МИНОБРНАУКИ РОССИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«САМАРСКИЙ ГОСУДАРСТВЕННЫЙ АЭРОКОСМИЧЕСКИЙ
УНИВЕРСИТЕТ ИМЕНИ АКАДЕМИКА С.П. КОРОЛЕВА
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)»

М.С. Стенгач

Лекционный ресурс электронного курса «Информатика» (система дистанционного обучения MOODLE)

Электронное интерактивное учебное пособие

С А М А Р А

2011

Автор: **Стенгач Михаил Сергеевич**

Стенгач М.С., Лекционный ресурс электронного курса «Информатика» (система дистанционного обучения «Moodle») [Электронный ресурс] : электрон. интерактив. учеб. пособие / М.С. Стенгач; Минобрнауки России, Самар. гос. аэрокосм. ун-т им. С. П. Королева (нац. исслед. ун-т). - Электрон. текстовые и граф. дан. (1,6 Мбайт). - Самара, 2011. - 1 эл. опт. диск (CD-ROM). - Систем. требования: ПК Pentium; Windows 98 или выше.

Представлен комплект лекций по курсу «Информатика», использованный как лекционный ресурс в системе дистанционного обучения MOODLE. Система размещена на сайте кафедры общей информатики СГАУ. СОД MOODLE предназначена для интерактивного освоения материалов курса и дает возможность контроля текущих знаний студентов.
Лекции включают основные положения дисциплины «Информатика». Даются основы программирования на алгоритмических языках Паскаль и С++, рассматриваются многочисленные примеры решения стандартных задач.
Учебный курс ориентирован на 1 факультет, 1 курс, I семестр (очное) для подготовки:

- бакалавров по направлениям подготовки 220700.62, 221700.62, 221400.62
- студентов по специальностям 160100.65, 010701.65, 160400

Подготовлено на кафедре общей информатики СГАУ.

ЛЕКЦИЯ 1

* * *

В курсе лекций изучается программирование на языке Паскаль (с 2003 года – язык Delphi) и работа в визуальной среде программирования Lazarus.

Алфавит языка Паскаль

Алфавит Паскаля содержит следующие символы:

- Заглавные и строчные латинские буквы и символ "подчеркивание":
A...Z, a...z, _
- Арабские цифры: 0...9
- Двадцать два специальных символа:
( ) [ ] { } + - * / . , ; : = > < # $ ^ @ '

Русские буквы и другие символы могут использоваться только для записи комментариев и в символьных константах.

Структура программы. Комментарии

В общем виде программа на Паскале состоит из заголовка (содержит слово program и имя программы), раздела описаний и выполняемой части.

Раздел описаний начинается сразу за заголовком программы. Выполняемая часть начинается со слова BEGIN и заканчивается словом END, после которого обязательно ставится точка (конец программы):

```
program <имя программы>;
   < РАЗДЕЛ ОПИСАНИЙ >
BEGIN
   < ВЫПОЛНЯЕМАЯ ЧАСТЬ >
END.
```

Операторы могут располагаться в программе произвольным образом, но обязательно должны заканчиваться точкой с запятой.

Для пояснения текста программы в нее могут включаться комментарии. Комментарий – это произвольная последовательность символов, находящаяся:

1. между фигурными скобками.
2. между круглыми скобками со звездочками: (* … *).
3. после двух символов //.

Пример простой программы:

```
program primer;                    (* заголовок программы *)
  var x,y,z:real;                  // описание переменных
begin                            { начало выполняемой части }
  readln(x,y);                        // ввод чисел X и Y
  z:=x*y;                                // вычисление Z
  writeln(z);                  // вывод на экран значения Z
end.                                        { конец программы }
```

Консольные приложения в среде программирования Lazarus

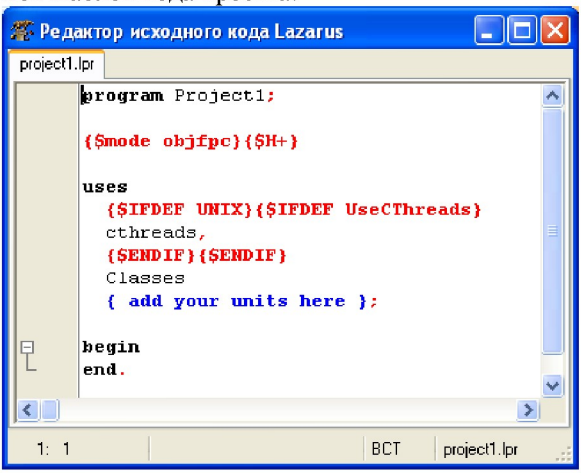
После запуска Delphi на экране появляются несколько окон, из которых нас пока интересуют только два: Главное окно и Редактор исходного кода.

Главное окно осуществляет управление проектом. Здесь располагается главное меню.

Командным кнопкам соответствуют следующие горячие клавиши:

F9 – компилирует и выполняет программу,
Ctrl+F2 – завершение работы программы в случае зависания,
F8 – пошаговое выполнение программы,
F7 – пошаговое выполнение программы с отслеживанием работы вызываемых подпрограмм.

Редактор исходного кода предназначен для создания и редактирования текста программы. Для создания консольного приложения необходимо выполнить команду `Создать проект.../Программа` из пункта главного меню `Проект`. Появится окно редактора кода. В окне находится шаблон кода проекта:



Идентификаторы

Идентификаторы в языке Паскаль – это имена констант, переменных, меток, типов, процедур и функций. Идентификатор может содержать буквы, цифры и знак подчеркивания и начинаться только с буквы или знака подчеркивания. Пробелы и специальные символы не могут входить в идентификатор.

Важно помнить, что соответствующие заглавные и строчные буквы в идентификаторах и служебных словах не различаются. Таким образом, следующие три идентификатора обозначают одну и ту же переменную: `index`, `INDEX`, `IndEx`.

В качестве идентификаторов нельзя использовать служебные слова. Служебные слова будут рассматриваться по мере изучения языка.

Пример идентификаторов:

правильно	не правильно
`punkt_1`	`1_punkt`
`nomer1`	`#1`
`fort`	`for` (служебное слово)

Оператор присваивания

Оператор присваивания имеет вид: `<переменная>:=<выражение>;`
где символ `:=` означает присвоить.
Пример:

```
a:=2; b:=3;
c:=a+b;  {c=5}
```

Основные типы данных в Паскале

REAL - ВЕЩЕСТВЕННЫЕ ЧИСЛА. Это числа, содержащие целую и дробную части, разделяемые точкой. Для типа `REAL` используются следующие арифметические операции:

+ сложение; - вычитание; * умножение; / деление.

Список операций приведен в порядке повышения приоритета. Так, например, в выражении `a:=2+6/2*3;` сначала выполнится деление `6/2=3`, затем умножение `3*3=9`, а затем сложение `2+9=11`. Выражение в скобках имеет высший приоритет и выполняется первым.

Пример:

```
a:=2+6/(2*3); {рез.=3}
a:=(2+6)/2*3; {рез.=12}
```

INTEGER - ЦЕЛЫЕ ЧИСЛА. Для типа `INTEGER` вместо операции деления / определены две специальные операции:

`DIV` деление с отбрасыванием дробной части;
`MOD` взятие остатка от целочисленного деления.

Пример:

```
b:=5 div 2; {рез.=2}
b:=5 mod 2; {рез.=1}
```

BOOLEAN - ЛОГИЧЕСКИЙ ТИП. Объекты типа `BOOLEAN` могут принимать только два значения: `TRUE` - "истина", и `FALSE` - "ложь".

CHAR - СИМВОЛЫ. В переменную этого типа может быть помещен любой символ. Значение переменной `CHAR` задается в апострофах.

Пример: `a:='a';`

STRING - СТРОКИ. Массив символов.

Пример: `a:='абракадабра';`

Переменные

Каждая переменная должна быть описана в разделе описаний. Описание переменных начинается со слова VAR.

```
VAR <имя_переменной>: <тип_переменной>;
```

Например:

```
Var a: real;
    i,j: integer;
    b: Boolean;
```

Константы

Константы описываются в разделе описаний. Описание константы начинается со слова CONST. Значение константы нельзя изменить в теле программы.

Например:

```
Const  e=2.71828;    // Раздел
Var    x: real;      // описаний
       begin
         x:=e;       // Допустимое действие
         e:=3.14;    // Ошибка!
       end.
```

Основные математические функции

Стандартные математические функции языка Паскаль:

Идентификатор (имя) функции	Функция
PI	Число π
Abs(x)	Абсолютная величина |x|
Exp(x)	Экспонента e^x
Sqr(x)	x^2
Sqrt(x)	Квадратный корень x
Ln(x)	Натуральный логарифм *ln* x
Sin(x)	*Sin* x (угол в радианах)
Cos(x)	*Cos* x (угол в радианах)
ArcTan(x)	*Arctg* x (значение в радианах)
Round(x)	Округляет число до ближайшего целого
Trunc(x)	Отбрасывает дробную часть числа
Int(x)	Целая часть числа
Frac(x)	Дробная часть числа
Randomize	Инициация счётчика случайных чисел
Random(x)	Случайное число в диапазоне 0…(x-1) (число x – целое)

Очень много математических функций добавлено в языке Delphi. Все они находятся в модуле math, который необходимо подключить:

```
uses
  {$IFDEF UNIX}{$IFDEF UseCThreads}
  cthreads,
  {$ENDIF}{$ENDIF}
  Classes, math
  { add your units here };
```

Некоторые из функций модуля math:

`Power(x,y)`	x^y
`Log10(x)`	Десятичный логарифм *lg* x
`LogN(n,x)`	$log_n x$
`Tan(x)`	*tg*x

Процедуры ввода и вывода

Операторы вывода:

`WRITE`. Процедура `Write` выводит на экран выражения без перевода строки. Символьные (текстовые) константы заключаются в апострофы. Например, в результате выполнения следующего фрагмента:

```
A := 7;
Write('A=', a);
Write('The end.');
```

на экране будет напечатано следующее:

```
A= 7The end.
```

`WRITELN`. Процедура `Writeln` отличается от `Write` только тем, что она после завершения вывода переводит текущую позицию в начало следующей строки. Например:

```
A := 7;
Writeln ('A=', a);
Write('The end.');
```

В результате выполнения этого фрагмента на экране будет напечатано следующее:

```
A= 7
The end.
```

Переменные типа `real` выводятся с помощью оператора `write` в экспоненциальной форме. Например:

```
c:=13.4;
write('C=',c);
```

На экран выведется:

```
C=1.34000000000000E+0001
```

Для вывода чисел в нормальной форме необходимо использовать форматный вывод: после имени переменной ставится ":" и указывается общее количество позиций под число, и, через еще одно ":" – количество позиций под дробную часть. Например:

```
write('C=',c:8:3);
```

На экран выведется:

```
C=  13.400
```

Операторы ввода:

READ. Процедура Read выполняет ввод с клавиатуры значений переменных.

Пример:

Read(a); - в этом месте выполнение программы приостанавливается, программа ожидает ввода с клавиатуры переменной a.

READLN. Процедура Readln отличается от Read только тем, что после завершения ввода она переводит текущую позицию в начало следующей строки.

ЛЕКЦИЯ 2

Логические выражения

В логических выражениях используются следующие операции сравнения:

< меньше; > больше; = равно;
<= меньше или равно; >= больше или равно; <> не равно.

Результатом операции сравнения является "истина" (TRUE) или "ложь" (FALSE).

Например:

(X+1)<>2 - имеет значение "ложь", если X=1, и значение "истина", если значение переменной X отлично от 1.

Логические операции:

NOT логическое НЕ;
AND логическое И;
OR логическое ИЛИ.

Запись таких логических выражений, как min < X < max осуществляется следующим образом: (min < X) and (X < max)

Результатом логической операции and является "истина", когда "истине" равны оба сравнения.

Условный оператор IF

Оператор IF предназначен для выбора одного из двух возможных действий в зависимости от некоторого условия.

Структура условного оператора:

```
IF   <условие>
THEN <оператор_1>
ELSE <оператор_2>;
```

где <условие> – логическое выражение.

Если <условие> истинно (TRUE) выполняется <оператор_1>, иначе (<условие>=FALSE) выполняется <оператор_2>.

Простой пример:

```
Program primer_1;
  Var  a: real;
  Begin
    readln(a);
    if a>=0
      then writeln('positive number')
      else writeln('negative number');
  End.
```

Условный оператор может быть также без ELSE-альтернативы:

```
IF   <условие>
THEN <оператор>;
```

Составной оператор

В условном операторе IF после THEN и ELSE можно выполнить только один оператор. Чтобы выполнить группу операторов нужно использовать *составной оператор*, т.е. заключить группу операторов между BEGIN и END;.

Пример. Вычислить и распечатать значение k только для случая a>b

```
if a>b then
          begin
            k:=a+b;
            writeln('k=',k);
          end;
```

Оператор безусловного перехода GOTO. Метки

Оператор GOTO передаёт управление оператору, которому предшествует соответствующая *метка*.

Метка – это идентификатор или целое число.

Каждая метка должна быть предварительно описана в разделе описаний:

```
LABEL <метка>;
```

Структура оператора:

```
GOTO  <метка>;
<метка>: <оператор>;
```

Один оператор может быть помечен несколькими метками, которые в этом случае отделяются друг от друга двоеточиями.

Оператор выбора CASE

Оператор выбора позволяет выбрать одно из нескольких возможных продолжений программы.

Структура оператора:

```
CASE <ключ_выбора> OF
     <выбор_1>: <оператор_1>;
     <выбор_2>: <оператор_2>;
     . . .
     <выбор_k>: <оператор_k>;
 ELSE <оператор>
 END;
```

где

<ключ_выбора> - значение переменной, которая проверяется оператором CASE;

<выбор_1...k> - переменные того же типа, что и <ключ_выбора>.

В последовательности операторов <выбор_1...k> отыскивается такой, значение которого равно значению <ключ_выбора>. Соответствующий найденному выбору оператор выполняется, после чего оператор CASE завершает свою работу. Если переменная, равная значению <ключ_выбора>, не будет найдена, управление передается оператору, стоящему за словом ELSE.

Часть ELSE <оператор> может отсутствовать.

Любому из операторов от 1 до k может предшествовать не одна, а несколько значений <выбора>, разделенных запятыми.

Приведем пример программы случайного предсказания одного из десяти вариантов ближайшего будущего с вероятностью 1/20, в остальных случаях - вы "неудачник".

```
PROGRAM FUTURE;
var N : integer;
BEGIN
  writeln('ПРЕДСКАЗАНИЕ БУДУЩЕГО');
  Randomize;
  N:=Random(20)+1;       { N - случайное число от 1 до 20 }
  writeln;   write('Вас ожидает ');
  case N of
     1 : writeln('счастье');
     2 : writeln('пятерка');
     3 : writeln('дорога');
     4 : writeln('двойка');
     5 : writeln('болезнь');
     6 : writeln('здоровье');
     7 : writeln('деньги');
     8 : writeln('любовь');
     9 : writeln('встреча');
     10 : writeln('дети')
      else writeln('неудача')
  end;
  writeln('Нажми Enter');
  readln;
END.
```

Здесь функция Random(x) генерирует случайное число, с равномерной плотностью распределения на заданном интервале. Для инициализации распределения в начале программы необходимо вызвать процедуру Randomize.

ОПЕРАТОРЫ ЦИКЛА

Цикл - конструкция языка, позволяющая несколько раз выполнять один оператор или группу операторов.

В Паскале имеется три оператора цикла: WHILE, REPEAT и FOR.

Для того чтобы прервать выполнение цикла используется оператор break.

Оператор цикла WHILE

Структура:

```
WHILE  <условие>  DO  <оператор>;
```

где <условие> – логическое выражение.

Если <условие> истинно (TRUE) выполняется <оператор>, который будет выполняться в цикле до тех пор, пока <условие> не станет ложным (FALSE).

Таким образом, цикл выполняется пока <условие> истинно.

Пример. Программа табулирования функции $y=ax^2$:

```
var x,xk,dx,y,a:real;
begin
write('xn=');readln(x);
write('xk=');readln(xk);
write('dx=');readln(dx);
write('a='); readln(a);
while x<=xk do
  begin
  y:=a*x*x;
  writeln(x:7:2,' ',y:7:2);
  x:=x+dx;
  end;
readln;
end.
```

Оператор цикла REPEAT

Структура:

```
REPEAT  <оператор_1>; <оператор_2>; ... <оператор_k>;
UNTIL   <условие>;
```

Отличия REPEAT от ранее рассмотренного оператора цикла WHILE:

1. Условие проверяется в конце (т.о. цикл выполняется хотя бы один раз).

2. Признаком окончания цикла является выполнение условия (имеет значение TRUE). Таким образом, цикл выполняется пока условие ложно.

3. В теле цикла может содержаться произвольное количество операторов (REPEAT и UNTIL - ограничители).

Если для организации цикла использовать оператор REPEAT...UNTIL, то приведенный выше пример табулирования функции $y=ax^2$ будет иметь вид:

```
repeat
  y:=a*x*x;
  writeln(x:7:2,' ',y:7:2);
  x:=x+dx;
until x>xk;
```

ЛЕКЦИЯ 3

Оператор цикла FOR

Структура:

```
FOR <параметр>:=<нач.знач.> TO <кон.знач.> DO <оператор>;
```

где <параметр> - переменная цикла типа INTEGER или CHAR.

Значение <параметра> изменяется в цикле от <нач.знач.> до <кон.знач.> с шагом равным 1 и это определяет количество выполнений <оператора>.

Пример.

```
for i:=1 to 3 do write(i:2);
```

Переменная i изменяется в цикле от 1 до 3 с шагом 1 и оператор write выполняется 3 раза.

Результат работы оператора for: 1 2 3.

Существует другая форма оператора цикла:

```
FOR <параметр>:=<нач.знач.> DOWNTO <кон.знач.> DO <оператор>;
```

Замена служебного слова TO на DOWNTO означает, что шаг изменения переменной <параметр> равен -1.

МАССИВЫ

Массив - это совокупность данных одного типа, занимающая непрерывную область оперативной памяти. Массив имеет имя и состоит из ячеек. Каждая ячейка имеет свой номер (индекс массива). Индексы у массива могут иметь тип integer или char.

Чтобы осуществить доступ к ячейке массива, нужно указать имя массива и номер ячейки в квадратных скобках.

Одномерный массив

Описание массива:

```
Var a: array [1..7] of Integer;
Const b: array [0..3] of Real=(1.1,1.2,-3.2,0.5);
```

	1	2	3	4	5	6	7	
a	-2	4	6	8	-5	12	17	a[6]:=12;

	0	1	2	3
b	1.1	1.2	-3.2	0.5

В качестве индекса массива можно использовать арифметические выражения, имеющие целый результат

```
i:=2;
a[2*i]:=8;          // a[4]:=8;
b[2*i-1]:=0.5;      // b[3]:=0.5;
```

Примеры

1. Ввод элементов массива

```
Var a: array [1..10] of Integer;
    i: Integer;
begin
    for i:=1 to 10 do
        begin
        Write('a[',i:2,']=');
        ReadLn(a[i]);
        end;
end.
```

2. Вывод элементов массива

```
for i:=1 to 10 do  WriteLn('a[',i,']=',a[i]);
```

3. Операция присваивания с массивами

Для массивов одного типа и одной размерности.

```
Var a,b: array [1..10] of Integer;
< Ввод массива a >
b:=a;
```

4. Сумма элементов массива

```
S:=0;
for i:=1 to 10 do  S:=S+a[i];
```

5. Произведение элементов массива

```
P:=1;
for i:=1 to 10 do  P:=P*a[i];
```

6. Подсчет отрицательных элементов массива

```
n:=0;
for i:=1 to 10 do
if a[i] < 0 then n:=n+1;
```

7. В массиве из 10 элементов найти самое большое число

```
max:=a[1];
for i:=2 to 10 do
if a[i] > max then max:=a[i];
```

8. Найти порядковый номер ячейки, содержащей самое большое число

```
k:=1;
for i:=2 to 10 do
if a[i] > a[k] then k:=i;
```

10. Сортировка массива - расположить элементы массива либо по возрастанию, либо по убыванию

1	2	3	4	5
-2	4	6	8	-5

```
for i:=1 to 5 do
for j:=1 to 5-i do
  begin
   if a[j] > a[j+1] then
      begin
        r:=a[j];
        a[j]:=a[j+1];
        a[j+1]:=r;
      end;
  end;
```

Двумерные массивы (матрицы)

Двумерный массив - это совокупность ячеек памяти, расположенных по строкам, доступ к каждой ячейке памяти осуществляется путем задания имени матрицы и двух индексов (1-ый - строка, 2-ой - столбец).

Описание:

```
Var a: array [1..3,1..5] of Real;
Const b: array [1..2,1..3] of Integer=((1,2,3),(4,5,6));
```

		1	2	3	4	5
a	1	-2	4	6	8	-5
	2	3	8	4	0	0.1
	3	4.22	3	9.9	6	1.3

При обработке значений, хранящихся в матрице, необходимо использовать двойной цикл.

Примеры

1. Ввод двумерного массива (матрицы) 3x5

```
Var a:   array [1..3, 1..5] of Real;
    i,j: Integer;
begin
    for i:=1 to 3 do
      for j:=1 to 5 do
        begin
          Write('Введите a[',i:2,j:2,']=');
          ReadLn(a[i,j]);
        end;
 end.
```

2. Вывод двумерного массива (матрицы) 3x4

```
    for i:=1 to 3 do
      for j:=1 to 4 do
        Writeln('a[',i:2,j:2,']=',a[i,j]);
```

3. Сумма элементов матрицы

```
S:=0;
for i:=1 to 3 do
for j:=1 to 5 do  S:=S+a[i,j];
```

4. Просуммировать элементы главной диагонали матрицы

```
S:=0;
for i:=1 to 3 do S:=S+a[i,i];
```

5. Транспонирование матрицы

```
for i:=1 to 3 do  // Матрица
for j:=1 to 3 do  // д.б. квадратной
     if i<j then
     begin
        r:=a[i,j];
        a[i,j]:=a[j,i];
        a[j,i]:=r;
     end;
```

Тип данных CHAR

Переменная типа `char` представляет собой любой символ и занимает один байт памяти.

Всего символов 256. Каждому символу соответствует цифровой код в интервале 0..255.

Соответствие между символом и его кодом установлено стандартом ANSI. Первая половина символов с кодами 0..127 стандартная, а вторая половина с кодами 128..255 меняется для различных шрифтов.

Например:

коды 48..57 - символы '0'...'9'

коды 97..122 - символы 'a'...'z'

Для работы с кодами переменных типа `char` используются переменные типа `BYTE`, занимающие 1 байт памяти и предназначенные для хранения целых чисел из диапазона от 0 до 255.

Существуют функции преобразования символа в код и наоборот.

Функция `CHR` Число -> Символ Результат:

```
Var a: byte;                          z
begin
   a:=122;
   writeln(chr(a));
end.
```

Функция `ORD` Символ -> Число Результат:

```
Var a: byte;                        122
    b: char;
begin
   b:='z';
   a:=ord(b);
   writeln(a);
end.
```

Функция `UPCASE` преобразует маленькие латинские буквы в большие

```
upcase(a:char): char;
```

ЛЕКЦИЯ 4

СТРОКИ

Строка (`string`) - это последовательность элементов типа `char`.

Описание:

```
Var s: string;
```

Можно описать т.н. короткую строку произвольной длины в пределах 255 символов:

```
Var s: string[N];
```

Эта запись означает, что машина выделяет в памяти N+1 байт.

Например:

```
Var s: string[100];
```

Приведенное описание переменной `s` эквивалентно описанию:

`Var s: array[0..100] of char;` но не идентично.

Количество фактически введенных или существующих символов в переменной типа `string` постоянно находится в её нулевой ячейке. При любых операциях, приводящих к изменению длины строки, число в нулевой ячейке обновляется.

Пример:

```
Var a: string[9];
begin
  a:='самолёт';
end.
```

0	1	2	3	4	5	6	7	8	9	
7	с	а	м	о	л	ё	т			a

`writeln(a[3]);` выведет букву ”м”

`writeln(a[0]);` выведет символ, код которого = 7.

`writeln(ord(a[0]));` выведет цифру 7

В случае массива `CHAR` отслеживания количества символов в нулевой ячейке не происходит, поэтому операции и функции, применимые к данным типа `string`, не подходят к переменным типа `array of char`.

Основные операции для типа string

1. Вывод и ввод строки `writeln(s); readln(s);`

2. Две строки можно сравнивать с помощью условного оператора `if`

```
if a=s then ...
```

Можно применить операцию < или >, тогда строки сравниваются посимвольно. Более длинная строка, в случае совпадения первых символов с более короткой, считается больше.

3. Операция конкатенации - сцепления двух строк.

```
b:='само';
c:='лет';
a:=b+c;
d:=b+'вар';
```

Функции для работы со строками

1. Функция определения длины строки

`k:=lenght(a);` результат - число типа `integer`.

2. Функция `copy(st,from,count)`. Она копирует из строки `st` `count` символов, начиная с `from`.

```
a:='самолет'; //записать "оле" в другую строку
b:=copy(a,4,3); for i:=1 to 4 do write(b,'! ');
```

3. Процедура `delete(st,from,count)`. Из строки `st` удаляет `count` символов, начиная с символа `from`.

```
a:='арбат'; //надо, чтобы осталось "арба"
delete(a,5,1);
```

4. Процедура `insert(subst,st,from)` - вставка подстроки `subst` в строку `st`, начиная с символа `from`.

```
b:='шайтан';
insert(a,b,7);
```

5. Функция `pos` (от position)

```
p:=pos(subst,str);
```

2 входных параметра: строка `subst` и `str`. Функция возвращает целое число - позицию первого символа подстроки `subst` в строке `str`.

```
str:='гидроэлектростанция';
subst:='электро';
p:=pos(subst,str);
```

Результат: `p=6`

Если подстрока не обнаружена, то `p=0`.

Массивы строк

Пример:

```
var s: array [1..4] of string[5]; // массив из 4 строк
                                  // string[5]
one,two,first,second,i,j: integer;
begin
  s[1]:='наука';
  s[2]:='умеет';
  s[3]:='много';
  s[4]:='гитик';
 writeln('In what rows are your cards?');
  readln(one);
  readln(two);
 for i:=1 to 5 do
 for j:=1 to 5 do
     if (s[one,i]=s[two,j]) and (i<>j) then
 begin first:=i; second:=j; end;
 writeln(s[one,first],' ',s[two,second]);
end.
```

ПОДПРОГРАММЫ

Подпрограмма – самостоятельная часть программы, предназначенная для решения конкретной задачи.

В Паскале 2 вида подпрограмм:

1. Функции.
2. Процедуры.

Для того чтобы прервать выполнение подпрограммы используется оператор `exit`.

Функции

Предназначена для выдачи одного единственного результата. Возвращаемое значение передается из функции с помощью оператора `Result` или присваивается внутри функции ее имени. Описание функции начинается со слова `function`.

Структура программы с функцией:

```
<Раздел описаний>
 Function <имя функции>(<описание параметров>):<тип функции>;
 begin
   <тело функции>
   result:=… {или}  <имя функции >:=…
 end;
begin
<тело программы>
<переменная>:=<имя функции>(<список параметров>);
end.
```

Пример подпрограммы-функции:

Программа определения площади треугольника по формуле Герона.

```
Var m,n,k,s: real;
  Function str(a,b,c: real): real;
  Var p: real;
  begin
    p:=(a+b+c)/2;
    str:=sqrt(p*(p-a)*(p-b)*(p-c));
  end;
begin
  writeln('Введите стороны треугольника');
  readln(m,n,k);
  s:=str(m,n,k);     //вызов функции
end.
```

Функция `str` может иметь и такой вид:

```
  Function str(a,b,c: real): real;
  Var p: real;
  begin
    p:=(a+b+c)/2;
    result:=sqrt(p*(p-a)*(p-b)*(p-c));
  end;
```

Формальные и фактические параметры

Параметры, которыми манипулирует функция, называются формальными, а параметры, которые передаются функции при ее вызове из основной программы, называются фактическими. Содержимое фактических параметров при вызове функции копируется формальными параметрами.

В нашем примере `m,n,k` – фактические параметры, `a,b,c` – формальные параметры.

Глобальные и локальные параметры

Глобальные параметры – параметры, определенные в основной программе, доступны в любой точке программы.

Локальные параметры – параметры, определенные в теле подпрограммы, доступны только в самой подпрограмме.

В нашем примере `m,n,k,s` – глобальные параметры, `a,b,c,p` – локальные параметры.

Т.о. обмен информацией между основной программой и подпрограммой может быть реализован через глобальные параметры.

```
Var a,b: integer;         // Глобальные параметры
Function aib: integer;    // Локальных параметров нет
begin
  Result:=a+b;
end;
begin
  readln(a,b);
  writeln('a+b=',aib);
end.
```

Передача параметров в функцию по ссылке

Механизм передачи параметров в функцию по ссылке позволяют сделать доступными в теле функции параметры, объявленные в основной программе. Значения этих параметров могут меняться после окончания работы функции. Для этой цели в заголовок функции перед списком передаваемых ей параметров ставится ключевое слово `Var`.

```
Var a,b,str,hyp: real;
    Function s(var x,y: real): real;
    begin
         Result:=x*y/2;
         x:=sqr(x);
         y:=sqr(y);
    end;
begin
  readln(a,b);
  str:=s(a,b);     // Площадь прямоуг. треугольника.
  hyp:=sqrt(a+b); // Гипотенуза, т.к. из функции s
end.              // вернулись квадраты переменных a и b
```

ЛЕКЦИЯ 5

Процедуры

Отличия процедуры от функции:

1. Вместо ключевого слова `function` - слово `procedure`;
2. У процедуры не определяется тип;
3. Имеются входные и выходные параметры;
4. Для вызова процедуры в основной программе просто указывается ее имя со списком параметров.

Пример.

```
{Процедура, вычисляющая сумму и произведение 3-х чисел}
Var a,b,c,d,e: integer; // глобальные параметры
procedure sumpr(x,y,z: integer; var v,w: integer);
  begin                    // x,y,z - входные параметры
   v:=x+y+z; w:=x*y*z      // v,w - выходные параметры
  end;
begin
 readln(a,b,c);
 sumpr(a,b,c,d,e);      // вызов процедуры
 writeln('sum=',d,' mult=',e);
end.
```

ТИПЫ ДАННЫХ

Тип определяет множество допустимых значений переменной.

В Паскале имеются следующие типы данных: простые типы, структурированные типы, строки и указатели.

ПРОСТЫЕ ТИПЫ

К простым относятся порядковые, вещественные (real) типы и тип дата-время.

Порядковые типы данных

Порядковые типы: `integer`, `char`, `boolean`, а также перечисляемый тип и тип-диапазон.

К ним применимы следующие процедуры и функции:

Процедура `dec(x,y)` - уменьшает значение числа `x` на `y`.

Пример: `x:=5; y:='d'; dec(x,2); dec(y,3);` рез.: `x=3 y='a'`.

Процедура `inc(x,y)` - увеличивает значение числа `x` на `y`.

Процедуры `dec` и `inc` могут иметь только один параметр: `dec(x)`, `inc(x)`. В этом случае значение `x` уменьшается (увеличивается) на единицу. Часто вместо `i:=i+1;` используют `inc(i);`

Функция `pred(x)` - возвращает значение, предшествующее `x`.

Пример: `x:='b'; y:=pred(x);` рез.: `y='a'`

Функция `succ(x)` - возвращает значение, следующее за `x`.

Оператор TYPE

В Паскале существует ключевое слово `TYPE`, которое позволяет использовать существующие в Паскале типы данных, а также описывать свои.

Задание типа осуществляется перед описанием переменных:

```
type <тип>=<допустимые значения>;
var  <переменная>:<тип>;
```

Перечисляемый тип данных

Определяется набором идентификаторов, разделенных запятой и указываемых в круглых скобках.

Раздел описаний:

```
type week=(Monday,Tuesday,Wednesday,Thursday,Friday,
Saturday,Sunday);
var  day: week;
begin
  day:=Sunday;
  if day < Saturday  then  writeln('Week-day');
  if day > Friday  then  writeln('Week end');
end.
```

Значениями переменной `day` могут быть только идентификаторы, перечисленные в типе `week`.

Известный нам логический тип является частным случаем перечисляемого типа:

```
type boolean=(False,True);
```

Перечисляемый тип можно использовать в качестве индекса массива.

Переменные перечисляемого типа не могут вводиться оператором `read` и выводиться оператором `write`.

Тип-диапазон

Тип-диапазон есть подмножество своего базового типа. Задается min и max значениями, например:

```
type digit= '0'..'9'; //базовый тип char
```

Тип-диапазон можно не описывать в разделе `type`, а сразу указывать при описании переменной:

```
var month: 1..12; //базовый тип integer
```

Тип-диапазон имеет все свойства базового типа, но с ограничениями по min и max значениям.

Пример.

```
type WeekEnd=Saturday..Sunday; //базовый тип week
var: w: WeekEnd;
begin
  w:=Saturday;
end.
```

Код переменной типа диапазон равен её порядковому номеру в базовом типе (нумерация начинается с нуля). Так `Ord(w)` вернет значение 5.

Тип дата-время

TDateTime – тип дата-время, предназначен для хранения даты и времени. Вещественное число. В целой части храниться дата, в дробной – время. Дата - количество суток, прошедших с 30.12.1899 г., а время – как часть суток, прошедших с 0 часов. Над данными типа TDateTime определены те же операции, что и над вещественными числами.

Функции для типа дата-время:

Date: TDateTime; - возвращает текущую дату;

Now: TDateTime; - возвращает текущую дату и время;

DateToStr(D: TDateTime): String; – преобразует дату в строку символов в формате короткой даты, который устанавливается в Windows (обычно dd.mm.yyyy);

FormatDateTime(F: String; D: TdateTime): String; – преобразует дату и время в строку символов в формате, указанном в F.

Пример:

```
var D:TDateTime; s1,s2,s3:string;
begin
D:=Now;
s1:=DateToStr(D);
s2:=FormatDateTime('dd.mm.yy hh:mm:ss',D);
s3:=FormatDateTime('dd mmmm yyyy',D);
end;
```

Результат работы программы:

s1=’12.03.2002’

s2=’12.03.02 16:45:07’

s3=’12 Март 2002’

Преобразование типов

В подпрограммы нельзя передавать в явном виде массивы. В этом случае тип массива нужно преобразовывать в простой тип.

Пример.

Неверной будет запись:

```
procedure sum(a:array[1..10] of real);
```

Мы должны переопределить тип переменной a в простой тип:

```
type massive=array[1..10] of real;
Var a: massive;  summa: real;
    i: integer;
      procedure sum(b:massive; var s:real);
      begin
       for i:=1 to 10 do s:=s+b[i];
      end;
    begin
       for i:=1 to 10 do readln(a[i]);
       sum(a,summa);
       writeln('summa=',summa:1:2);
    end.
```

СТРУКТУРИРОВАННЫЕ ТИПЫ

Четыре: массивы, записи, множества, файлы.

Записи

Записи позволяют хранить в одной переменной данные, имеющие различный тип.

Структура объявления типа записи:

```
type <запись> = RECORD
  <поля записи>
END;
```

После объявления типа можно описать переменную типа запись:

```
Var <переменная> : <запись>;
```

Пример:

```
type Info = record
  Name: string[25];
  Income: real;
  Gr: TdateTime;
end;
var person: Info;
    student: array [1..20] of Info;
```

Поля записи

Для обработки доступна как вся запись, так и отдельные ее поля.

При обращении к отдельным полям указывается имя всей записи и имя отдельного поля, разделенные точкой.

```
person.Gr:=StrToDate('11.05.85');
student[1].Name:='Куролесов';
for i:=1 to 20 do
    student[i].Income:=10000;
writeln(student[1].Income:1:2);
```

Оператор WITH...DO

Применяется при совместной обработке нескольких полей записи.

Идентификатор записи указывается однократно в начале фрагмента программы обработки записи. Это позволяет более компактно представлять переменные записей.

Например, вместо:

```
person.Name:='Одуванчиков';
person.Income:=1000;
```

можно записать:

```
WITH person DO
begin
  Name:='Одуванчиков';
  Income:=1000;
end;
```

ЛЕКЦИЯ 6

Язык программирования C++

Идентификаторы (имена)

Идентификаторы – это имена констант, переменных, меток, типов и функций. Идентификатор всегда начинается с буквы или знака подчеркивания, за которыми могут следовать буквы, цифры и знак подчеркивания.

Заглавные и строчные буквы различаются. Таким образом, следующие три идентификатора обозначают разные переменные: index, INDEX, IndEx.

Переменные

Типы переменных:

`float, double` – вещественный тип. Это числа, содержащие целую и дробную части, разделяемые точкой.

`int` – целый тип.

`char` – символ - единичный байт, задается в " ", например "a".

`bool` – логический тип, могут принимать только два значения: 1 - "истина", и 0 - "ложь".

Каждая переменная в программе должна быть описана:

`int a,b,c,M;` - целые переменные `a,b,c,M`

`char C_1;` - символьная переменная `C_1`

Арифметические операции: `+`, `-`, `*`, `/` и `%` - взятие остатка от деления. Оператор `++` увеличивает переменную на 1, оператор `--` уменьшает на 1. Выражение `n++` идентично `n+1`, а выражение `n--` идентично `n-1`.

Оператор присваивания:

```
a=3; b=c=7; M=5-a*(b+B)/c;
```

Оператор `+=`. Пример `a=4; a+=3;` - значение `a` равно `7`.

Аналогично: `-=, *=, /=, %=`. Пример `a=4; a%=3;` (`a` равно `1`).

Константы

```
#define имя значение
```

Пример:

```
#define  NUMBER  15
#define  B  -2.5
```

Математические функции

Для того, чтобы были доступны, необходимо подключить библиотеку математических функций:

```
#include  <math.h>
```

Идентификатор функции	Название функции	Математическое обозначение
`abs(x)`	Абсолютная величина	$\|x\|$
`exp(x)`	Экспонента	e^x

`log(x)`	Натуральный логарифм	ln x
`log10(x)`	Десятичный логарифм	log x
`sqrt(x)`	Квадратный корень	$\sqrt{x}$
`pow(x,y)`	Возведение в степень	x^y
`sin(x) asin(x)`	Синус, арксинус	sin x, arcsin x
`cos(x) acos(x)`	Косинус, арккосинус	cos x, arccos x
`tan(x) atan(x)`	Тангенс, арктангенс	tg x, arctg x
`floor(x)`	Округление вещественного X до меньшего целого	
`ceil(x)`	Округление вещественного X до большего целого	

Функции ввода и вывода

Программа, содержащая функции ввода-вывода, должна обязательно иметь подключение стандартной библиотеки ввода-вывода:

```
#include  <stdio.h>
```

В функциях ввода-вывода определяются типы аргументов:

`%d` - целый тип `int`, `%f` - вещественный тип `float`, `%lf` - вещественный тип `double`, `%c` - символ `char`, `%s` – массив символов `char[]`.

Функция ввода `printf`. Для перевода текущей позиции в начало следующей строки используется эскейп-последовательность `\n`.

Например, вывести на печать `a` и `b`:

```
a= 7; b=8; printf («a=%d\n»,a); printf («b=%d»,b);
```

или `printf("a=%d\nb=%d",a,b);`

Функция вывода `scanf`.

Например, ввести с клавиатуры значение переменной `a`:

```
scanf («%d»,&a);
```

Структура программы. Комментарии

Структура:

```
директивы препроцессора
<тип> main() заголовок функции main
{             начало функции main
              операторы
}             конец программы
```

```
#include  <stdio.h>     //функции printf и scanf
#include  <math.h>      //математические функции
#include  <conio.h>     //функция getch
#define A 3             /* описание констант */
int main()              /* определение функции main */
{                       /* начало функции main */
double x,y;             /*описание переменных*/
printf("x="); scanf("%lf",&x);   /* ввод числа x */
y=pow(x,A);             /* вычисление y */
printf("Y=%1.2lf\n",y);/* вывод числа y */
getch();                //задержка экрана

}                        /* конец прогаммы */
```

Для пояснения текста программы в нее могут включаться комментарии. Комментарий это произвольная последовательность символов, находящаяся:

1. между слэшами со звездочками: `/* ... */`.
2. после двух символов `//`.

Логические выражения

Логические операции:
&& логическое И;
|| логическое ИЛИ;
! отрицание.
Операции сравнения:
< меньше; <= меньше или равно; > больше;
>= больше или равно; == равно; != не равно.
Результатом операции сравнения является 1 (истина) или 0 (ложь).

Например, логическое выражение `(x+1)!=3` - имеет значение 0, если X=2, и значение 1, если значение переменной X отлично от 2.

Логическое выражение `min < X < max` на языке Си:

```
(min<X&&X<max) .
```

Условный оператор if

Оператор `if` предназначен для выбора одного из двух возможных действий в зависимости от некоторого условия.

Структура:

```
if (условие) оператор1 else оператор2
```

Если `условие` истинно выполняется `оператор1`, иначе `оператор2`.

Пример:

```
void main()
{
int a=0, b=4, max;
if(a > b) max=a;
     else max=b;
}
```

В операторе `if` после условия или `else` можно выполнить только один оператор. Чтобы выполнить группу операторов нужно использовать *составной оператор*, т.е. заключить группу операторов в фигурные скобки.

```
if(a > b) { max=a; min=b; }
```

Оператор goto.

Оператор безусловного перехода вызывает передачу управления оператору, которому предшествует соответствующая метка. Метки не могут быть числами.

Структура:

```
GOTO  метка;
метка: оператор
```

Пример:

```
void main()
{
int USA, mozgi;
oops: USA=1; /*number one*/ mozgi=0;
if (mozgi>USA) goto Wow;
    else goto oops;
Wow: printf("Yankee - it sounds gordo!");
}
```

Оператор выбора switch

Оператор позволяет выбрать одно из нескольких возможных продолжений программы.

Структура оператора:

```
switch (ключ_выбора){
     case выбор_1: оператор_1; break;
     case выбор_2: оператор_2; break;
     . . .
     case выбор_k: оператор_k; break;
     default: оператор;}
```

где

выбор_1...k - переменные того же типа, что и ключ.

В последовательности операторов выбор_1...k отыскивается такой, значение которого равно значению ключ_выбора. Соответствующий найденному выбору оператор выполняется. Если переменная, равная значению ключ_выбора, не будет найдена, управление передается оператору, стоящему за словом default.

Пример

```
int student;
scanf("%d",&student);
switch (student) {
case 0: printf("политех\n"); break;
case 1: printf("самолет\n"); break;
case 6: printf("the best\n"); break;
default: printf("придумай сам");
}
```

Выбор ветви case реализуется как переход на метку, поэтому после выполнения одной ветви case программа переходит на следующую ветвь. Оператор break вызывает немедленный выход из цикла.

Оператор while

Структура:

```
while (выражение) оператор
```

Оператор выполняется до тех пор, пока выражение есть истина.

```
i=1;
while (i<4) {
i=i+1;
printf("%d\n",i); }
```

Цикл do-while

Структура:

```
do оператор
while (выражение)
```

Оператор выполняется до тех пор, пока выражение есть истина.

```
int i,l,p;
printf("1. Login\n2. Parol\n3. Exit\n");
do {
     scanf("%d",&i);
     switch (i)
     {
       case 1: scanf("%d",&l); break;
       case 2: scanf("%d",&p); break;
     }
   }
while (i!=3);
```

Оператор for

```
for (i=0; i<3; i++) printf("%d ",i);
```

Переменная `i` (тип `int`) изменяется в цикле от 0 (`i=0;`) до 2 (`i<3;`) с шагом 1 (`i++`).

Результат работы оператора: `0 1 2`.

ЛЕКЦИЯ 7

Указатели

Указатели используются для связи переменных с машинными адресами. При работе с указателями две основные операции: * - доступ к значению переменной через ее адрес, & - операция взятия адреса.

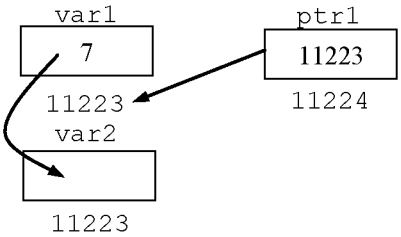
Пример:

```
int var1=7, var2;
int *ptr;

 var2=var1;
```

или

```
 ptr=&var1; // ptr присваивается адрес var1 (11223)
 var2=*ptr; // переменной var2 присваивается значение,
            // на которое указывает ptr (значение var1=7)
```




```
 var1=10;
```

или

```
*ptr=10;     // ptr присваивает значению var1 = 10
```

Пример

```
double a=2,b=3,c=4,d, *p1,*p2,*p3=&c;
p1=&a;
b=*p1;//b=a=2
b=*p3;//b=c=4
p2=p1;
d=*p2;//d=a=2
*p2=*p3;//a=c=4
```

Массивы

Массив - это совокупность данных одного типа.

Описание: `int b[10];` - массив из 10-ти элементов от `b[0]` до `b[9]`.

Двумерный массив: `int a[2][3];` - массив из 6-ти элементов от `a[0][0]` до `a[1][2]`.

Имя массива является указателем на массив.

Пример: Ввести значения двумерного массива и вывести на печать значения главной диагонали:

```
int i,j; float a[3][3],b[3];
for (i=0; i<3; i++)
for (j=0; j<3; j++)
{printf("a[%d][%d]=",i,j); scanf("%f",&a[i][j]);
if(i==j) b[i]=a[i][j];}
for (i=0; i<3; i++) printf("b[%d]=%3.2f\n",i,b[i]);
```

Другой вариант вывода на печать значений главной диагонали:

```
for (i=0,j=0; i<3&&j<3; i++,j++)
printf("a[%d][%d]=%3.2f\n",i,j,a[i][j]);
```

Перечисления

Описание:

```
enum целый тип {нумераторы};
```

Пример:

```
enum flag {white, blue, red};
// переменная типа flag м.иметь 3 значения
enum flag color; // переменная color имеет тип flag;
color=red; // переменной color присвоено значение red;
```

Каждому нумератору назначается целое значение. Первое значение – 0, второе – 1 и т.д.

`printf("%d",red);` - напечатает 2.

Функции

Программирование на языке С основано на понятии функции. Функция – самостоятельная единица программы, предназначенная для решения конкретной задачи. Предназначена для выдачи одного единственного результата. Возвращаемое значение передается из функции с помощью оператора `return`.

Структура:

```
тип_функции имя_функции (тип_1 арг_1, тип_2 арг_2,…)
{
Объявление локальных данных
Выполнение действий
return возвращаемое_значение
}
main() {…
```

`тип_функции` – это тип возвращаемого значения. Если функция не возвращает никакого значения, следует указать тип `void`. По умолчанию тип функции `int`. Далее следует имя функции. Затем в круглых скобках перечисляются аргументы функции с указанием типа каждого. Аргументы функции могут отсутствовать.

Если количество аргументов, передаваемых функции заранее неизвестно, можно использовать многоточие:

```
void Fun(int i,…);
```

Такая запись означает, что за аргументом `i` могут следовать, а могут и не следовать другие аргументы. Необходимо самому отслеживать действительное количество переданных аргументов.

Функция может возвращать только ОДНО значение. Получение большего количества значений можно сделать только через УКАЗАТЕЛИ (опять они!), передаваемые в качестве аргументов.

Пример: Функция возвращает значение – сумму `a` и `b`, а также через параметры передает значения разности и произведения `a` и `b`:

```
int fun( int *a, int *b, int *raz, double *pr)
  {int sum;
  *raz=*a-*b; *pr=*a**b;
   sum=*a+*b;
  return sum;}
void  main()
{
int a,b,raz; double pr;
int sum;
 a=4; b=2;
 sum=fun(&a,&b,&raz,&pr); //вызов функции
 printf("sum=%d\n",sum);
 printf("raz=%d\npr=%f\n",raz,pr);
}
double sqr(double x)
{
   return x*x;
}
int fun( int *a, int *b, int *raz)
  {
  *raz=*a*3-*b;
  return *a**b;
  }

int main()
{
int x=10,y=9,r=5,pr,f; double a;
a=sqr(2);
pr=fun(&x,&y,&r);
printf("pr=%d\n",pr);
printf("r=%d\n",r);
printf("Press any key");
getch();
}
```

Структуры

Структура – набор переменных различных типов, образующих единый объект.

Описание структуры:

```
struct тип_структуры {типы переменных};
тип_структуры имя_структуры;
```

Пример:

```
struct stype {
int i;
char c;
double f;
};
stype sname;
```

или с инициализацией:

```
stype sname = {100,"C",3.14};
```

Можно одновременно задать тип и имя:

```
struct stype {…} sname;
```

Обращаться к элементам структуры можно, указав имя структурной переменной и имя элемента, разделенные точкой: `sname.f;`

```
pi=sname.f;
sname.i=9;
```

Для передачи структуры в функцию необходимо создать указатель на структуру:

```
stype *sptr;
```

и при использовании указателей пользоваться операцией `->`:

```
sptr->f;
```

Пример передачи структуры в функцию:

```
struct stype{
int i;
double f;
} sname={12,3.14};
stype *sptr;

void main()
{
int ff(stype *sptr);
int j; double pi;
sptr=&sname;
j=ff(sptr);
pi=sname.f;
printf("%d %f\n",j,pi);
}

int ff(stype *sptr)
{
int k;
sptr->i=5;
```

```
sptr->f=3.62;
k=sptr->i;
return k;
}
```

Результат работы программы: `5 3.62`

Описание массива структур:

```
struct stype sname[3];
```

Объединения

Аналогичны структурам – набор переменных различных типов, образующих единый объект. Вместо ключевого слова `struct` - слово `union`.

```
union utype {…} uname;
```

Объявление typedef

Объявление типа `typedef` позволяет приписать новое имя существующему типу данных. Пример:

Объявление	Пример применения	Значение
`typedef int SHORT;`	`SHORT a,b;`	`int a,b;`
`typedef struct {int n; char c[30];} ID;`	`ID data;`	`struct {int nCode; char cVal[30];} data;`

Дополнительные операции

1. `sizeof (переменная)` **или** `sizeof (тип)`

Позволяет получить размер объекта в байтах

```
int a;
a=1; printf("%d\n",sizeof (a));
// результат 4 - размер переменной типа int 4 байта
printf("%d\n",sizeof (double));
// результат 8 - размер типа double 8 байтов
```

2. `(тип) значение`

Преобразует значение в тип, указанный в круглых скобках

```
(float)4;
```

- целое число 4 преобразуется в число с плавающей точкой 4.0.

3. `?:`

Операция условия. Структура:

```
выражение1 ? выражение2 : выражение3
```

Результат равен значению `выражение2` если `выражение1` истинно и значению `выражение3`, в противном случае.

Пример: Функция `max` находит наибольшее из 2-х чисел.

```
int max(int a, int b) {return (a>b)?a:b;}
```

Файлы

Для работы с файлами существует специальный тип FILE.

Пример

```
include <stdio.h>
void main()
{
   FILE *f; // указатель на файл
   int i = 100;
   char c='a';
   double p = 1.234;
f = fopen("d:/myfile.dan", "w");
// создали файл с именем myfile.dan
// и установили на файл указатель f
```

Значок "w", означает, что файл создается (открывается) по записи. При создании необходимо указать "w". В дальнейшем, при открытии уже существующего файла, для добавления записей можно указать "a", а только для чтения записей – "r".

```
fprintf(f, "%d %c %f", i, c, p);      //запись в файл
fscanf(f, "%d %c %f", &i, &c, &p); //считывание из файла
printf("%d %c %f", i,c,p);
//вывели на экран переменные, считанные из файла
fclose(f); //закрыли файл myfile.dan
}
```

Заголовочные файлы

Файлы с расширением `.h` (хэдер-файлы). Могут содержать: определения констант и типов, описания данных и функций.

Подключение h-файла: `#include "заголовочный файл"`

В файле `stepen.h` содержится:

```
int i; double f;
double step(double *f,  int *i);
```

В файле `stepen.cpp` содержится:

```
#include <stdio.h>
#include "stepen.h"
void main()
  {  scanf("%le %d",&f,&i);
     printf("%f\n" , step(&f,&i));
  }
double step(double *f,  int *i)
  {   int j;   double rez=1;
      if (*i<1) *i=1;
      for (j=0; j<*i; j++) rez*=*f;
      return rez;
  }
```