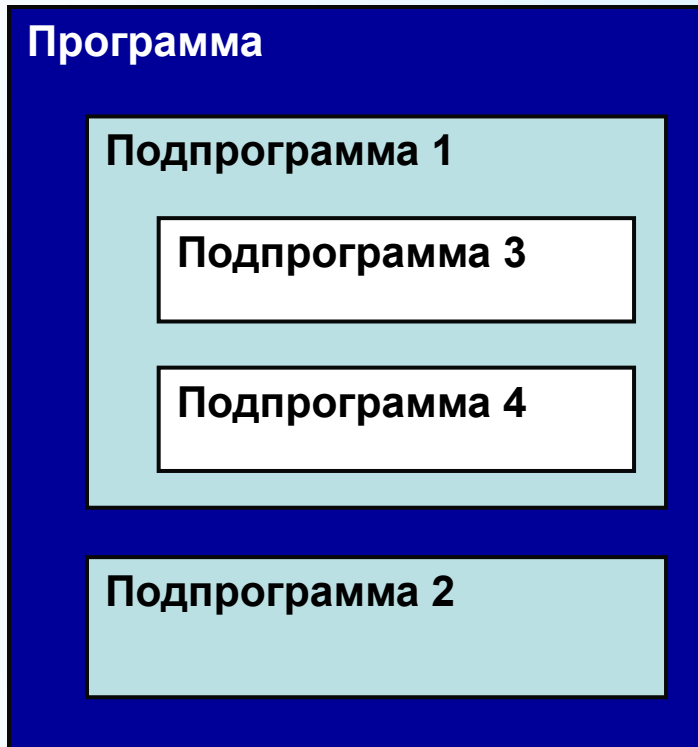
Подпрограмма 2

Подпрограмма 1

Подпрограмма 2

End.

Вложенность подпрограмм



Область доступности переменных

- Переменные, описанные в некоторой ППР, известны в пределах её тела, **ВКЛЮЧАЯ** и все вложенные ППР (переменная описанная в ППР1 будет доступна и в ППР3,4).
- Имена переменных, описанных в ППР, могут совпадать с именами переменных из других ППР (в каждой из ППР может быть объявлена одинаковая переменная).
- Имя переменной, описанное в ППР, экранирует одноимённые переменные из ППР, объемлющих данный (если в ППР1 и ППР3 объявлены одинаковые переменные, то в каждой ППР эти переменные будут уникальны).

Переменные программы называются **глобальные**,
А переменные **подпрограмм** – локальные.

Синтаксис записи:

```
Procedure {имя процедуры};  
{Раздел переменных, констант, типов, меток, модулей}  
Begin  
{Тело процедуры};  
End;
```

Пример использования процедуры без параметров:

```
Program My_Program;  
Uses CRT;  
  
Procedure Firm;  
Begin  
  TextColor(Red);  
  TextBackground(Yellow);  
  Write('Apom');  
  TextColor(LightGray);  
  TextBackground(Black);  
end;  
  
Begin  
  Write('Фирма '); Firm; WriteLN(' 10 лет продает яблоки. ');  
  Write('Покупайте яблоки у фирмы '); Firm; Write('!');  
  
End.
```

Пример экранирования одноимённых переменных:

```
Program My_program;
```

```
Uses Crt;
```

```
Var A:String;
```

```
// Описание процедуры
```

```
Procedure My_Procedure;
```

```
Var A:String;
```

```
Begin
```

```
  A := 'Подпрограмма';  
  Writeln(A);
```

```
End;
```

```
Begin
```

```
  A := 'Тело';  
  Writeln(A);
```

```
  My_Procedure; // Вызов процедуры
```

```
  Writeln(A);
```

```
End.
```

Тело

Подпрограмма

Тело

Тело

Подпрограмма

Подпрограмма

Для того, чтобы подпрограммы были более универсальными, применяются специальные механизмы обмена данными между программой и её подпрограммами.

Системные процедуры с параметрами:

```
SetColor (Red);           { Процедура с одним параметром }
Rectangle (0, 0, 639, 479); { Процедура с несколькими параметрами }
InitGraph (Gd,Gm, 'd:\BP\bgi'); { Процедура с несколькими разнотипными }
                                { параметрами }
```

Синтаксис записи процедуры с параметрами

```
Procedure {Имя} ( {Область входных переменных}
                 Var
                 {Область выходных переменных (для каждой переменной свой Var)} );
```

Структура областей:

```
Переменная_1, Переменная_2, ... : Тип;
.....
Переменная_N-1, Переменная_N, ... : Тип;
```

Любая из областей может отсутствовать

Использование входных параметров

Процедура рисования окаймляющего экран прямоугольника,
заданного цвета

Без параметров

```
Procedure Cir;  
Begin  
  SetColor (i);  
  SetBkColor (b);  
  Rectangle (0, 0, 639, 479);  
End;
```

Вызов процедуры в программе:

```
i:=6; b:=12;  
Cir;
```

С параметрами

```
Procedure Cir (i, b:Integer);  
Begin  
  SetColor (i);  
  SetBkColor (b);  
  Rectangle (0, 0, 639, 479);  
End;
```

Вызов процедуры в программе:

```
Cir(6, 12);
```

ПРОЦЕДУРЫ С ПАРАМЕТРАМИ

Использование выходных параметров

Процедура преобразования угла из градусной меры в радианную.

```
PROGRAM EX_26_3;
VAR x, a: real;

PROCEDURE Rad(alfa: real;
              var betta: real); {выходная переменная}
BEGIN
    Betta := pi*alfa/180;
END;

BEGIN
    Write('Введите угол в градусах: ');
    Readln(x);
    Rad(x, a);    {Вызов процедуры}
    Writeln('Ему равен угол в радианах = ', a:6:4);
END.
```

1. Напишите процедуру, рисующую треугольник, в следующем формате:
Triangle (x1,y1, x2,y2, x3,y3, Color)
2. Напишите процедуру вычисления площади прямоугольника в следующем формате:
SRect (a, b, S)
где, S – выходной параметр процедуры.

Используемый материал:

```
Procedure {Имя} ( {Область входных переменных}  
    Var  
    {Область выходных переменных} );
```

Структура областей:

```
Переменная_1, Переменная_2, ... : Тип;
```

.....

```
Переменная_N-1, Переменная_N, ... : Тип;
```



Синтаксис записи функции

```
Function ( {Область входных переменных}
          Var
          {Область выходных переменных} ) : {Тип} ;
```

Оформление процедуры

```
Procedure S(a,b:real; var s:real);
Begin
    s := a+b;
End;
```

Вызов процедуры

```
S(10, 20, A);
Write (A);
```

Оформление функции

```
Function Sum (a,b: real): real;
Begin
    Sum := a+b;
End;
```

Вызов функции

```
A := S(10, 20);
WriteLN (A);
WriteLN ( S(20, 30) );
```

Оформлять некоторую подпрограмму как функцию целесообразно только в том случае, если ожидается некоторый результат её работы.

Если подпрограмма ориентирована только на выполнение некоторой последовательности действий (вывод на экран, рисование и т.д.), лучше оформить её как процедуру.

Напишите функцию вычисления площади прямоугольника в следующем формате:

`SRect (a, b)`

Используемый материал:

```
Function {Имя} ( {Область входных переменных}  
    Var  
    {Область выходных переменных} ): {Тип} ;
```

Структура областей:

Переменная_1, Переменная_2, ... : Тип;

.....

Переменная_N-1, Переменная_N, ... : Тип;



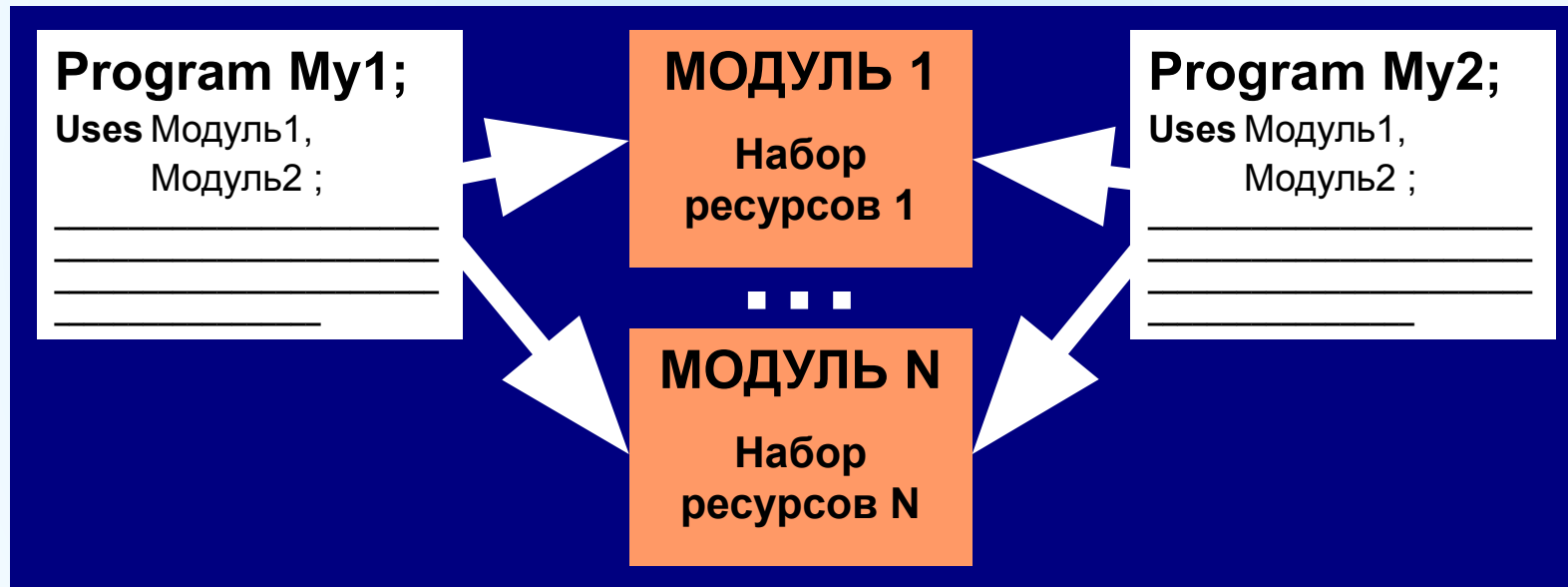
МОДУЛИ



Образовательный центр

ШКОЛЬНЫЙ УНИВЕРСИТЕТ

Модуль (UNIT) - самостоятельная программная единица, ресурсы (процедуры и функции) которой могут быть использованы другими программами.



Модули подключаются к программе через раздел **USES**

Модуль состоит из следующих частей:

1. Заголовок модуля.
2. Интерфейсная часть. (описания видимых объектов)
3. Реализационная часть. (описания скрытых объектов)
4. Инициализационная часть (не обязательна).

```
Unit {Имя модуля};
```

```
Interface
```

```
{ Раздел глобальных переменных, констант, типов }
```

```
{ модулей }
```

```
{ Список заголовков процедур и функций }
```

```
Implementation
```

```
{ Раздел локальных переменных, констант, типов }
```

```
{ Реализация процедур и функций }
```

```
Begin
```

```
{Часть инициализации }
```

```
End.
```

! Имя заголовка модуля должно совпадать с именем файла модуля !

Пример модуля:

```
Unit My_Unit;  
Interface  
  
Var ms: Array [1..100, 1..10] of Real; { Глобальный массив }  
Function Cub(x:integer):Integer;      { Функция Cub = x^3 }  
Function ext4(x:integer):Integer;     { Функция ext4 = x^4 }
```

Implementation

```
Function Cub(x:integer):Integer; { Реализация функции Cub }  
  Begin  
    Cub := x*x*x;  
  End;  
  
Function Ext4(x:integer):Integer; { Реализация функции ext4 }  
  
  Begin  
    Ext4 := x*x*x*x;  
  End;  
  
End.
```

Напишите модуль одной функцией и одной процедурой:

```
{ Функция вычисление  $X1=1/x$  }
```

```
X1 (a:real) :real;
```

```
{ Процедура печати на экране слова S, в позиции X, Y }
```

```
WriteXY (S:String; X,Y:Integer);
```

Используемый материал (структура модуля):

```
Unit {Имя модуля};
```

```
Interface
```

```
{ Раздел глобальных переменных, констант, типов}
```

```
{ модулей }
```

```
{ Список заголовков процедур и функций }
```

```
Implementation
```

```
{ Раздел локальных переменных, констант, типов}
```

```
{ Реализация процедур и функций }
```

```
Begin
```

```
{Часть инициализации }
```

```
End.
```

