Объектно-ориентированное программирование (ООП)

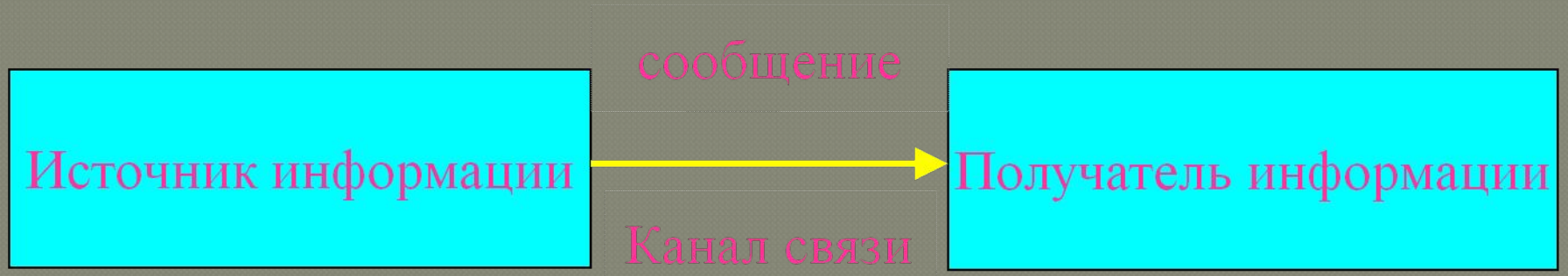
По мере усложнения задач, которые требовалось решать программистам, усложнялись и программы. Для увеличения производительности работы программиста, уменьшения ошибок и улучшения удобочитаемости программы.

Классы и объекты.
С точки зрения программирования класс можно рассматривать как набор данных (полей, атрибутов, членов класса) и функций для работы с ними (методов).

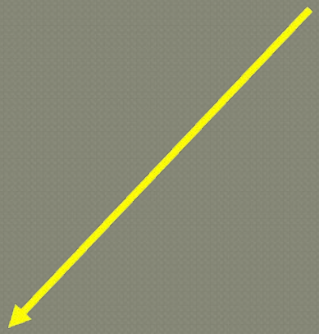
Описав класс, мы можем создать его экземпляр – объект. Объект – это уже конкретный представитель класса.

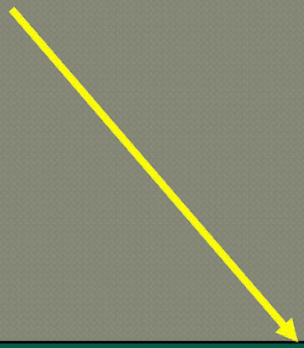
Информация – это знания или сведения о ком-либо или о чем-либо, которые можно собирать, хранить, передавать, обрабатывать, использовать.

Объекты информации



Для измерения информации вводятся два параметра





Объем информации (объемный подход)

Количество информации (вероятностный подход)

Объемный подход

Если количество информации, содержащейся в сообщении из одного символа, принять за единицу, то объем информации (данных) V в любом другом сообщении будет равен количеству символов (разрядов) в этом сообщении. В памяти компьютера объем информации записывается двоичными знаками и равен количеству требуемых для этой записи двоичных кодов.

Единицы измерения информации

Наименьшая единица измерения - 1бит

1 байт = 8 бит

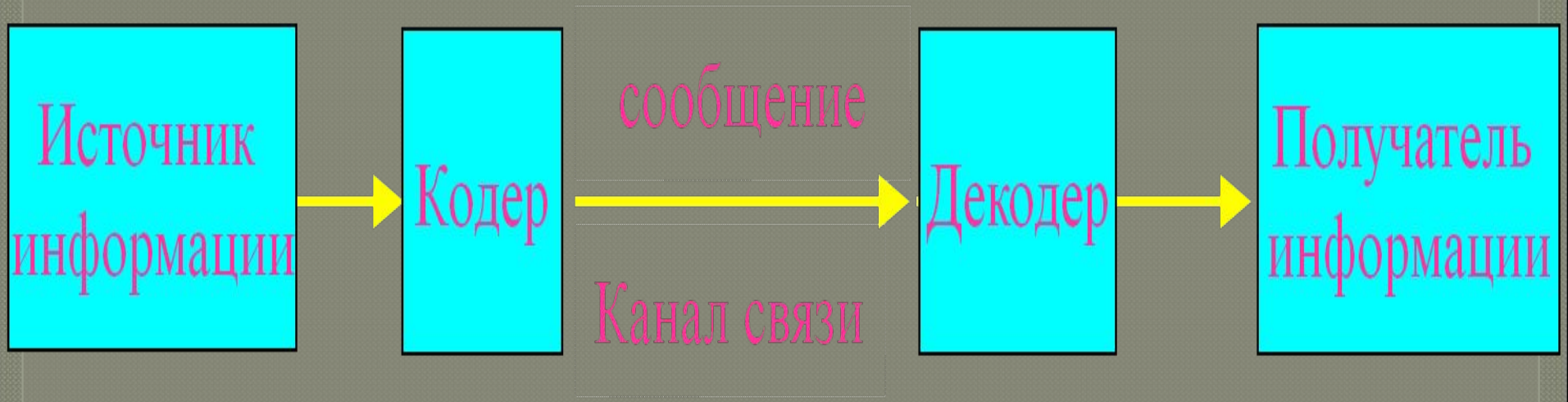
1 Кбайт = 1024 байт

1 Мбайт = 1024 Кбайт = 1 048 576 байт;

1 Гбайт = 1024 Мбайт = 1 073 741 824 байт;

1 Тбайт = 1024 Гбайт = 1 099 511 627 776 байт.

Кодирование информации. Основные понятия

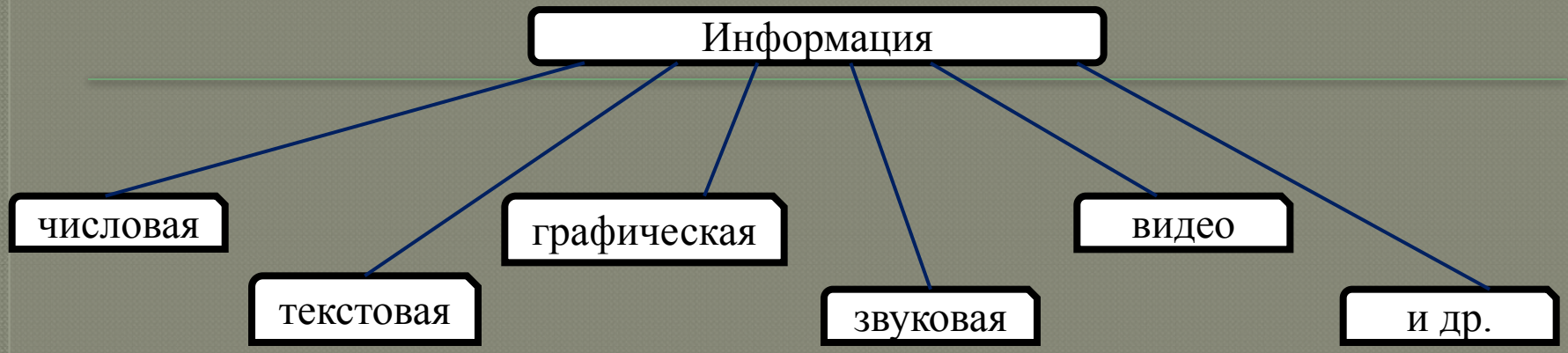


Источник представляет сообщение в алфавите, который называется ***первичным***, далее это сообщение попадает в устройство, преобразующее и представляющее его во ***вторичном алфавите***.

- ***Код*** – правило, описывающее соответствие знаков (или их сочетаний) первичного алфавита знаком (их сочетаниями) вторичного алфавита.
- ***Кодирование*** – перевод информации, представленной сообщением в первичном алфавите, в последовательность кодов.
- ***Декодирование*** – операция обратная кодированию.
- ***Кодер*** – устройство, обеспечивающее выполнение операции кодирования.
- ***Декодер*** – устройство, производящее декодирование.

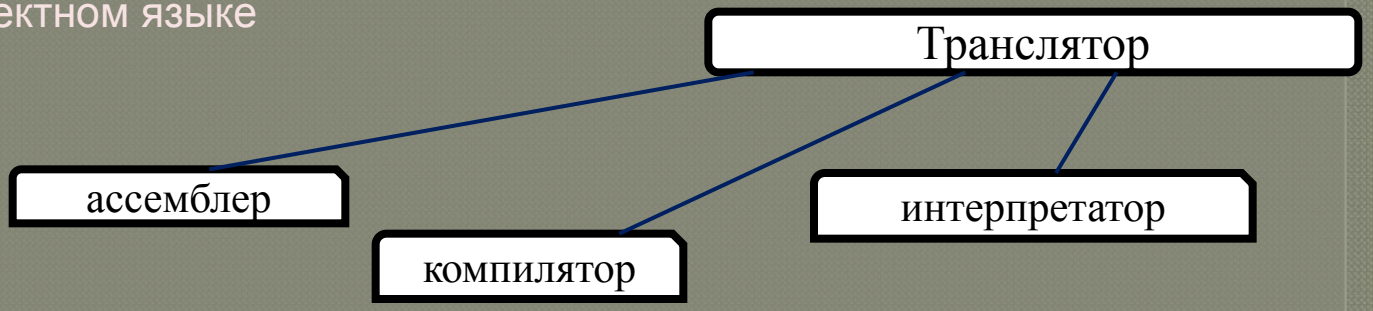
Операции кодирования и декодирования называются ***обратимыми,*** если их последовательное применение обеспечит возврат к исходной информации без каких-либо ее потерь.

Информация передается в виде сообщений. Информация может быть по своей физической природе



Любая информация, обрабатываемая в ЭВМ, должна быть представлена двоичными цифрами {0,1}, т.е. должна быть закодирована комбинацией этих цифр. Различные виды информации (числа, тексты, графика, звук) имеют свой правила кодирования. Коды отдельных значений, относящиеся к различным видам информации, могут совпадать. Поэтому расшифровка кодированных данных осуществляется по контексту при выполнении команд программы.

Транслятор - обслуживающая программа, преобразующая исходную программу, предоставленную на входном языке программирования, в рабочую программу, представленную на объектном языке



Язык, на котором представлена входная программа, называется исходным языком, а сама программа — исходным кодом. Выходной язык называется целевым языком, а выходная (результирующая) программа — объектным кодом.

Компилятор - это обслуживающая программа, выполняющая трансляцию на машинный язык программы, записанной на исходном языке программирования. Результат компилятора – это exe файл. И может быть запущен в рамках ОС

Интерпретатор - программа или устройство, осуществляющее пооператорную трансляцию и выполнение исходной программы.

Транслятор

- генерирует выходную программу (ее часто называют объектной) на языке машинных команд;

- анализирует транслируемую программу, в частности определяет, содержит ли она синтаксические ошибки;

- распределяет память для объектной программы.

Процесс поиска и устранение ошибок называется **отладкой.**

Ошибки

Синтаксические ошибки – это ошибки в записи конструкций языка программирования

Логические ошибки это ошибки, связанные с неправильным содержанием действий и использованием недопустимых значений величин

Семантические ошибки это нарушение логики программы, приводящее к неверному результату.

Машинно-ориентированные (низкого уровня) - каждая команда соответствует одной (нескольким) командам процессора (ассемблер)

Ассемблер - системная обслуживающая программа, которая преобразует символические конструкции в команды машинного языка. Это языки, в которых вместо численного обозначения команд и областей памяти используются буквенные. После ассемблеров наступил рассвет языков так называемого высокого уровня.

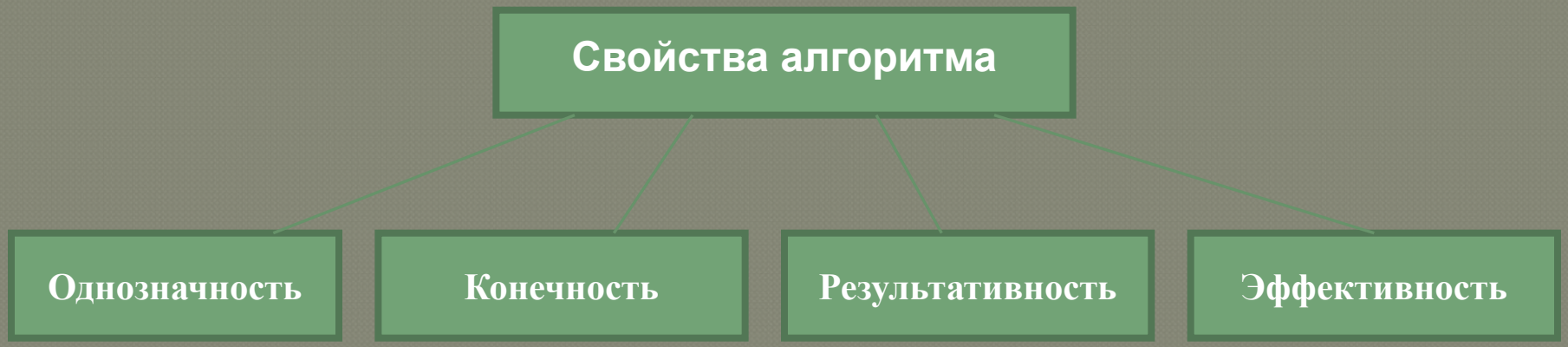
Языки высокого уровня – приближены к естественному языку, легче воспринимаются человеком, не зависят от операционной системы

Для того чтобы решить задачу с помощью ПК, необходимо пройти определенные этапы ее решения.

- *Формализация задачи.*
- *Создание математической модели.*
- *Детальное описание алгоритма (текстовое, псевдокод, блок-схема).*
- *Реализация на языке программирования.*
- *Отладка программы.*
- *Тестирование программы.*
- *Анализ результатов работы.*

АЛГОРИТМ. СПОСОБЫ ЗАПИСИ АЛГОРИТМА.

Алгоритм - это конечная последовательность однозначных предписаний, исполнение которых позволяет с помощью конечного числа шагов получить решение задачи, однозначно определяемое исходными данными



Способы записи алгоритма

✔ Словесно-формульное описание (на естественном языке с использованием математических формул).
✔ Графическое описание в виде блок-схемы (набор связанных между собой геометрических фигур).
✔ Описание на каком-либо языке программирования (программа).

Свойства алгоритма (вычислительного)

- дискретность: состоит из отдельных шагов (команд)
- понятность: должен включать только команды, известные исполнителю
- определенность: при одинаковых исходных данных всегда выдает один и тот же результат
- конечность: заканчивается за конечное число шагов
- массовость: может применяться многократно при различных исходных данных

Программа – это алгоритм, записанный на каком-либо языке программирования / набор команд для компьютера

Команда – это описание действий, которые должен выполнить компьютер.

Распространены две методики (стратегии) разработки программ, относящиеся к структурному программированию:
– программирование «сверху вниз»;
В данном случае программа конструируется иерархически - сверху вниз: от главной программы к подпрограммам самого нижнего уровня
– программирование «снизу вверх».
методика разработки программ, начинающаяся с разработки подпрограмм (процедур, функций), в то время когда проработка общей схемы не закончилась.

Словесно-формульное описание

Запись алгоритма на псевдокоде называется структурным планом.

основные ключевые слова	дополнительные ключевые слова
алг (алгоритм) **рез** (результат) **нач** (начало) **кон** (конец) **арг** (аргумент) **знач** (значение) **тип** **вещ** (вещественный) **цел** (целый) **лит** (литерный) **таб** (таблица) **сим** (символьный) **не** **то** **если** **и** **все** **выбор** **если** **иначе** **от** **до** **шаг** **для** **при** **да** **нет** **пока** :=(оператор присваивания) **нц** (начало цикла) **кц** (конец цикла)	**запись** **истина** **лог** (логический) **ложь** **массив** **множество** **функция** **дано** **надо** **ввод** **вывод** **утв** (утверждение)

Схема алгоритма – это графическое представление метода решения задачи, в котором используются символы для отображения операций, данных, потока, оборудования и т.д.

При всем разнообразии структур алгоритмов можно выделить три типовых структуры

Линейный - алгоритм, в котором все предписания (шаги) выполняются так, как записаны, без изменения порядка следования, строго друг за другом

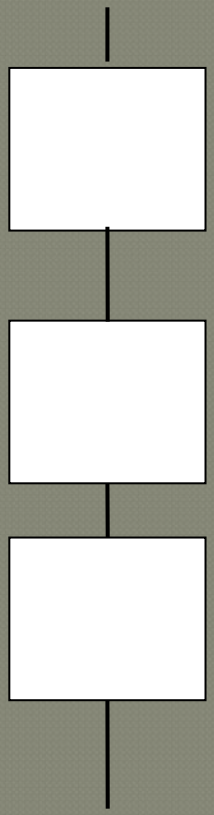
Разветвляющийся - алгоритм, в котором выполнение того или иного действия (шага) зависит от выполнения или не выполнения какого-либо условия

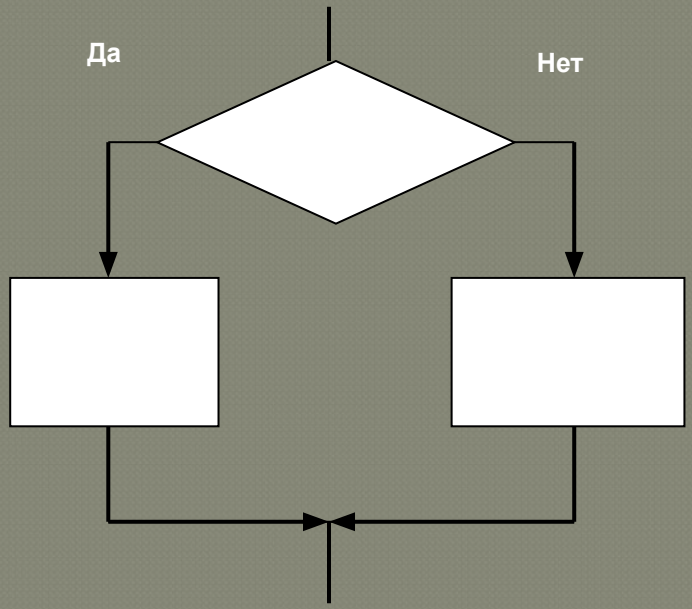
Циклический - алгоритм, в котором некоторая последовательность действий повторяется несколько раз

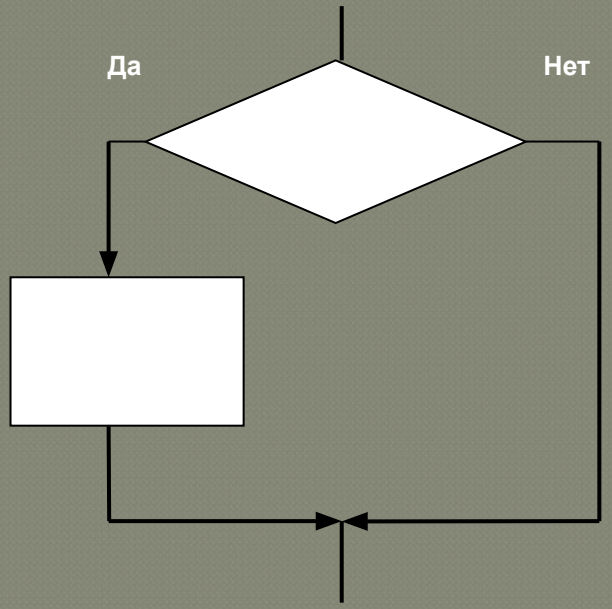
Типовые структуры алгоритмов

Линейный

Разветвляющийся



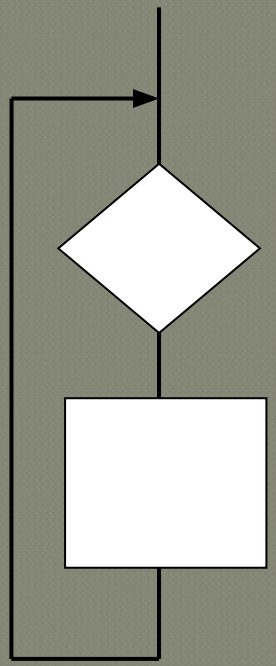


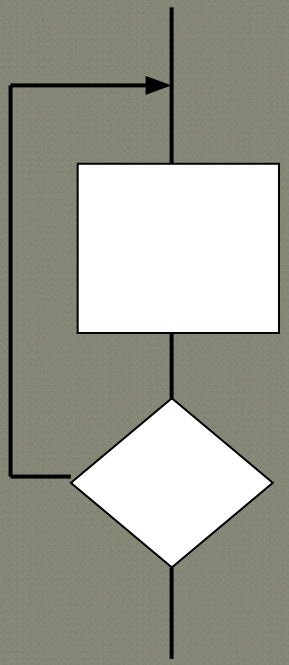


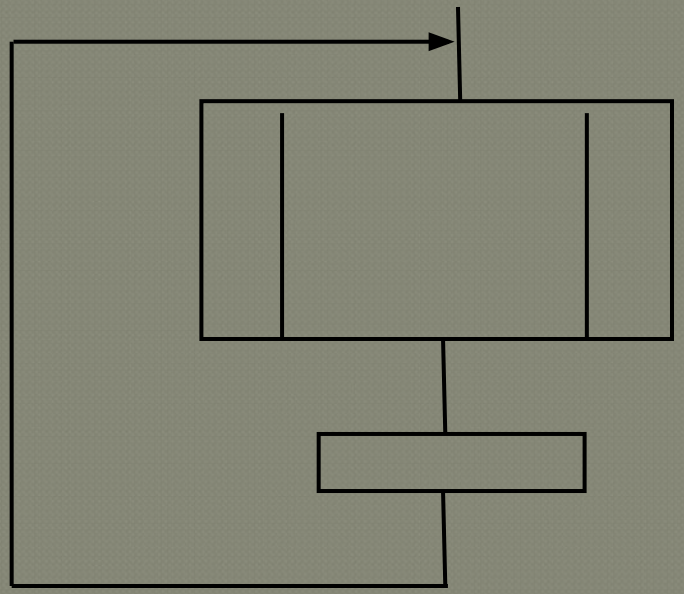
а) - следование;

б, в) – ветвление (полное и неполное).

Циклический







а) – цикл с предусловием; б) – цикл с постусловием в) цикл с параметром

Программирование на языке Pascal ABC

Язык Паскаль был создан Никлаусом Виртом в **1968—1969** годах. Язык назван в честь французского математика, физика, литератора и философа Блеза Паскаля, который создал первую в мире механическую машину, складывающую два числа.

В 80-е годы наиболее известной реализацией стал компилятор Turbo Pascal фирмы Borland, в 90-е ему на смену пришла среда программирования Delphi, которая стала одной из лучших сред для быстрого создания приложений под Windows. Delphi ввела в язык Паскаль ряд удачных объектно-ориентированных расширений, обновленный язык получил название Object Pascal.

Язык программирования PascalABC.NET - это язык Pascal нового поколения, включающий в себя все возможности стандартного языка Pascal, расширения языка Delphi Object Pascal, собственные расширения, а также обеспечение возможности совместимости с другими NET-языками.

PascalABC.NET является языком мультипарадигменным - на нем можно программировать в различных стилях:

-структурное программирование,
-объектно-ориентированное программирование,

-функциональное программирование.

Программа содержит ключевые слова, идентификаторы, комментарии. Ключевые слова используются для выделения синтаксических конструкций и подсвечиваются жирным шрифтом в редакторе. Идентификаторы являются именами объектов программы и не могут совпадать с ключевыми словами.

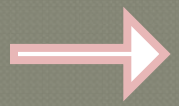
Программа - это набор команд (инструкций), которые управляют работой компьютера.

Структура программы на языке программирования PASCAL :

```
program имя программ;
 раздел описаний
begin
   операторы;
end.
```

Первая строка называется ***заголовком программы*** и не является обязательной.

```
program test2;
var a,b:real;
begin
Read(a);
b := a + 2;
a := (a + 2) * (b - 3);
  write('результат:',a:5:2);
end.
```



```
var a,b:real;
begin
Read(a);
b := a + 2;
a := (a + 2) * (b - 3);

write('результат:',a:5:2);
end.
```

Раздел описаний может включать разделы описания переменных, констант, типов, процедур и функций, которые следуют друг за другом в произвольном порядке.

Операторы отделяются один от другого символом "точка с запятой". В конце программы ставится оператор END.

Раздел описания переменных начинается со служебного слова **var.**

var <список имен переменных>**:** тип**;**

Имена в списке перечисляются через запятую.

```
program test2;
var a,b:real;
Begin
операторы
end.
```

Раздел описания именованных констант начинается со служебного слова **const**

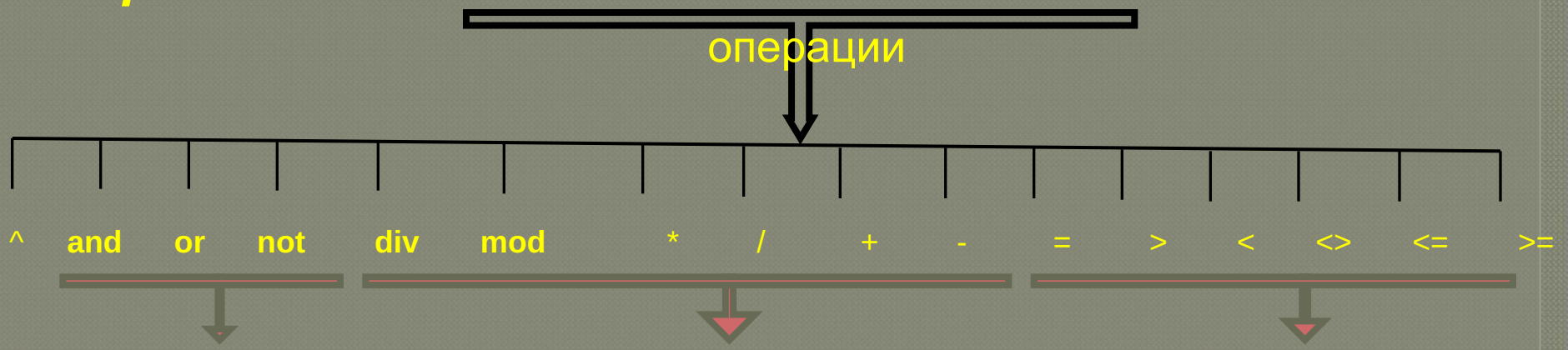
const <имя константы> = <значение>;

или

const <имя константы> : <тип> = <значение>;

```
program test2;
Var b:real;
Const a=20;
begin
Read(a);
b := a + 2;
a := (a + 2) * (b - 3);
  write('результат:',a:5:2);
end.
```

Данные, к которым применяются операции, называются ***операндами.***



Простейшими выражениями являются переменные и константы. Более сложные выражения строятся из более простых с использованием операций, скобок

Выражение, имеющее числовой тип, называется ***арифметическим.*** Выражение имеет тип integer или real.

Выражение, имеющее тип boolean, называется ***логическим***.

Выражение, имеющее тип **string**, называется ***строковыми***.

Константа – постоянная величина, имеющая имя.

Переменная – изменяющаяся величина, имеющая имя (ячейка памяти).

Переменная – это величина, имеющая имя, тип и значение. Значение переменной можно изменять во время работы программы

```
i2 = 45; { целое число }
pi = 3.14; { вещественное  число }
qq = 'Вася'; { строка символов }

L  = True; { логическая величина }
```

может принимать два значения:

- True (истина, "да")
- False (ложь, "нет")

Типы переменных:

- integer { целая }
- real { вещественная }
- char { один символ }
- string { символьная строка }
- boolean { логическая }

Процедура – вспомогательный алгоритм, описывающий некоторые действия.

Функция – вспомогательный алгоритм для выполнения вычислений (вычисление квадратного корня, sin).

Идентификаторы служат в качестве имен программ, модулей, процедур, функций, типов, переменных и констант.

Любой используемый в блоке идентификатор должен быть предварительно описан. В одном блоке не может быть описано двух переменных, констант или типов с одним именем

В блоке может быть описано несколько процедур или функций с одним именем, но с разным набором параметров

Область действия идентификатора простирается от момента описания до конца блока, в котором он описан.

Идентификатором считается любая последовательность латинских букв или цифр, начинающаяся с буквы.

var a, b:real;

program test2;

var a33 ,b_ST:real;

ВЕРНО

НЕВЕРНО

program 2test;

program мир22ST;

```
program test2;
var a,b:real;
begin
Read(a);
if b>0 then
begin
b := a + 2;
a := (a + 2) * (b - 3);
end;
  write(‘результат:’,a:5:2);
end.
```

Блоком называется раздел описаний, после которого следуют операторы, заключенные в операторные скобки

begin / end.

Оператор присваивания имеет вид:

переменная := выражение

Простое логическое выражение состоит из переменных или выражений, связанных операцией отношения:

= (равно);

<> (не равно);

< (меньше чем);

<= (меньше чем или равно);

> (больше чем);

>= (больше чем или равно)

Используя ключевые слова AND (И) или OR (ИЛИ) можно объединить вместе несколько простых логических выражений.

(X>0) or (Y>0)

```
program test2;
var a,b:real;
begin
Read(a);
if  (b>0) or (a<>0)  then
begin
b := a + 2;
a := (a + 2) * (b - 3);
end;
  write('результат:',a:5:2);
end.
```

Арифметическое выражение может включать

- константы
- имена переменных
- знаки арифметических операций: `+ - * / div mod`

- вызовы функций
- круглые скобки ( )

Оператор присваивания служит для изменения значения переменной

{Пример:

вычисление значения переменной}

```
program test2;
var a,b:real;
begin
Read(a);
b := a + 2;
a := (a + 2) * (b - 3);
  write('результат:',a:5:2);
end.
```

Порядок выполнения операций

- вычисление выражений в скобках
- умножение, деление, `div`, `mod` слева направо
- сложение и вычитание слева направо

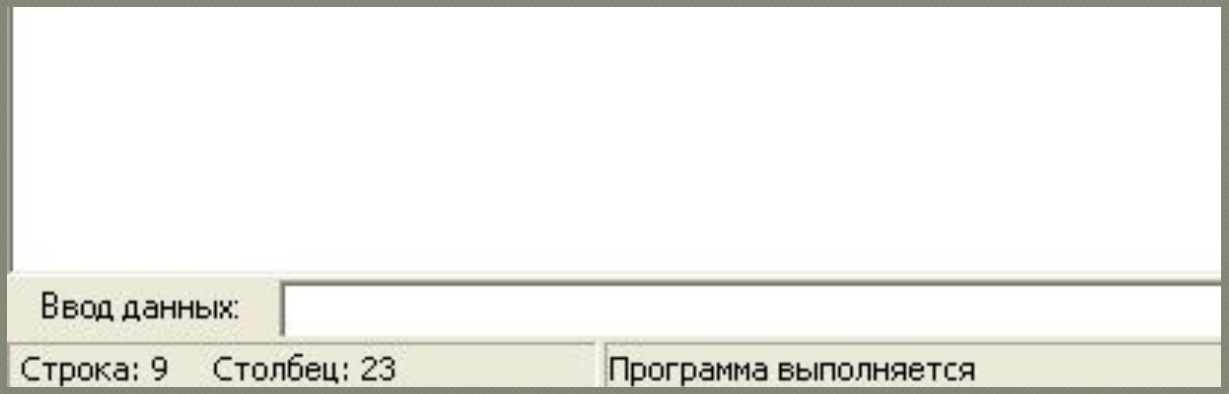
```
          2 3 5 4   1    7 8   6   9
z := (5*a*c+3*(c-d))/a*(b-c)/ b;
```

$$z = \frac{5ac + 3(c - d)}{ab}(b - c)$$

Для ввода предпочтительно использовать функции ReadIn, Read.

Read(a);

После выполнения этого оператора, появляется строка вывода.



Вводимые значения необходимо разделять пробелами, а завершать ввод - нажатием клавиши Enter. Можно также нажимать Enter после каждой вводимой величины.

Для вывода в окно вывода используются стандартные процедуры

write **или** writeln

Параметры в списке перечисляются через запятую

writeln(f**,** 'abc:‘ **,**l);

В процедурах вывода write и writeln после каждого выводимого значения типа может указываться ***формат вывода*** вида перем:m:n, где m-количество знаков в выводимой переменной, n-количество знаков после запятой

writeln(f, ‘abc‘,I:6:2);

формат вывода

Комментарий – это любой не исполняемый текст, заключённый в фигурные скобки или выделенный //

// текст комментария

{ Текст комментария }

{Пример:

вычисление значения переменной}

```
program test2;
var a,b:real;
begin
Read(a);
b := a + 2;
a := (a + 2) * (b - 3);
  write('результат:',a:5:2);//вывод значения
end.
```

Стиль программирования

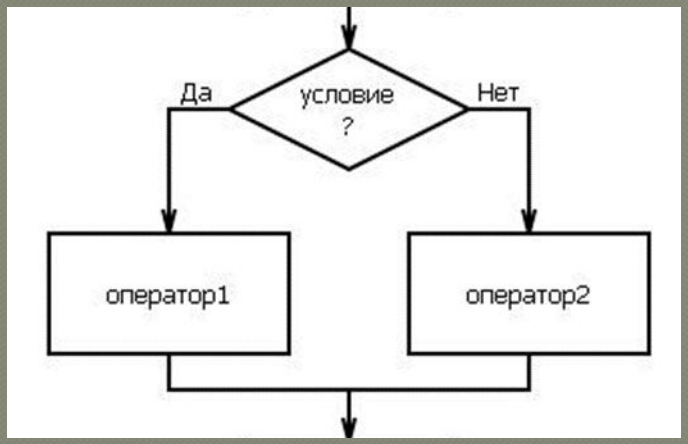
Работая над программой, программист, особенно начинающий, должен хорошо представлять, что программа, которую он разрабатывает, предназначена, с одной стороны, для пользователя, с другой — для самого программиста. Программа должна быть легко читаемой, ее структура должна соответствовать структуре и алгоритму решаемой задачи. Как этого добиться? Надо следовать правилам хорошего стиля программирования. Стиль программирования — это набор правил, которым следует программист (осознано или потому, что "так делают другие") в процессе своей работы. Очевидно, что хороший программист должен следовать правилам хорошего стиля.

- использование комментариев;
- использование несущих смысловую нагрузку имен переменных, процедур и функций;
- использование отступов;
- использование пустых строк.

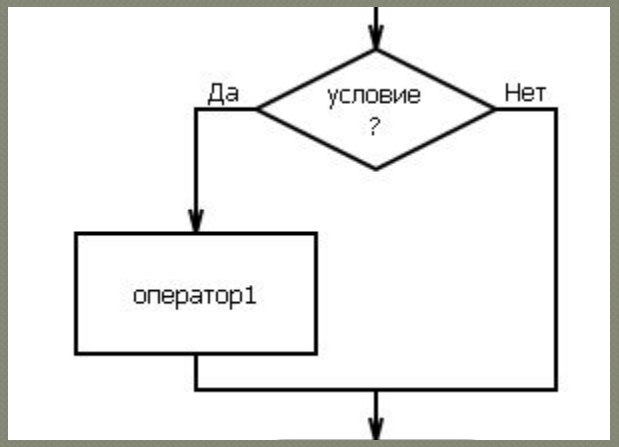
- Следование правилам хорошего стиля программирования значительно уменьшает вероятность появления ошибок на этапе набора текста, делает программу легко читаемой, что, в свою очередь, облегчает процессы отладки и внесения изменений

Условный оператор

Полная форма условного оператора



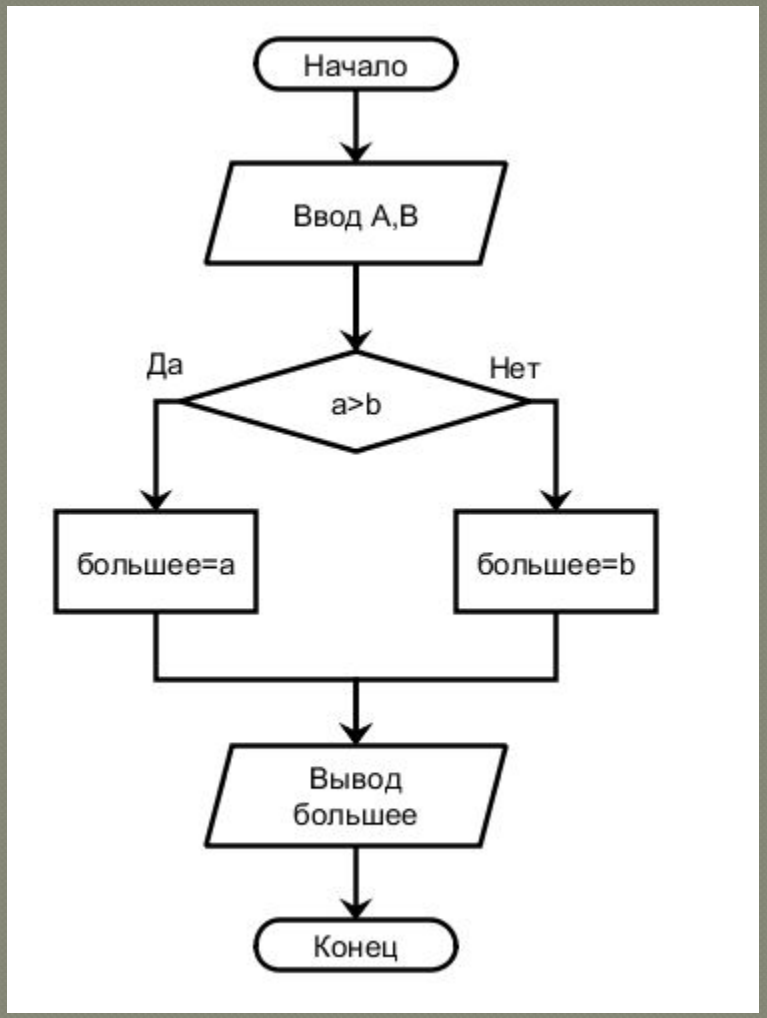
Краткая форма условного оператора



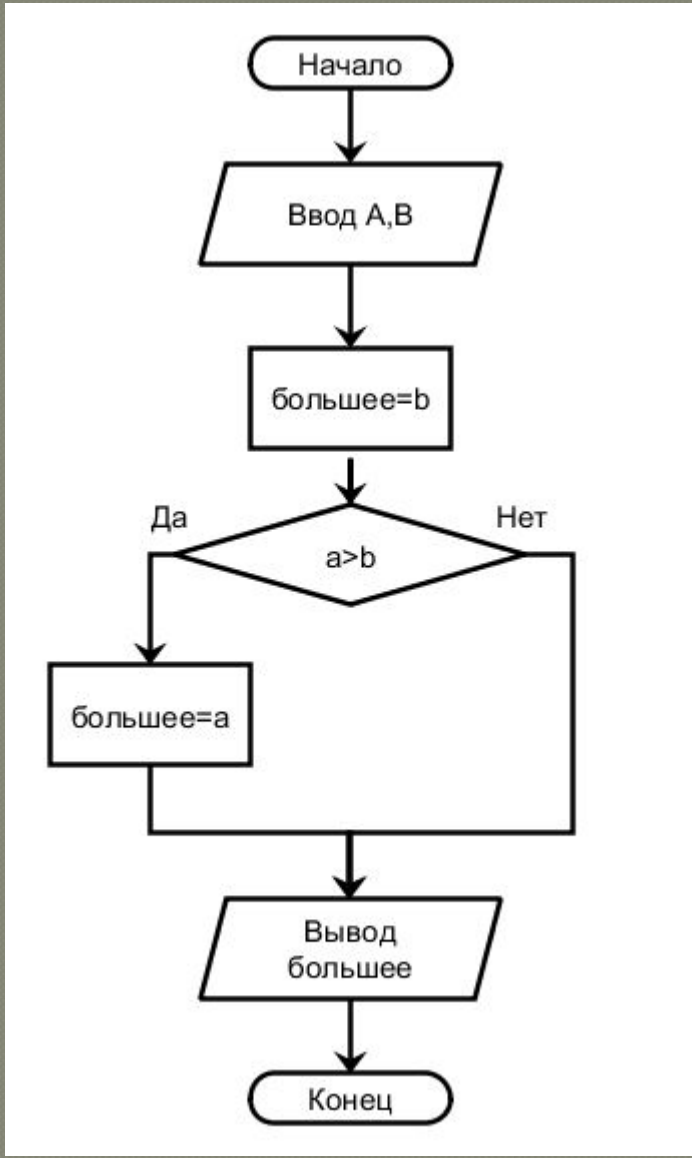
if условие
then
оператор1
else
оператор2

if условие
then
оператор1

Вариант 1. Схема алгоритма



Вариант 2. Схема алгоритма



Условный оператор имеет *полную* и *краткую* формы.

Полная форма условного оператора выглядит следующим образом:

if <условие> **then** <оператор1>

else <оператор2>;

Перед ключевым словом else точка с запятой не ставится.

Краткая форма условного оператора имеет вид:

if <условие> **then** <оператор>;

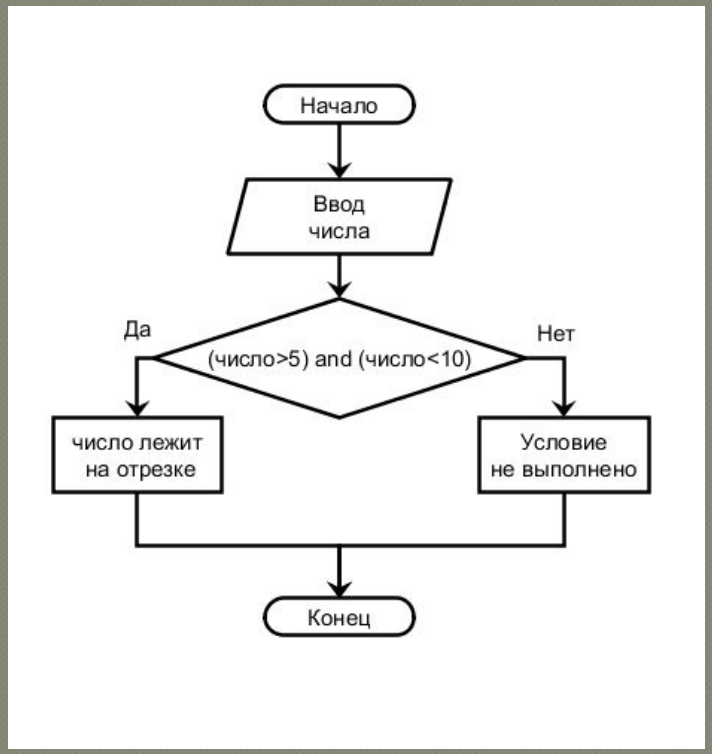
Составные логические выражения

Составное условие – это условие, состоящее из нескольких простых условий (отношений), связанных с помощью логических операций:

- `not` – НЕ (отрицание, инверсия)
- `and` – И (логическое умножение, конъюнкция, одновременное выполнение условий)
- `or` – ИЛИ (логическое сложение, дизъюнкция, выполнение хотя бы одного из условий)
- `xor` – исключающее ИЛИ (выполнение только одного из двух условий, но не обоих)

Простые условия (отношения)

`<` `<=` `>` `>=` `=` `<>`




```
if (число > 5) and (число < 10) then
    WRITELN('введенное число лежит на отрезке[5,10]')
  else
 WRITELN('условие не выполнено')
```

Порядок выполнения

- выражения в скобках
- `not`
- `and`
- `or, xor`
- `<, <=, >, >=, =, <>`

Особенность – каждое из простых условий обязательно заключать в скобки.

Пример

```
     4      1      6     2       5      3
if not (a > b) or (c <> d) and (b <> a)
then begin
   ...
end
```

Оператор выбора case…end

Оператор выбора выполняет одно действие из нескольких в зависимости от значения некоторого выражения, называемого переключателем. Он имеет следующий вид:

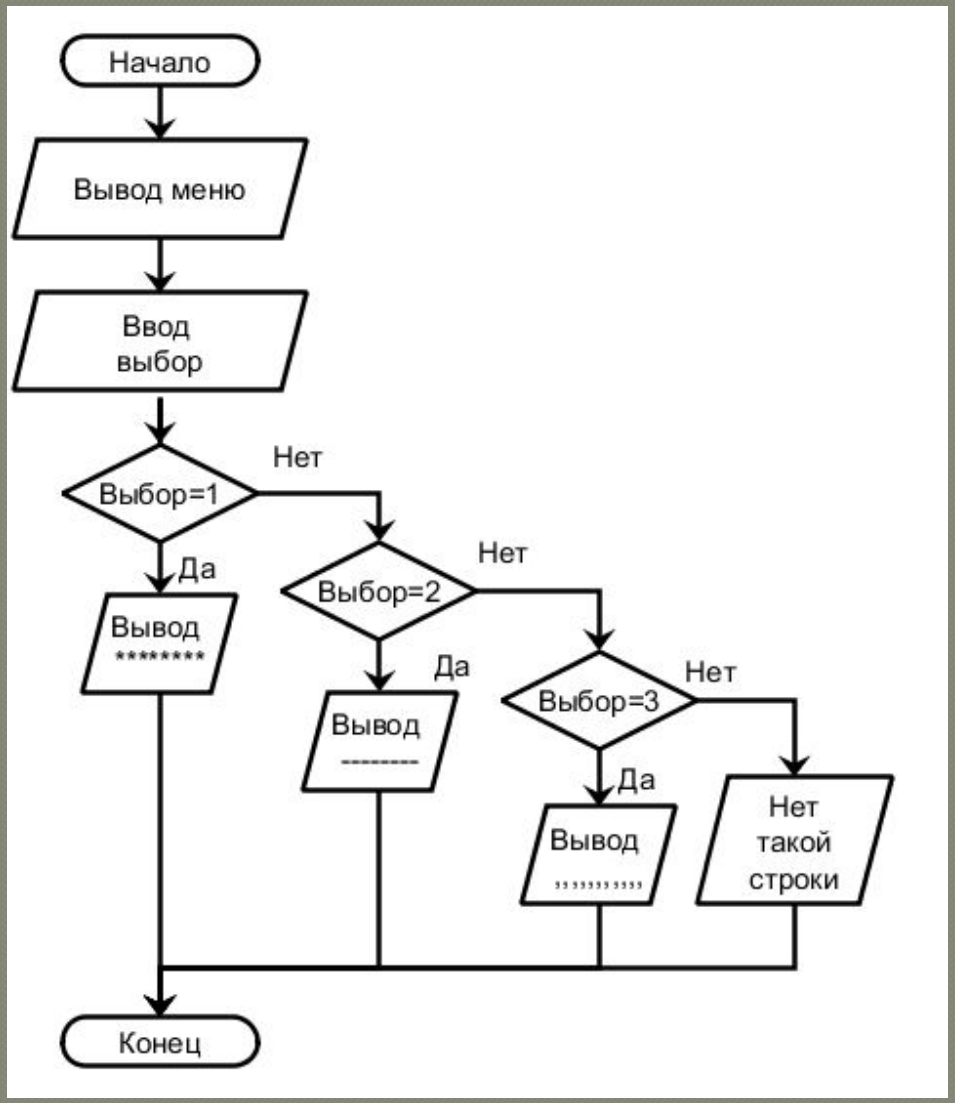
```
case переключатель of
  список выбора 1: оператор1;
  ...
  список выбора N: операторN;
  else оператор0
end;
```

«Переключатель» представляет собой выражение, а списки выбора содержат константы совместимого типа. Как и в операторе **if**, ветка **else** может отсутствовать.

Оператор **case** работает следующим образом. Если в одном из списков выбора найдено текущее значение переключателя, то выполняется оператор, соответствующий данному списку. Если же значение переключателя не найдено ни в одном списке, то выполняется оператор по ветке **else** или, если ветка **else** отсутствует, оператор **case** не выполняет никаких действий.

Список выбора состоит либо из одной константы, либо из диапазона значений вида **a..b** (константа a должна быть меньше константы b); можно также перечислить несколько констант или диапазонов через запятую:

```
case ДеньНедели of
  1..5: writeln('Будний день');
  6,7: writeln('Выходной день');
end;
```


Начало
Вывод меню
Ввод
выбор
Выбор=1
Нет
Да
Вывод
****
Выбор=2
Нет
Да
Вывод

Выбор=3
Нет
Да
Вывод
,,,,,,,,,,
Нет
такой
строки
Конец

```
program N_15_4;
 var выбор: integer;
begin
  WRITEln('Меню команд');
 WRITEln('1. Вывести строку из звёздочек');
 WRITEln('2. Вывести строку из минусов');
 WRITEln('3. Вывести строку из запятых');
WRITEln('Ваш выбор?');
  case выбор  of
    1: WRITE ('******************');
    2: WRITE ('------------------');
    3: WRITE (',,,,,,,,,,,,,,,,,,');
  else
    WRITEln('Нет такой строки в меню')
  end;
end.
```

Оператор цикла while

Оператор цикла **while** имеет следующую форму:

while <условие>
do
оператор;

```
//  Найти произведение нечетных чисел. Количество чисел задать
Program ch;  //Заголовок программы
var i,P,n,S:integer;
begin
CLS;       //Очистка экрана
P:=1;
S:=1;
write('Введите n-количество чисел: ');
read(n);
write('Произведение нечетных чисел:   ');
while i<n do
begin
P:=P*S;  //Нахождение произведения нечетных чисел
write(S,'    ');
S:=S+2; //Получение нечетных чисел
i:=i+1;  //увеличение иттерации на один
end;

write('   равно=',P);
end.
```

```
Введите n-количество чисел: 5
Произведение нечетных чисел:   1    3    5    7    9       равно=945
```

Оператор цикла repeat

Оператор цикла **repeat** имеет следующую форму:

repeat

операторы

until <условие>;

Отличие оператора цикла repeat от while является то, что, по крайней мере, один раз оператор в теле цикла выполнится. Поскольку условие выхода проверяется в конце. Таким образом, пока условие истинно, программа идет на следующую итерацию, условие нарушается – выходим. Поэтому **оператор repeat ещё называют оператором выхода**.

Чтобы прервать зациклившуюся программу, следует либо использовать специальную кнопку, либо комбинацию клавиш

Ctrl-F2 .

```
//Найти от a до b кв корень числа, с шагом 3
var k,a,b:real;
i:integer;
begin
cls;
write('a:');
read(a);
write('b:');
read(b);
k:=a;
repeat
writeln;//пустой оператор
writeln('число:',k,' его корень:',sqr(k));

k:=k+3; //шаг
writeln('добавим шаг 3, получим новое число:',k);
until k>b;
writeln;//пустой оператор
writeln('новое число ', k, ' больше, чем b=', b);
end.
```

```
a:5
b:10

число:5 его корень:25
добавим шаг 3, получим новое число:8

число:8 его корень:64
добавим шаг 3, получим новое число:11

новое число 11 больше, чем b=10
```

Оператор безусловного перехода goto

Оператор безусловного перехода goto имеет следующую форму:

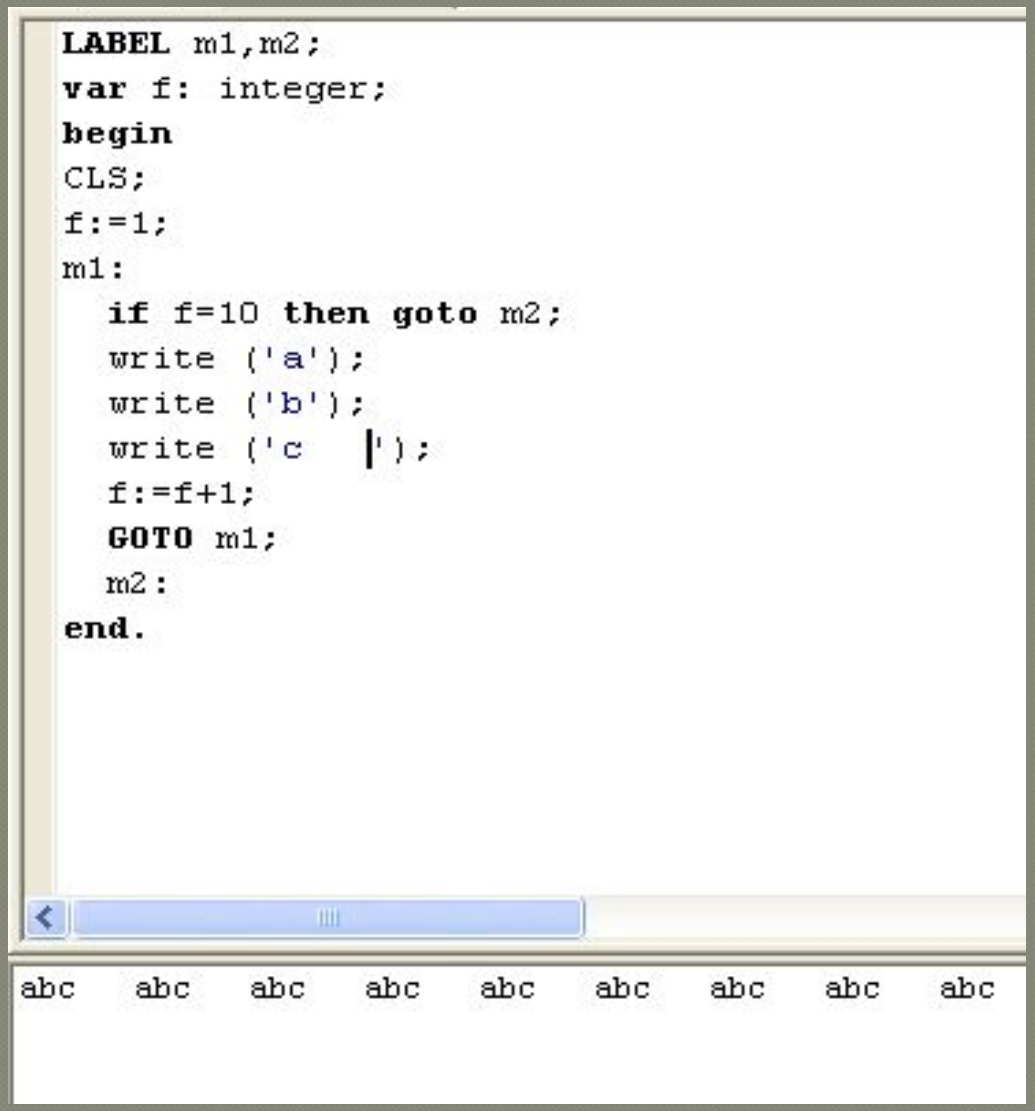
goto метка

Он переносит выполнение программы к оператору, помеченному меткой m1:

Метка представляет собой идентификатор или целое без знака. Чтобы пометить оператор меткой, необходимо перед оператором указать метку с последующим двоеточием

m2:end.

Метка должна помечать оператор в том же блоке, в котором описана. Метка не может помечать несколько операторов. Переход на метку может осуществляться либо на оператор в том же блоке, либо на оператор в объемлющей конструкции. Так, запрещается извне цикла переходить на метку внутри цикла. Оператор перехода или оператор Goto в Pascal (он также называется меткой) — это цикл, т.е. цикличное повторение одного или нескольких операторов.

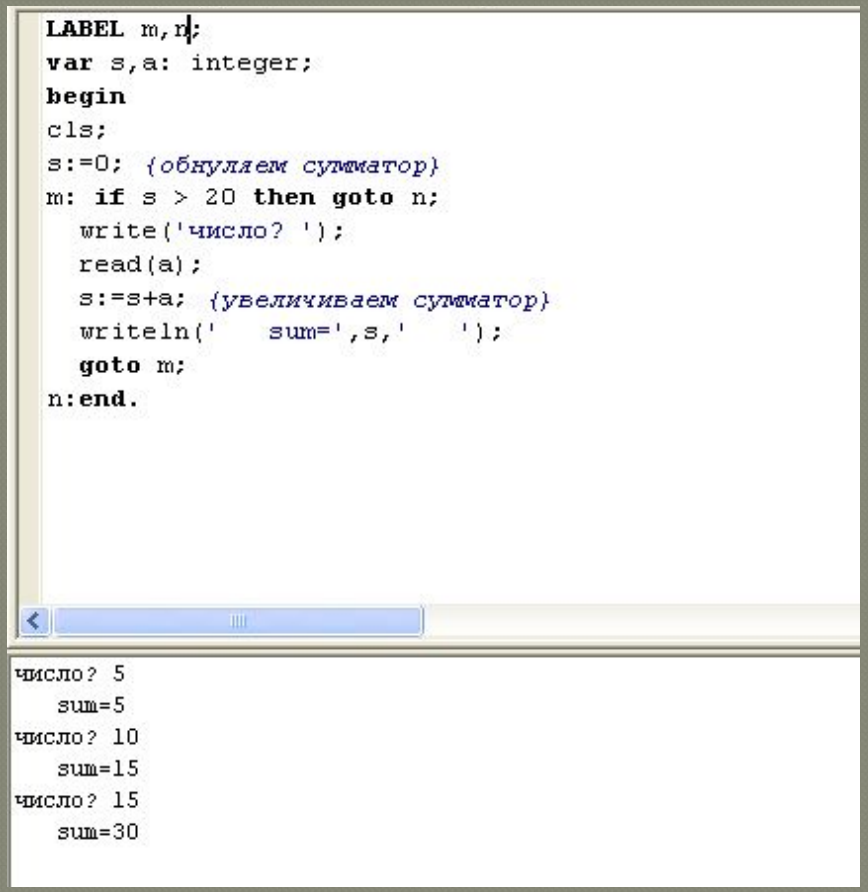

LABEL m1,m2;
var f: integer;
begin
CLS;
f:=1;
m1:
if f=10 then goto m2;
write ('a');
write ('b');
write ('c ');
f:=f+1;
GOTO m1;
m2:
end.
abc abc abc abc abc abc abc abc abc

Особенностью цикла является то, что для выхода из цикла необходимо использовать условный оператор if или другие, предусмотренные для выхода из цикла операторы. Если этого не сделать, то будет организован бесконечный цикл.

```
LABEL m;
var s,a: integer;
begin
cls;
s:=0; {обнуляем сумматор}
m:
   write('число? ');
   read(a);
   s:=s+a; {увеличиваем сумматор}
   writeln('    sum=',s,'    ');
   goto m;
end.
```

```
число? 3
   sum=3
число? 5
   sum=8
число? 12
   sum=20
число?
```

Для выхода из цикла необходимо использовать условие. Условный оператор if позволяет прервать выполнение цикла и выйти из программы. Однако в этом случае уже необходимы две метки в программе: одна — для циклических возвратов, другая — для выхода из цикла.



```
LABEL m,n;
var s,a: integer;
begin
cls;
s:=0; {обнуляем сумматор}
m: if s > 20 then goto n;
  write('число? ');
  read(a);
  s:=s+a; {увеличиваем сумматор}
  writeln('   sum=',s,'   ');
  goto m;
n:end.
```

```
число? 5
   sum=5
число? 10
   sum=15
число? 15
   sum=30
```

Добавлена метка n: и условный оператор

if s > 20 then goto n

Как только sum > 20 происходит переход на метку n: и выход из программы.

Пример: Необходимо вывести в обратном порядке заданное количество элементов. Перед вами пример решение одной и той же задачи разным способом

```
var i:integer;
label 1,2;
begin
cls;
Write ('Число?');
read(i);
2:if i<0 then goto 1;
write(' ',i);
Dec(i)  ;
goto 2;
1:end.
```

```
Число?5
 5 4 3 2 1 0
```

Оператор безусловного перехода goto

```
var i,n: integer;
a: array [1..10] of integer;
begin
cls;
 write('число? ');
  read(n);
For i:=n downto 0 do
  write(i,' ');
end.
```

```
число? 5
5 4 3 2 1 0
```

Оператор цикла с параметром

ПРИМЕЧАНИЕ: Использование оператора безусловного перехода в программе считается признаком плохого стиля программирования. Для основных вариантов использования goto в язык Паскаль введены специальные процедуры: break - переход на оператор, следующий за циклом, exit - переход за последний оператор процедуры, continue - переход за последний оператор в теле цикла. Единственный пример уместного использования оператора goto в программе - выход из нескольких вложенных циклов одновременно.

Оператор цикла for

Оператор цикла **for** имеет одну из двух форм:

for <переменная> := <начальное значение> **to** <конечное значение> **do**

параметр цикла

заголовок цикла

оператор

тело цикла

возрастающий

или

For <переменная>:=<конечное значение> **downto** <начальное значение> **do**

Оператор

убывающий

В зависимости от направления изменения параметра цикла (возрастание - **to** или убывание - **downto**) в языке Паскаль оператор цикла **for** может быть записан в одной из двух форм. Параметр цикла, вне зависимости от возрастания или убывания, всякий раз изменяется на единицу.

В большинстве программ встречается необходимость многократного выполнения некоторого оператора (или блока операторов). Для организации подобного рода конструкций могут использоваться операторы цикла. Итерация цикла - однократное повторение тела цикла

Если **число повторений тела** цикла заранее известно, то используется **оператор цикла for**, который также часто называют оператором цикла с параметром.

Рассмотрим работу цикла **for**

Перед началом выполнения оператора цикла вычисляются начальное значение, присваиваемое переменной-параметру, и конечное значение. Затем, циклически выполняются следующие операции:

1. Сравнивается текущее значение параметра с конечным значением.
2. Если условие параметр <= кон_знач истинно, то выполняется тело цикла, в противном случае оператор **for** завершает работу и управление передается оператору, следующему за циклом.

Если в теле цикла необходимо использовать более одного оператора, то применяется составной оператор (операторные скобки **begin** / **end**).

```
program v; //неизвестно
var x:integer;
y,z:real;
const L=3;
begin
writeln('найти значение 2-х функций');
writeln('----------------------------');
writeln('   x    :    y    :   z   ');
writeln('----------------------------');
for x:=-5 to 5 do

if x<>0 then
begin
y:=L*sin(x);
z:=cos(x)/x;
writeln('   ',x:2,'   :   ',y:4:1,'   :   ',z:4:1);
end;

end.
```

```
найти значение 2-х функций
----------------------------
   x    :    y    :   z
----------------------------
   -5   :    2.9   :   -0.1
   -4   :    2.3   :    0.2
   -3   :   -0.4   :    0.3
   -2   :   -2.7   :    0.2
   -1   :   -2.5   :   -0.5
    1   :    2.5   :    0.5
    2   :    2.7   :   -0.2
    3   :    0.4   :   -0.3
    4   :   -2.3   :   -0.2
    5   :   -2.9   :    0.1
```

Простейшие алгоритмы обработки массивов

Вместо того, чтобы присваивать уникальное имя каждой отдельной переменной, можно задать одно имя для целого их массива, а каждый элемент этого массива идентифицировать с помощью числового индекса. Правила образования имен массивов те же, что и для имен простых переменных.

Для обращения к отдельному элементу массива достаточно после имени массива написать индекс, заключенный в скобки A[i]

Отдельные переменные в массиве называются элементами массива, а максимальное число элементов в массиве называется размером массива.

$$A=(a_1 \quad a_2 \quad a_3 \quad a_4 \quad a_5 \quad a_6 \quad a_7)$$

Кроме размера массив может иметь размерность - число индексов. Массивы, имеющие один индекс, называются одномерными, а массивы, имеющие больше одного индекса - многомерными.

Тип массива в разделе описания конструируется следующим образом:

array [тип индекса1, ..., тип индексаN] of базовый тип

```
Var
  B: array [1..3,1..3] of integer;
```
// массив B размерностью 2 целого типа, содержит 3 строки и 3 столбца;

При описании можно также задавать инициализацию массива значениями, используя для этого ключевое слово const.

```
var
  const a: array [1..10] of integer:= (1,2,3,4,5,6,7,8,9,0);
```

или

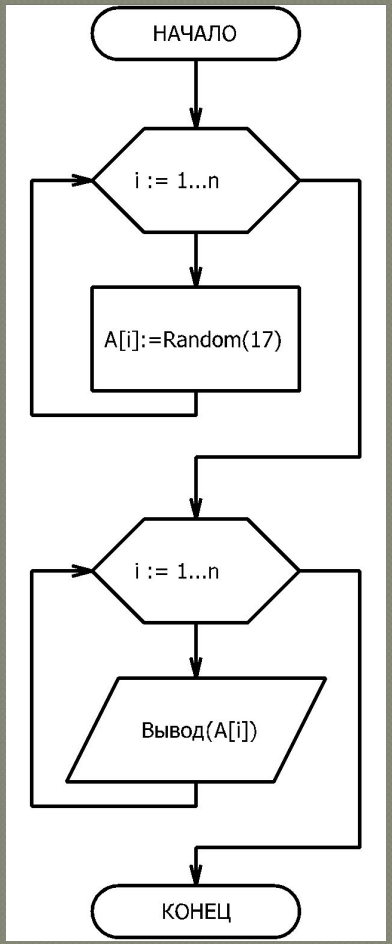
```
Var
  const d: array [1..3,1..4] of real := ((1,2,3,4), (5,6,7,8), (9,0,1,2));
```

Элементам массива можно присвоить значение и с помощью генератора случайных чисел function **Random**.

```
const
  N = 7;     // присваиваем константе N  значение
 var
  A: array[1..N] of Integer;      {описание массива
                                   размером N}
  i: Integer;
begin
for i := 1 to N do
    A[i] := Random(17);
```

{заполнение массива случайными числами Random(maxV: integer); возвращает случайное целое в диапазоне maxV-1, в нашем случае от 0 до 16}

```
for i := 1 to N do
    write(A[i]:4);
end.
```

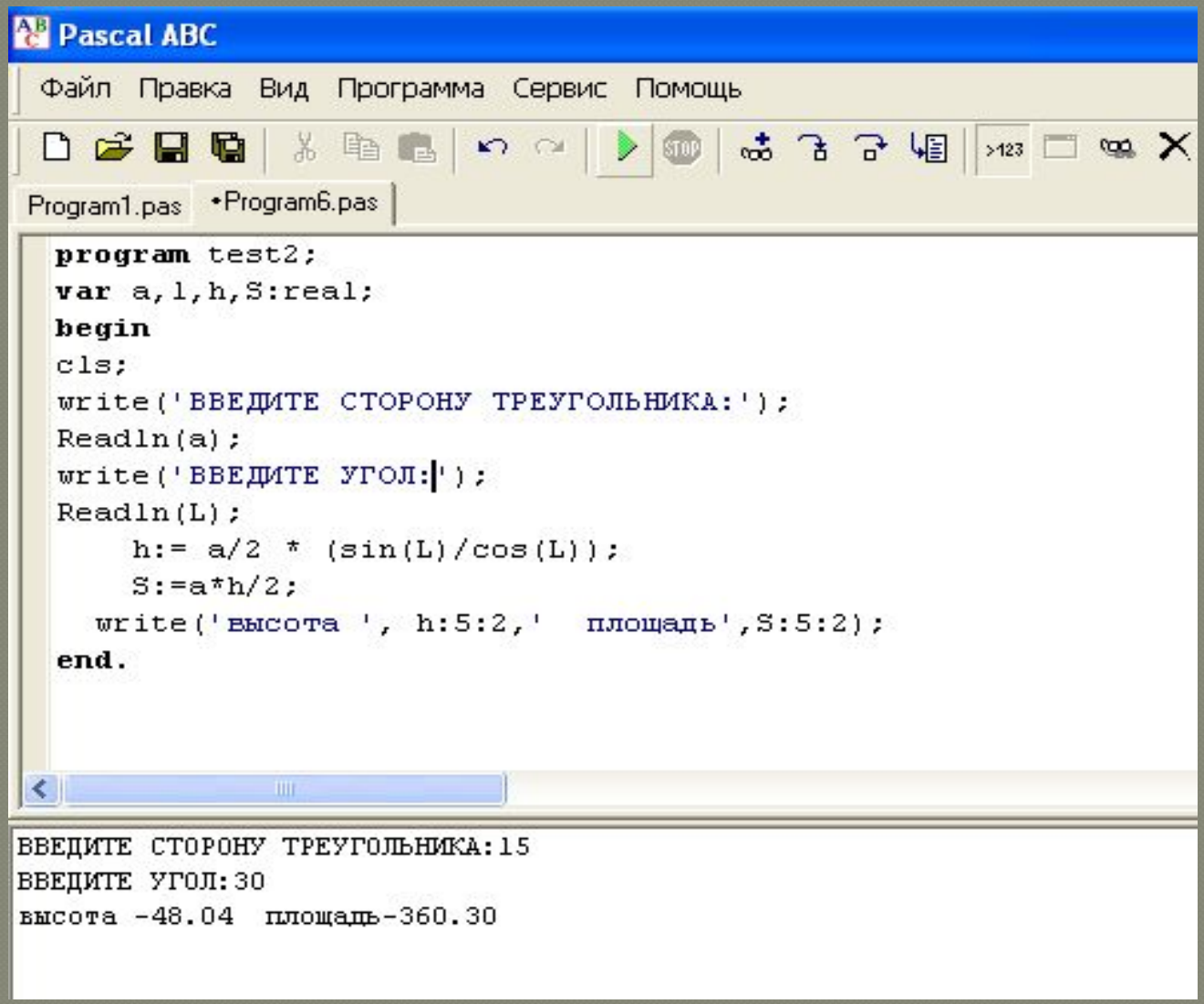


Стандартные функции и процедуры

Имя и параметры	Действие
Abs(x)	возвращает абсолютное значение (модуль) x
Sqr(x)	возвращает квадрат x
Sqrt(x)	возвращает квадратный корень из x
Sin(x)	возвращает синус x
Cos(x)	возвращает косинус x
Ln(x)	возвращает натуральный логарифм x
Exp(x)	возвращает e в степени x (e=2.718281...)
Power(x,y)	возвращает x в степени y
Round(x)	возвращает результат округления x до ближайшего целого
Int(x)	возвращает целую часть x
Frac(x)	возвращает дробную часть x
Random	возвращает случайное вещественное в диапазоне [0..1)

Функция - имя со списком параметров в виде констант, переменных или выражений

Выражение в скобках называется аргументом функции


Pascal ABC
Файл Правка Вид Программа Сервис Помощь
Program1.pas
•Program6.pas
program test2;
var a,l,h,S:real;
begin
cls;
write('ВВЕДИТЕ СТОРОНУ ТРЕУГОЛЬНИКА:');
Readln(a);
write('ВВЕДИТЕ УГОЛ:');
Readln(L);
h:= a/2 * (sin(L)/cos(L));
S:=a*h/2;
write('высота ', h:5:2,' площадь',S:5:2);
end.
ВВЕДИТЕ СТОРОНУ ТРЕУГОЛЬНИКА:15
ВВЕДИТЕ УГОЛ:30
высота -48.04 площадь-360.30

СПАСИБО ЗА ВНИМАНИЕ