# Язык Паскаль(Pascal)

# Урок 1 - Введение в язык Паскаль(Pascal)

|  |  |
| --- | --- |
| |  | | --- | |  |   Для того чтобы войти в среду программирвоания на Паскаль, необходимо:  1. Скачать эту программу. 2. Найти файл BP.EXE и запустить его. На экране появится окошко следующего вида:  pascal  В меню Файл вы можете сохранить свою программу, открыть существующую, а также создать новую. Комбинация клавиш Alt+Enter переведет программу в полноэкранный режим программирование на Pascal.  Итак, приступим за дело. Давайте напишем программу, которая делает следующее:**Выводит сообщение о вводе 2-х целочисленных чисел. Делит одно число на другое и выводит полученный результат.** Алгоритм работы программы: ОБЪЯВЛЕНИЕ ПЕРЕМЕННЫХ  НАЧАЛО РАБОТЫ ПРОГРАММЫ  ВВОД ЗНАЧЕНИЯ ПЕРЕМЕННЫХ  ДЕЛЕНИЕ ОДНОГО ЧИСЛА НА ДРУГОЕ  ВЫВОД ПОЛУЧЕННОГО РЕЗУЛЬТАТ  КОНЕЦ  А теперь напишем это на языке ПАСКАЛЬ:  program lesson1; {здесь описывается заголовок программы}  var a,b:integer; {переменные в которые мы будем заносить числа и они целочисленные} c:real; {переменнная в которую заносится значение деления 2-х чисел} BEGIN  writeln('Vvedite chisla A, B'); {Эта строка выводит сообщение на экран} ReadLn(a,b); {Здесь мы заносим значение 2-х чисел} c:=a/b; {деление числа a на b и результат присваивается переменнной c} writeln('C = ', c); {вывод значения C} readln; {задержка экрана пока не нажата клавиша Enter} end.{конец работы программы}  **Грамматика**  Как вы заметили нужно после каждой строки ставить точку с запятой. Если вы где-то забудете поставить точку с запятой, то компилятор не пропустит это и выведет синтаксическую ошибку.  Объявление переменных имеет вид: ( var a,b:integer; ). Var - означает то, что мы объявляем переменные. a, b - это имена переменных с которыми мы будем работать в программе. integer и real - это типы переменных, в частности целочисленный и дробный тип. Вывод сообщения на экран имеет следующий вид (writeln('Сообщение');). Writeln - это оператор, посредством которого мы сможем вывести что-то на экран монитора. То есть выводится то, что стоит внутри скобков и отделено одинарными кавычками. Если же мы хотим вывести какое-то сообщение и значение какой-то переменнной, то нам нужно сообщение внести в одинарные кавычки, потом поставить запятую и написать переменную, результат которой нужно вывести на экран (writeln('C = ', c); ).  **Результат работы программы:**  **Результат работы программы на Pascal** |

**Урок 2 - Работа в среде Borland Pascal на примере программ линейной структуры**

|  |  |
| --- | --- |
| |  | | --- | |  |   Алгоритм линейной структуры - это алгоритм, в котором блоки выполняются в указанном порядке, последовательно друг за другом. Программа линейной структуры реализует соответствующий линейный алгоритм.  Чаще всего линейные алгоритмы используются для программирования вычислений по формулам. В этом случае удобно использовать набор встроенных функций Pascal: Sin (x) – синус х; cos(x) – косинус х; tan(x)- тангенс х ; ln (x) – натуральный логарифм х ;  exp (x) -показательная функция e^x ; sqr (x) - квадрат  x;   sqrt(x)- корень квадратный от х;  abs(x)- модуль x. Для тригонометрических функций аргумент х  измеряется в радианах. Гиперболические функции записываются с применением экспоненты, например: Sh (x)=(exp (x)-exp (-x)) /2  **Пример программы линейной структуры**  Требуется вычислить высоты треугольника по трем заданным сторонам a, b, c. Для вычисления высот воспользуемся двумя выражениями  площади треугольника - по трем сторонам (формула Герона) и как половина произведения стороны на соответствующую высоту: S=1/2\*ah     ;       S=http://life-prog.ru/pascal/pas1_clip_image002.gif; здесь *p* - полупериметр*p=(a+b+c)/2*. Соответствующая этому алгоритму программа вычисления  высот треугольника имеет вид:  program linejn  ; {программа вычисления высот треугольника } var a, b, c, p, ha, hb, hc, t : real; {описание используемых переменных } begin writeln (‘Введите стороны тр-ка a,b,c’); readln (a,b,c); p:=(a+b+c)/2.; {вычисление полупериметра треугольника } t:=2.\*sqrt (p\*(p-a)\*(p-b)\*(p-c)); ha:=t/a; hb:=t/b; hc:=t/c; {вычисление высот треугольника } writeln (‘Высоты треугольника: ha=‘,ha:8:2,’ hb=‘, hb:8:2,’  hc=‘,hc:8:2); readln; end.  Дадим краткое пояснение. Комментарий в программе на языке Паскаль служит для разъяснения действий основных блоков текста и программы в целом. Комментарий заключается в фигурные скобки {} или в пары символов (\*   \*) и не влияет на компиляцию программы. Это - средство самодокументирования программ.  Процедуры read , write (writeln) служат для  ввода с клавиатуры и вывода на экран соответственно. Символы ln в процедуре writeln служат для перевода на новую строку. В программе описаны переменные a, b, c, ha, hb, hc, t вещественного типа (real). В данной программе из математических функций используется sqrt(x) - квадратный корень из x. При выводе можно включить общую ширину поля вывода и число позиций после десятичной точки. В данном случае высоты треугольника выводятся в поле из 8 позиций каждая, из которых одна отводится на десятичную точку, а две - на цифры после десятичной точки. Остальные 5 позиций отводятся на знак и целую часть числа. Вывод текста ‘  hb=’  содержит по крайней мере один пробел после  первого апострофа для отделения от предыдущего значения. Завершает текст  программы  оператор end. Обратим внимание, что в программе вычисляется промежуточная переменная t (удвоенная площадь треугольника), через которую вычисляются высоты. Вычисление такого рода промежуточных переменных для сокращения дальнейших вычислений - характерный признак программ на языках высокого уровня, в том числе Паскаль.  **Общие сведения  о  интегрированной Среде Borland Pascal**  Экраны интегрированных Сред  Turbo и Borland  Pascal подобны и содержат главное меню, окно редактирования, строку состояния. Для входа в главное меню следует нажать F10.  Главное меню содержит пункты:  File(Файл), Edit(Редактирование), Search(поиск), Run(Исполнить), Compile(Компиляция), Debug(Отладка), Tools(Инструменты), Options(Параметры), Window(Окно), Help(Помощь). Из этих пунктов для нас важнейшими будут File, Edit, Compile, Run, Help. В меню File имеются пункты Open (Открыть), при помощи которого открывается существующий программный файл, New (Создать) - создается новый файл, Save (Сохранить) - сохраняется редактируемый файл, Save as (Сохранить как) - файл сохраняется под новым именем, Change dir (Изменить каталог) - задаются входные и выходные каталоги. Последним пунктом нужно пользоваться, чтобы задать ваш каталог, где хранятся тексты программ. Пункт Quit(Выход) служит для выхода из Среды. В меню File приведены также “горячие “ клавиши (F3), (F2), (ALT+X) для открытия, сохранения файла и для выхода из Среды соответственно. В меню Edit имеются пункты Cut (Вырезать), Copy (Копировать), Paste (Вставить), Clear (Очистить), которые служат для вырезания, копирования, вставки и очистки выделенного фрагмента текста, как в любом текстовом редакторе. Выделение фрагмента текста производится установкой курсора в начале фрагмента, отметкой начала блока (CTRL+K+B), передвижением курсора в конец блока и отметкой конца блока (CTRL+K+K). Вырезание, копирование, вставка фрагмента осуществляются с помощью пунктов меню Edit или с помощью горячих клавиш (Shift+Del), (CTRL+INS), (Shift+INS) соответственно. Меню Compile содержит пункт Compile(Компиляция), горячие клавиши ALT+F9. Меню Run содержит пункт Run(Исполнить),  горячие клавиши CTRL+F9. Меню Help позволяет обратиться к подсказке (справочной системе), содержит пункты Contents (Содержание), Index (Предметный указатель) - горячие клавиши (Shift+F1), Topic search (Тематический поиск) - горячие клавиши (CTRL+F1). Подсказкой следует пользоваться для быстрого освоения Интегрированной Среды. Нажатие клавиши F1 приводит к открытию окна подсказки. Для получения справки о конкретной функции следует установить курсор на любой символ заголовка функции и нажать (CTRL+F1). |

**Урок 3 - Условный оператор IF. Оператор варианта Case. Программы разветвляющей структуры**

|  |  |
| --- | --- |
| |  | | --- | |  |   Алгоритм разветвляющейся структуры - это алгоритм, в котором вычислительный процесс осуществляется либо по одной, либо по другой ветви, в зависимости от выполнения некоторого условия. Программа разветвляющейся структуры реализует такой алгоритм.  **Условный оператор IF**  В программе разветвляющейся структуры имеется один или несколько условных операторов. Условный оператор в языке Паскаль имеет формат: if  <условие> then <оператор1> else <оператор2>;  (полная форма)   или if <условие> then <оператор1>;   (сокращенная форма). Точка с запятой перед зарезервированным словом else не ставится.  При  выполнении условия исполняется оператор1, при невыполнении - оператор2 (при полной форме условного оператора). Для неполной формы условного оператора при выполнении условия исполняется оператор1, в противном случае оператор1 пропускается и исполняется оператор, следующий за условным оператором. Оператор1 и оператор2 могут представлять простые операторы (один оператор), в этом случае они не заключаются в операторные скобки. Если же оператор1 и/или оператор2 представляют составной оператор (несколько операторов), то их нужно заключить в операторные скобки begin …  end.  В качестве примера приведем программу вычисления наибольшего из значений функции y1=x\*x+1, y2=7-x\*x или y=x+1 для любого х.  program razvetvl ; {программа разветвляющейся структуры} var   x , y1 , y2 , y3 ,  max :  real; begin writeln (‘Введите x’);      readln (x); y1:=x\*x+1;      y2:=7-x\*x ;      y3:=x+1; if  y1> y2   then   max:=y1 else max:=y2; if   y3 > max   then         max:=y3; writeln (‘ y1=‘,y1: 6:2’  y2=‘,y2: 6:2, ’  y3=‘,y3: 6:2,); writeln (‘при x=‘,x: 6:2,’  наибольшее  значение функций=‘,max: 6:2); readln; end.  **Оператор варианта Case**  Если количество разветвлений программы больше двух, то используется оператор вариантов Case, который является более общим случаем условного оператора.  Оператор Case выполняет один из нескольких операторов в зависимости от значения селектора. Если селектор не совпадает ни с одним из значений, то выполняется оператор после else  или следующий после case, если else отсутствует. Селектор может быть целого, символьного, перечислимого, интервального типа или выражением.  Например, определим время года по введенному номеру месяца.  Program  case1; {  множественный выбор} Label m;  Var    num: integer; begin m:  writeln (‘Введите номер месяца’); readln (num); write  (‘Время года: ‘, num); case   num  of { выбор значений селектора num} 1, 2, 12:   writeln (‘Зима’); 3..5:       writeln  (‘Весна’); 6..8:       writeln (‘Лето’); 9..11:      writeln (‘Осень’); else     begin writeln (‘нет с таким номером месяца!’); goto m  ; end;  end; {завершение оператора case} readln; end .  В приведенном примере программы при вводе номера месяца от 1 до 12 на экране печатается соответствующее время года и выполнение программы заканчивается. Если же номер месяца превышает 12 или меньше 0, то выводится сообщение о неверном вводе месяца, для чего служит зарезервированное слово else и выполняется переход на ввод нового значения месяца с помощью метки m.  В данной программе num – селектор. |

**Урок 4 - Циклы for, while, repeat ... until в языке Паскаль**

|  |  |
| --- | --- |
| |  | | --- | |  |   Алгоритм циклической структуры - это алгоритм, в котором происходит многократное повторение одного и того же участка программы. Такие повторяемые участки вычислительного процесса называются циклами. Программа циклической структуры содержит один или несколько циклов. Различают детерминированные циклы с заранее известным числом повторений и итерационные циклы, в которых число повторений заранее неизвестно. Изменяющаяся в цикле переменная называется параметром цикла.  Для организации цикла необходимо выполнить следующие действия: 1) задать перед циклом начальное значение параметра цикла; 2) изменять параметр перед каждым новым повторением цикла; 3) проверять условие повторения цикла; 4) управлять циклом, т.е. переходить к его началу, если он не закончен, или выходить из него по окончании.  В языке Паскаль существует 3 вида циклов:  1) цикл с параметром или цикл типа **for**,  2) цикл с предусловием или цикл типа **while**,  3) цикл с постусловием или цикл типа **repeat ... until**.  В цикле типа for число повторений известно заранее, в циклах типа while и repeat ... until число повторений цикла заранее неизвестно, производится проверка условия повторения цикла: в цикле типа while - перед циклом, в цикле типа repeat ... until - после его окончания. В циклах типов for и while повторяющяся часть (тело цикла) состоит из одного оператора, если требуется выполнить в цикле несколько операторов, они заключаются в операторные скобки begin ... end, образуя составной оператор. В цикле типа repeat ... until тело цикла помещается между зарезервированными словами языка (лексемами) repeat и until, операторные скобки не требуются, в названии цикла его тело условно обозначается тремя точками. С помощью цикла типа for удобно находить суммы, произведения, искать максимальные и минимальные значения и т.п. При нахождении суммы некоторой переменной, например S присваивается значение 0, затем в цикле к этой переменной прибавляется соответствующий член заданной последовательности. При нахождении произведения переменной присваивается значение 1, затем в цикле эта переменная умножается на общий член последовательности.  **Пример цикла типа for**  Вычисление n чисел Фибоначчи: F1=1;               F2=1;…;          Fn=Fn-1+Fn-2 ,  например F3=F2+F1=1 + 1 = 2; F4 = 2 + 1 = 3 и т.д.  program fib;  {Нахождение чисел Фибоначчи} var  x, y, z, i, n : integer; begin writeln  (‘Введите n’);   read (n); x:=1; y:=0; for   i:=1 to n do begin z:=x ;  x:=x+y ;  y:=z ; write ('   ', x ) ; end; readln; end.  **Пример цикла типа while**  Cоставление таблицы функции  y = a3/(a2+x2) для х, принадлежащих отрезку [-1; 1] с шагом 0.1. Так как параметр цикла     типа for должен быть целочисленным, удобнее использовать цикл while, в котором значение х можно изменять  при каждом шаге на Dх = 0.1  program cycl\_while; uses crt;   {вызов модуля Crt для управления режимом экрана}   var  a, x, y : real ;   i :  integer; begin clrscr ; {процедура очистки экрана  из модуля Crt}   writeln ('Введите а  ' );    readln (a);    x:= -1.;    writeln (‘     x   ‘,   ‘      y      ’); while   do   x<1.05   begin    y:= sqr(a)\*a/ (a\*a+x\*x);    writeln  (x:6:2, y:8:4);    x:=x+0.1 end; readln;  end .  Условие х<1.05 соответствует каждому значению х плюс половина шага .  **Пример цикла типа repeat ... until**  Определить  число n, при котором сумма квадратов натурального ряда чисел от 1 до n не превысит величину K, введенную с клавиатуры. Т.е.  S >= K,   где S= http://life-prog.ru/pascal/pas4_clip_image002_0000.gif  program sum\_sq;  {Сумма квадратов натурального ряда } uses crt; var                  k, s, n : integer; begin clrscr ;      writeln( 'Введите K' );   readln (k);    s:=0;    n:=1; repeat    s :=s+n\*n;    n := n+1; until  s > k; writeln  ('N= ', n : 3,  '   s= ' ,  s : 5 ); readln; end.  Цикл повторяется до тех пор, пока условие записанное после ключевого слова until, будет ложным (не выполняется). Как только это условие выполнится, происходит выход из цикла. После окончания цикла производится печать результата (оператор writeln). Отметим, что цикл с предусловием (типа while)  может не выполниться ни разу, цикл с постусловием repeat ... until выполнится по крайней мере 1 раз. Когда число повторений цикла неизвестно заранее, применяются циклы с предусловием или с остусловием . Когда число повторений цикла известно заранее, как правило, применяется цикл типа for. Но любой цикл типа for можно заменить циклом с предусловием или постусловием. |

**Урок 5 - Обработка массива на языке pascal(одномерный, двумерный)**

|  |  |
| --- | --- |
| |  | | --- | |  |   Массивы - структурированный тип данных с элементами одного типа. Количество элементов определяет размер массива. Например, массив составляют заработные платы сотрудников подразделения предприятия, здесь число элементов равно числу сотрудников; массив образуют набор чисел, их количество равно числу элементов массива. Номер элемента массива называется его индексом. Массив может иметь не один, а большее число индексов. Число индексов называется размерностью массива, например, массив с двумя индексами называется двумерным массивом. Таким двумерным массивом является, в частности, матрица системы  n линейных алгебраических уравнений с n неизвестными. В то же время столбец свободных членов этой системы является одномерным массивом.  Массив должен быть описан либо в разделе переменных VAR, либо в разделе типов TYPE  следующим образом:  TYPE <имя\_типа> = ARRAY [ti,t2....,tn] OF <тип\_элементов\_массива>; VAR <имя\_массива>  :  <имя\_типа> ; или VAR<имя\_массива>: ARRAY [ti,t2,...,tn]OF<тип\_элементов\_массива>; где      tl,...,tn - тип индекса (перечислимый или интервальный); Например: 1) VAR           a:  ARRAY   [1..5]   OF    real; Описан массив  а  действительных чисел, который состоит из 5 элементов. 2) TYPE   t =  ARRAY[1..3,1..4] OF integer; VAR   b  :  t; Описана матрица  b из  3 строк и 4 столбцов, элементы которой являются целыми числами. 3) TYPE tl  =  ARRAY  [1..4]   OF   integer; t =  ARRAY  [1..5]   OF    t1; VAR   d : t; к : t1;  В начале описан тип одной строки t1, затем тип всей матрицы t через тип строки t1. В разделе переменных указано ,что d-двумерный массив размером (5,4), a k - одномерный массив k(4).  **Пример обработки одномерного массива**  Дан одномерный массив MAS(12) из вещественных чисел. Найти наибольший элемент  массива и его индекс.  program  pr4\_1 ; const  n = 12;  (\*константа n определяет размер массива в описании\*) type m  = array [1..n] of   real ;      (\*  m – тип массива mas  \*) var    mas : m ; i , num : integer;    max : real; begin  for i:=1 to n do       (\*ввод элементов массива mas по 1 в строке\*) begin writeln(‘введите элемент массива’, i) ;   read  (mas[i]); end; num :=1;  max := mas [1]; for  i: =2 to n  do if  mas [ i ] > max  then     begin max := mas [i ]; num := i ; end; writeln;       (\* вывод   массива     в строку\*) for i:=1 to n do    write (mas [ i ]:5:1 );   writeln;   writeln (‘максимальный элемент = ‘, max:4:1,‘   его индекс=’ ,num ); readln; end.  Переменная max сравнивается с элементами массива, и если элемент массива больше max, то переменной max присваивается значение элемента массива, а переменной num - индекс этого элемента. По окончании цикла переменная max  будет иметь значение, равное максимальному элементу массива, а переменная num - значение индекса этого элемента.  **Пример обработки многомерного массива**  Как правило, при обработке многомерных массивов используются вложенные циклы, т.е. цикл по столбцам располагается внутри цикла по строкам.  Дана матрица A(3,4), и вектор B (4), состоящие из целых чисел . Умножить матрицу  А на вектор В .  program  pr4-2 ; const m=3;   n=4; var a  : array [ 1 .. m, 1 .. n ] of    integer; (\* описание матрицы \*) b  : array [ 1 .. n ] of   integer;                      (\* описание вектора \*) c  : array [ 1 .. m ]  of  integer;                      ( \* описание  С  \*) i,  j: integer; begin for i:=1 to m do                  (\* ввод матрицы  \*) begin  writeln (‘введите элементы ‘, i , ‘-той строки’); for j:=1 to n do read (a [i, j] ); writeln;  end; writeln (‘введите элементы вектора’); for j:=1 to n  do                             (\* ввод  вектора \*) read (b[ j]);   writeln; for i:=1 to m do begin c [ i ]:=0;        for j:=1 to n do       c[i] := c[ i ]+ a[i , j]\* b[j]; end; for i:=1 to m do      (\*форматный вывод матрицы  \*) begin for j:=1 to n do write (a [i, j]: 4 );  writeln;  end; for j:=1 to n do  write (b [ j ] :4);    (\* вывод  массива B \*) writeln ; for i:=1 to m do  write (c [ i ] :4);   (\* вывод  массива С  \*) readln; end.  В программе элементы матрицы вводятся по строкам по одному с подтверждением клавишей Enter. А выводятся в общепринятом виде: каждая строка матрицы  с новой строки экрана (цикл i по строкам  внешний, а цикл j – внутренний). |

**Урок 6 - Символьные данные в языке Паскаль**

|  |  |
| --- | --- |
| |  | | --- | |  |   Наряду с числовой информацией в Паскале используется алфавитно-цифровая или символьная информация, которая  включает в себя заглавные, строчные буквы, цифры от 0 до 9 и  вспомогательные  символы. Для описания символьных переменных используется тип данных CHAR или STRING.  **Тип  данных CHAR**  Каждая переменная символьного типа может принимать значение только одного символа. Все символы упорядочены в соответствии с принятым в ЭВМ коде (например ASCII). При этом порядковый но­мер символов называется кодом (например, код латинского символа 'А '  равен 65; символа '3' равен 51). Для символьных данных не определены никакие арифметические операции, но они могут сравниваться по своим кодам, участвовать в чтении, печати, операторах присваивания. Существуют две стандартные функции преобразования : 1) ORD (C) принимает значение кода символа С; 2) значение функции CHR(I) является символ с кодом Например:ORD('А ')=65 CHR(ORD(C))=C   ; CHR (65) = A   ; Строка - это последовательность символов. Строку можно представить как массив, элементы которого имеют тип CHАR. Например: BUK: array[1..17] of char; Массив BUK-массив символов, который содержит 17 символов. Если символов меньше, то строка дополняется пробелами справа. В про­тивном случае возникает ошибка несоответствия типов. Так как массивы символов являются обычными массивами с элементами типа CHAR, они обладают всеми свойствами массивов.  **Пример**: Из набора 10 любых символов напечатать только заглавные английские буквы и их коды.  program lr1; type sl =array [1..10] of char; var   s: sl; {описание массива символов} i: integer; begin  writeln ('введите 10  символов'): for i:=1 to 10 do  readln (s[i]);          {ввод массива} for i:=1 to 10 do if (s[i]>=’A’) and (s[i]<= ‘Z’) then  writeln (‘Символ :’, s[i], ‘ его код =’, ord (s[i]); readln; end.  **Тип данных STRING**  В Турбо Паскале предусмотрен тип данных STRING. Переменная типа STRING может принимать значения переменной длины. Максимально возможная длина переменной 255 символов. Например: str: STRING[200]; ow: STRING[10]; В скобках указывается максимальная длина для данной переменной. Для ввода значений типа STRING необходимо использовать READLN, а не READ. За один раз может быть введена только одна строка. Две строки можно сравнивать, используя операции отношения (сначала сравниваются самые левые символы, если они равны, то сравниваются следующие). Для работы с переменными типа STRING используют следующие стандартные процедуры и функции:  1) Функция LENGTH C:=LENGTH(str); Переменной С будет присвоено целое значение, показывающее количествo символов в строковой переменной str .  2) Функция СОNCAТ - сцепление строк в порядке их перечисления. str:=CONCAT(st1,st2,...,stN);str-переменная типа STRING, состоящая из строк st1,...,stN.  3) Функция POS P:=POS (st1, st2); Р-целое число, показывающее номер позиции, с которой начина­ется строка st1 в строкe st2.  4) Функция COPY  S1:=COPY( str, I, J); Sl-символьная подстрока, выделенная из строки str с позиции I, длиной J символов.  5) Процедура DELETE(Str, I, J);  Из строки str удаляется J символов, начиная с I позиции.  6) Процедура INSERT(Str1, Str2, I); Строка Str1 вставляется с I позиции в строку Str2.  7) Процедура STR (V, S1); Числовое значение переменной V преобразуется в строку символов и записывается в строку S1.  8) Процедура VAL (S1, V, C);  Строковое выражение S1 преобразуется в величину целочислен­ного или вещественного типа и записывается в переменной V . Если при  этом ошибок не обнаруживается, то С будет равно 0 . В противном  случае значение С будет равно номеру позиции пер­вого ошибочного символа и V будет неопределено. Строка S1 не должна содержать незначащих пробелов, перемен­ная V может быть целой или вещественной, а переменная С - только целой .  **Пример**: Подсчитать количество слов во введенной с клавиатуры строке.  program lr2; var s: string[30]; kol, i, n: integer; begin writeln ('введите строку'); readln (s); kol:=0;          {счетчик количества слов} n:= length(s);            {определяем длину введенного текста} s:= concat('  ',s); {добавляем пробел к первому слову} for i:=1 to n do if (copy (s,i,1)='  ') and (copy (s,i+1,1)<>' ')  then  kol := kol+1;{подсчет количества слов} writeln (s,'  количество слов= ',  kol); readln; end. |

**Урок 7 — Процедуры и функции в Паскале. Рекурсия**

|  |  |
| --- | --- |
| |  | | --- | |  |   Часто в задаче требуется повторить определенную последовательность операторов в разных частях программы. Для того, чтобы описывать эту последовательность один раз, а применять многократно, в языках программирования применяются подпрограммы.  Подпрограмма - автономная часть программы, выполняющая определенный алгоритм и допускающая обращение к ней из различных частей общей программы. Использование подпрограмм позволяет реализовать один из самых современных методов программирования - структурное программирование.  В языке Паскаль существует два вида подпрограмм: **процедура** (PROCEDURE ) и**функция** ( FUNCTION ). **Процедуры и функции в Паскале** объявляются в разделе описания за разделом переменных. В данном уроке приведены примеры и задачи использования процедуры и функций, а также использование рекурсии в языке Паскаль. Параметры, записываемые в обращении к подпрограммам, называются фактическими;  параметры, указанные в описании подпрограмм - формальными. Фактические параметры должны соответствовать формальным по количеству, порядку следования и типу. Параметры, объявленные в основной (главной) программе, действуют в любой подпрограмме  и называются глобальными. Параметры, объявленные в подпрограмме, действуют только в этой подпрограмме и называются локальными.    **Процедуры**  Процедуры используются в случаях, когда в подпрограмме необходимо получить несколько результатов. В языке Паскаль существует два вида  процедур: **процедуры с параметрами и без параметров**. Обращение к процедуре осуществляется по имени процедуры, за которым могут быть указаны фактические параметры. Все формальные параметры являются локальными для данной процедуры и глобальными для каждой процедуры в ней. При вызове процедуры  устанавливаетcя взаимно однозначное соответствие между фактическими и формальными параметрами, затем управление передается процедуре. После выполнения процедуры управление передается следующему, после вызова процедуры, оператору вызывающей программы.  **Пример 1.** Процедура без параметров, которая печатает строку из 60 звездочек.  procedure  pr;     var      i :  integer ;     begin for i :=1 to 60 do write (‘ \* ');    writeln;     end.  **Пример 2**. Процедура c параметрами.  Даны 3 различных массива целых чисел (размер каждого не превышает 15). В каждом массиве найти сумму элементов и среднеарифметическое значение.  program proc; var i , n , sum: integer; sr : real; procedure work (r:integer; var s:integer; var s1:real); {процедура work} var mas : array [1..15] of integer ; { объявление массива мas} j : integer; begin s:=0; for j:=1 to r do begin {ввод элементов массива mas} write(' Vvedite element - ', j,': ') ; read (mas[j]); s:=s+mas [j]; end; s1:=s/r; end; begin { главная программа} for i:=1 to 3 do begin write ('Vvedite razmer ',i, ' masiva: '); readln(n); work (n, sum, sr); {вызов процедуры work} writeln ('Summa elementov = ',sum); writeln ('Srednearifmeticheskoe = ',sr:4:1); end; readln; end.  **Результат работы программы:**  http://life-prog.ru/pascal/res7.gif  В программе трижды вызывается процедура work,  в которой формальные переменные r, s, s1 заменяются фактическими  n, sum, sr. Процедура выполняет  ввод элементов массива, вычисляет сумму и среднее значение. Переменные  s и s1 возвращаются в главную программу, поэтому перед их описанием ставится служебное слово var. Локальные параметры mas, j действуют только в процедуре. Глобальные - i, n, sum, sr доступны во всей программе.    **Функции в Паскале**  Набор встроенных функций в языке Паскаль достаточно широк (ABS, SQR, TRUNC и т.д.). Если в программу включается новая, нестандартная функция, то ее необходимо описать в тексте программы, после чего можно обращаться к ней из программы. Обращение к функции осуществляется в правой части оператора присваивания, с указанием имени функции и фактических параметров. Функция может иметь собственные локальные константы, типы, переменные, процедуры и функции. Описание функций в Паскале аналогично описанию процедур. **Отличительные особенности функций**:  **- результат выполнения - одно значение, которое присваивается имени функции и передается в основную программу; - имя функции может входить в выражение как операнд.**  **Пример 3.** Написать  подпрограмму-функцию степени  аx, где a, х – любые числа. Воспользуемся формулой: аx = ex ln a  program p2;            var  f,  b, s, t, c, d : real; { глобальные параметры}        function  stp (a, x : real) : real;        var   y : real;  { локальные параметры}           begin              y := exp (x \* ln ( a)) ;              stp:= y;{присвоение имени функции результата вычислений подпр-мы}      end;  { описание функции закончено }  begin             {начало основной программы }   d:= stp  (2.4, 5); {вычисление степеней  разных  чисел и переменных }   writeln (d,  stp (5,3.5));        read (f, b, s, t);    c := stp  (f, s)+stp  (b, t);   writeln (c);    end.  **Результат работы программы:**  **http://life-prog.ru/pascal/res7.2.gif**    **Рекурсия**  Процедуры и функции в Паскале могут вызывать сами себя, т.е. обладать свойством рекурсивности. Рекурсивная функция обязательно должна содержать в себе условие окончания рекурсивности, чтобы не вызвать зацикливания программы. При каждом рекурсивном вызове создается новое множество локальных переменных. То есть переменные, расположенные вне вызываемой функции, не изменяются.  **Пример 4.** Составить рекурсивную функцию, вычисляющую факториал числа n следующим образом:    n! = 1   , если   n= 1 и n!= ( n -1 )! · n ,  если  n > 1  function   f ( n : integer): integer; begin  if  n = 1 then  f := 1  else f := n \* f ( n -1 );  { функция f вызывает саму себя} end; |

**Урок 8 — Записи в Паскале. Примеры решение задач в Turbo Pascal**

|  |  |
| --- | --- |
| |  | | --- | |  |   **Записи в Паскале** – фиксированное число элементов одного или нескольких типов, то есть в  отличие от массивов, в которых содержатся элементы одного типа, в записях могут содержаться элементы как одного, так и разных типов. Тема, например, сведения о книгах имеет структуру: автор, название книги, издательство, год издания, её цена. Первые три элемента относится к строковому типу данных, четвертый к целому, а цена  - к вещественному типу. Элементами записей могут быть базовые типы, переменные, массивы, указатели, записи и т.д. Элементы записи вместе с их описанием называются полями записи. Над элементами записи можно выполнять действия, допустимые для данных этого типа.  Все записи должны быть описаны  в разделе **TYPE** . Описание записи начинается со служебного слова **RECORD** заканчивается **END**, между  которыми указывается список имен и типов полей, выбранных пользователем. Все идентификаторы полей в записи должны быть различными. Например, запись Воок можно описать следующим типом card:  TYPE  card = record Author : string [15];     Title: string [20];   Firm: string[10];    year : integer ;    cena  : real End; VAR Book  :  card;  Тип записи (например, card) вводит только шаблон записи и с его именем не связан никакой конкретный обьект. Обращение к полю выполняется с помощью составного имени (селектора записи),  которое состоит из :  Имя\_записи . имя\_элемента  Например, присвоить значения элементам записи Author и Title можно так:   Book.author:=’Довгаль С.И.’; Book.title:=’Турбо Паскаль V 7.0’;  Ввод цены книги с клавиатуры :  readln (Book.cena);  Для упрощения и сокращения записи составных имен используется оператор присоединения **WITH**. Имя записи выносится в заголовок оператора присоединения, а в блоке используются только имена полей записи.  Общий вид оператора присоединения: **WITH  имя записи  DO оператор;**  Предыдущие операторы можно записать проще:  **With  Book  do  begin author:=’Довгаль С.И.’; title:=’Турбо Паскаль V 7.0’; readln (cena); end;**  **Пример**: Из ведомости 3-х студентов с их оценками ( порядковый  номер,  Ф.И.О. и три оценки) определить количество отличников и средний бал каждого студента.  Program Spic; Type wed = record {Тип wed включает 3 поля: n, fio, bal} n : integer ; fio : string[40] ; bal : array [1..3] of integer {Поле bal – массив из 3 оценок } end; Var spisok : wed ; {Запись spicok типа wed} i, j, kol, s : integer; sr : real; Begin kol:=0; {kol- количество отличников} With spisok do {with присоединяет имя записи spisok ко всем } For i:=1 to 3 do { полям внутри цикла For по i } begin n:=i; Write (' Vvedite FIO # ', i ,' '); Readln (fio); s:=0; For j:= 1 to 3 do begin write ( 'Vvedite ocenky: ' ); readln ( bal [j] ); s := s+ bal [j]; end; if s=15 then kol:=kol+1; {подсчет количества отличников} sr := s/3; writeln ( fio, ', Sredniy bal = ', sr:4:1); end; writeln ( ' Kolichestvo otlichnikov = ', kol ) |

**Урок 9 — Файлы в Паскале (Pascal). Работа с файлами. Решение задач**

|  |  |
| --- | --- |
| |  | | --- | |  |   **Файл** - это упорядоченная последовательность однотипных компонентов, расположенных на внешнем носителе. Файлы предназначены только для хранения информации, а обработка этой информации осуществляется  программами. Использование файлов целесообразно в случае:   1. долговременного хранения данных ; 2. доступа различных программ к одним и тем же данным; 3. обработки больших массивов данных, которые невозможно целиком разместить в оперативной памяти компьютера.   В Паскале определены текстовые файлы, типизированные и нетипизированные. Файл, не содержащий ни одного элемента, называется пустым. Создается файл путем добавления новых записей в конец первоначально пустого файла. Длина файла, т.е. количество элементов, не задается при определении файла.  Все файлы должны быть описаны в программе либо в разделе переменных VAR, либо в разделе типов TYPE. Под чтением файла понимают ввод данных из внешнего файла, находящегося на диске, в оперативную память машины. Запись в файл - вывод результатов работы программы из оперативной памяти на диск в файл.  Работа с файлами выполняется следующими процедурами:  **Assign – устанавливает связь между именем файла в программе (файловой переменной ) и физическим именем файла, принятым в ОС. Reset  -  открывает существующий файл для чтения. Rewrite – создает и открывает новый файл для записи на внеш­нем устройстве (если файл ранее существовал, вся предыдущая информация из него стирается). Close  -  закрывает открытый файл.**  Для определения конца файла используется стандартная встро­енная функция EOF (файловая переменная), которая принимает значение True, если достигнут конец файла, и значение False в противном случае.    **Текстовые файлы**  **Текстовые файлы** – файлы на диске, состоящие из символов ASCII. Для разделения строк используются символы «конец строки».  Текстовые файлы являются файлами  с последовательным доступом. В любой момент времени доступна только одна запись файла. Другие записи становятся доступными лишь в результате последовательного продвижения по файлу. Текстовые файлы внутренне разделены на строки, длины которых различны. Для разделения строк используется специальный маркер конца строки. Объявляются текстовые файлы переменной типа text. Обрабатывать их можно только последовательно и с помощью процедур и функций:  **Readln  (f , st )- чтение строки st из файла f и переход на начало следующей ; Writeln (f, st )- запись строки st в файл f и маркера конца строки ; Append (f ) - процедура, открывающая файл f для добавления строк в конец файла; Eoln (st )- логическая функция, результат выполнения которой равен TRUE, если достигнут маркер конца строки st.**  **Пример 1.**  Создать текстовый файл, в который записать 3 предложения. Прочитать этот файл, вывести  его содержимое на  экран. Определить длину каждого предложения.  Program File\_text; var f1 : text; st : string; n: byte; begin assign   (f1, 'file1.txt'); {связать с файлом file1.txt  файловую переменную f1  }   rewrite  (f1); { создать новый файл  с именем file1.txt } writeln ( f1, 'Очень полезно изучать'); { записать предложения в файл} writeln ( f1, ' всем  студентам '); writeln (f1, ' язык  Pascal '); close (f1);    { закрыть файл для записи } reset (f1); { открыть файл для чтения } while not eof (f1) do { пока не конец файла  f1} begin readln (f1, st);    {читаем строку из файла f1 } writeln(st);         { выводим на экран } n:= length (st);     {определяем длину строки } writeln ('  длина =',n); end; close (f1); { закрыть файл  для чтения} end .  **Типизированные файлы**  **Типизированные файлы** – это файлы, состоящие из нумерованной последовательности объектов (записей) любого типа. С такими файлами можно работать в режиме прямого доступа, при котором выполняется непосредственное об­ращение к любой записи файла. Каждая запись файла имеет свой номер, начиная с 0 и т.д. Процедуры и функции обработки файлов:  1)  Write и Read- записывают и читают информацию из указанного файла и перемещают  указатель файла к сле­дующей записи. 2) Seek (файловая переменная, номер записи); процедура перемещения указателя на запись файла с заданным номером.  3) Truncate (файловая переменная); процедура, усекающая файл по текущей позиции указа­теля файла, т.е. все записи, находящиеся после указателя фай­ла, удаляются. 4) Функция Filesize (файловая переменная);  имеет тип Integer и определяет размер файла, т.е. число записей.   5) Функция Filepos (файловая переменная); имеет тип Integer и возвращает текущую позицию  указателя файла.  Для добавления записей в конец файла используются процедуры:  **Readln** (a ); **Seek** (f, filesize (f)); **Write** (f, a);  При этом указатель устанавливается за конец файла, т.к. нуме­рация записей начинается с нуля. После чего с помощью Write можно добавлять записи. Открывать файл можно только проце­дурой Reset (f). Для того, чтобы в режиме произвольного доступа считать, а затем изменить значение записи, следует выполнить два вы­зова процедуры Seek.Один вызов перед операцией Read, а другой - перед операцией Write (т.к. Read после чтения записи переместит указатель к сле­дующей записи).  **Пример:** Cоздать файл  из списка 10 студентов с их оценками ( номер,  Ф.И.О. и три оценки). Вывести его содержимое на экран, изменить фамилию студента с номером, введенным с клавиатуры, заново прочитать файл.  Program  file;  Type wed  =  record      {Тип wed включает 3 поля: n, fio, bal} n : byte ;       fio : string[15] ; bal : array [1..3] of byte;   {Поле bal – массив из 3 оценок } end; Var    spisok : wed ;       {Запись spicok  типа  wed} sp : file  of wed;  {Файл записей типа wed} procedure vvod;     { процедура создания файла} var    i,j:byte; begin { оператор assing  находится в основной прграмме } rewrite ( sp);            {открытие файла для записи} with spisok do For   i:=1  to  10  do      begin n:=i;                writeln (' Введите фамилию -  ',  i );    readln (fio);   writeln (' Введите 3 оценки  ', fio );    For  j:= 1 to 3 do  readln ( bal [j] ); write (sp , spisok);     { запись в файл информации о студенте} end; close (sp);      { закрытие файла для записи } end; procedure    print;  { процедура  чтения   и печати всего файла } var    j : byte; begin reset ( sp);               {открытие файла для чтения} writeln (‘                 Список студентов:  ‘);  while not eof (sp) do      with spisok do  begin  Read (sp, spisok);   {чтение данных из файла} write (n,'  ',fio);      {вывод записи на экран} For  j:= 1 to 3 do  write (' ', bal [j] ); writeln ; end; readln; close (sp) ; end; procedure       work; var    num: integer; begin reset ( sp);               {открытие файла для чтения} writeln ('номер= ');    readln (num); seek (sp, num-1); {поиск записи с указанным номером (нумерация записей с 0)} read (sp,spisok);{чтение и перемещение указателя к сле­д. записи} write ('fio='); writeln (spisok.fio); seek (sp,filepos(sp)-1); {возвращение к изменяемой записи } writeln (‘ Введите новую фамилию’  ); readln (spisok.fio); write (sp, spisok);  {запись в файл измененной записи} close (sp); end;  begin      {начало основной программы} assign (sp,'Vedom.DAT'); {связать файловую перем-ю  sp с файлом Vedom.dat} vvod;      print; {процедуры создания и чтения файла}      work;   print;   {корректировка и чтение измененного файла} readln end. |

**Урок 10 — Динамическая память. Односвязный список. Примеры и использование**

|  |  |
| --- | --- |
| |  | | --- | |  |   Рассмотренные ранее структуры данных являлись  статическими.  Это  означает, что память под эти данные выделялась на этапе компиляции. Такое выделение памяти  не  всегда  удобно, т.к.  заранее  трудно  предвидеть, например,  размер  массива  для сортировки или число уравнений в системе. Поэтому в Турбо Паскале существует возможность выделения памяти не на этапе компиляции, а на этапе  исполнения программы - динамическая память.  Процедура  выделения памяти связана с процедурой ее освобождения после использования. В Турбо Паскале имеется три пары  процедур  **выделения-освобождения  памяти**:  **New  -  Dispose, GetMem - FreeMem, Mark - Release**. Чаще всего используется пара New для выделения памяти и Dispose для ее освобождения. Подчеркнем, что  использование  Dispose для освобождения выделенной динамической памяти не является обязательным, после выхода из программы  динамическая  память освобождается автоматически, но частое использование процедуры выделения памяти без ее освобождения может привести к состоянию невозможности дальнейшего выделения памяти и к остановке  программы.  Рассмотрим работу с динамической памятью на  примере  односвязного  списка. Такой  список  содержит  записи  с информационной  частью (например, фамилия, имя, отчество, адрес, номер телефона и  т.д.) и ссылочной частью (например, адрес следующей или  предыдущей записи). Такой список принципиально отличается от массива тем,  что  в массиве записи упорядочены, удаление одной записи приводит  к необходимости переписывать остальные записи на  место  предыдущих,  в  то  время  как в связном списке следует лишь переставить  указатель. Число записей в массиве фиксировано, в связном  списке  - нет.  **Пример:**  program spisok;{односвязный список} type uk=^rec; rec = record fio : string[15];{ ФИО пациента} davl : integer;  { давление} v : uk           { указатель на следующую запись} end; var  tek, pred, perv, rab : uk; {указатели на текущую, предыдущую, первую, рабочую записи} k : integer; begin {ввод 5 записей пациентов} new (tek); {выделение памяти} write ('введите фамилию пациента:  ');    readln (tek^.fio); write ('введите давление:  ');             readln (tek^.davl); tek^.v := nil; {следующей записи пока нет} pred:=tek; perv:=tek; for k:=2 to 5 do begin new (tek); {выделение памяти} pred^.v := tek; write ('введите фамилию пациента:  ');      readln (tek^.fio); write ('введите давление:  ');                  readln (tek^.davl); tek^.v:=nil; {следующей записи пока нет} pred:=tek; end; {удаление записей пациентов с давлением большим 140} tek:=perv; {указатель-на начало списка} pred:=nil;  {предыдущей записи пока нет} while tek<>nil do begin if tek^.davl>140 then begin rab:=tek; if tek<>perv then  pred^.v:=tek^.v else  perv:=tek^.v; if tek^.v=nil then {запись последняя} if tek<>perv then{список не пуст} pred^.v:=nil else {список пуст} perv:=nil; dispose (rab); end else pred:=tek;{переставляем указатель  предыдущей записи, если текущую запись не удаляли} tek:=tek^.v; {переход к следующей записи} end; {печать оставшихся записей} writeln ('Оставшиеся записи в односвязном списке'); if perv<>nil then {список не пуст} begin tek:=perv; while tek<>nil do begin writeln('ФИО  ',tek^.fio,'  давление   ', tek^.davl); tek:=tek^.v {переход к след. записи} end end  end.  В  данной программе rec - запись, содержащая сведения о пациенте: fio - фамилия, имя, отчество (15 символов),  davl  -  давление  (целое  число),  v  - указатель на следующую запись. Тип указатель  занимает в памяти 4 байта (2 байта сегментный  регистр,  2  байта  смещение),  независимо от размера переменной, на которую он указывает. При описании типа указателя uk : ^rec; допускается  употреблять  неописанный  тип rec (тип rec описан позже). Перед типом rec  ставится символ ^ (карат), что означает: uk является указателем на  тип rec. Когда речь идет о конкретной переменной, на которую  указывает  указатель, знак карата переносится в конец имени указателя, например, tek^.davl означает поле "давление" записи, на которую указывает указатель tek. Для выделения памяти используется процедура new(<указатель>). В данном случае new(tek) означает выделение памяти для  записи, на  которую указывает tek. Таким образом, tek - это адрес в оперативной  памяти,  с  которой начинается вновь образованная переменная  tek^ , имеющая тип записи rec. Конечно, заранее  знать  этот  адрес  невозможно, поэтому выделение динамической памяти производится на  этапе исполнения (RunTime).  Тип uk (указатель на rec) имеют переменные tek, pred, perv, rab  для текущей, предыдущей, первой, рабочей записей. Односвязный список  всегда  проходится в одном направлении, в данном случае от начала  к концу, для установки на начало служит указатель на  первую  запись perv.  В записи имеется ссылочная  (адресная) часть v - указатель  на следующую запись. Для последней записи списка,  это - указатель в никуда, имеющий  имя  nil.  В начале программы вводится 5 записей пациентов. Ввод  первой записи  описан отдельно от остальных для инициализации указателей  (в частности, указателя pred на предыдущую запись, которая   ранее  не существовала и определяется только после ввода первой записи). Для каждой записи вначале ссылочному полю v присваиваем nil,  что  означает отсутствие следующей записи. В дальнейшем, при выделении  памяти  следующей  записи  процедурой new в это поле записывается  текущий указатель, но не для текущей, а для предыдущей записи.  В  последней же записи на месте v остается nil.  Затем  в программе описано удаление записей пациентов с давлением, большим  140. Односвязный  список просматривается с начала,  для чего текущему указателю tek присваивается значение perv  указателя на первую запись. Просматривается поле давление  tek^.davl  для  текущей записи и если оно больше 140, запись подлежит удалению. При этом действия различны,  является  ли  удаляемая  запись  первой или нет.  Если  удаляемая запись первая, то указатель на первую запись  perv  нужно  передвинуть  на  следующую запись perv:=tek^.v.  Если  требуется удалить и следующую запись, она вновь будет  первой,  и  вновь указатель первой записи будет изменен, как и следует. Если  же удаляемая запись не первая, то указатель предыдущей   записи устанавливаем на адрес, на который указывает ссылка в  текущей  записи,  таким  образом,  текущая запись будет пропущена в  списке.  Если удаляемая запись последняя (tek^.v=nil), то  следует  в  ссылочной части предыдущей записи установить признак конца списка  pred^.v:=nil,  если  эта  предыдущая запись существует (список не  пуст, т.е. удалены не все записи). Если же  удалены  все  записи,  следует установить perv:=nil.  Далее  в  программе  следует печать оставшихся записей, если  список не пуст. Устанавливаем текущий указатель на первую запись и  выводим на экран информационные поля fio и davl, затем  переходим  к следующей записи tek:=tek^.v. |

**Урок 11 — <<Графика в Паскале>> Примеры**

|  |  |
| --- | --- |
| |  | | --- | |  |   Графика в Паскале строится при помощи подключения модуля **Граф**, то есть на экране компьютера можно получать не только последовательности  символов, но и разнообразные рисунки, схемы, картинки. В нашем примере - это **построить график функции в Паскале**. Для этого в Паскаль включаются специальные средства - **графические процедуры** и **операторы**, которые находятся в **модуле Graph** (uses Graph;).  Цель урока – это **познакомиться с возможностями графических операторов**,**построение графических изображений в Паскале**.  Следует отметить, что графическое изображение на экране составляется из точек (например, как фотографии в газетах, журналах и др.). Количество точек (пикселей) на экране зависит от разрешающей способности экрана. Каждая точка задается двумя координатами (x, y). Точка с координатами (0,0) находится в левом верхнем углу экрана. Ось Х направлена вправо, а ось У вниз.  **Процесс  построения изображений с помощью графических процедур разбивается на отдельные этапы:**  1. Переключить монитор в графический режим с помощью оператора InitGraph (Driver, mode, <путь к драйверу>). 2. Установить разрешающую способность экрана по умолчанию режимом Detect или процедурой SetGraphMode. Режим Detect устанавливает разрешающую способность экрана 640\*480 пикселей, т.е. координата  Х может принимать значения от 0 по 639, а У от 0 по 479. 3. Очистить  и инициализировать графический экран процедурой ClearDevice. 4. Установить цвет фона  оператором  SetBkColor и цвет изображения  оператором SetColor. 5. Вывести на экран точки,  отрезки, прямоугольники, дуги, окружности, эллипсы. 6. Вывести на экран закрашенные фигуры. 7. Вывести тексты и подписи на экран. Для использования операторов Write и Writeln в графическом режиме необходимо выполнить следующую операцию присваивания : DirectVideo := FALSE;  Или использовать процедуру Outtextxy(x,y,st), которая выводит строку st,  начиная с позиции  x, y.  **Цвета  в операторах  задаются с помощью своих кодов:**  0 – черный;        4 – красный;           8 – темно-серый;      12 – ярко-красный; 1 – синий;          5 – пурпурный;           9 – ярко-синий;    13 – ярко-пурпурный; 2 –зеленый;       6 – коричневый;     10 – ярко-зеленый;    14 – желтый; 3 – бирюзовый;      7 – светло-серый;    11 – ярко-бирюзовый;    15 – белый.  Цифровое кодирование цвета соответствует последовательности IRGB для 0-3 битов. Бит 3 – бит интенсивности I, бит 2 – бит красного R, бит 1 – бит зеленого G, бит 0 – бит синего B. Например, 11=8+2+1, т.е. биты 3,1,0 – интенсивный сине-зеленый (бирюзовый).  **Основные графические операторы для построения изображений:**  PutPixel (X, Y, цвет) - вывод точки на экран, где X, Y  - координаты точки ; Line ( X1, Y1, X2, Y2 ) - проводит линию из точки с координатами (X1, Y1 ) в точку с координатами ( X2, Y2 ); Rectangle (X1, Y1, X2, Y2 ) - прямоугольник со сторонами, параллельными осям координат;  (X1, Y1) и (X2, Y2)  - координаты, определяющие одну из диагоналей прямоугольника ; Bar ( X1, Y1, X2, Y2 ) - закрашенный прямоугольник (без окантовки); Circle ( X, Y, радиус ) – на экран выводится окружность с центром в точке  ( X, Y )(тип integer) ; Arc (X, Y, начальный угол, конечный угол, радиус) - на  экран  выводится  дуга окружности с центром в точке (X, Y ); углы задаются в градусах; дуга рисуется ПРОТИВ часовой стрелки; Ellipse ( X, Y, начальный угол, конечный угол,  горизонтальный радиус, вертикальный радиус) -  на экран выводится эллиптическая дуга с центром в точке с координатами ( X, Y ) (тип integer); SetFillStyle (заполнение, цвет) – определение вида и цвета заполнения области; FloodFill (x, y, цвет границы) – заливка замкнутой области.  **Пример программы построения  графика функции.**  program graphic;  uses graph;  var driver, mode, errorcode : integer; xm,ym,i,j : integer;  pi,pi300,x1,y1,x2,y2, sc : real;  st1,st2,st3 : string;  function f(x:real) : real; begin f:=sin(x)+sin(2\*x)+sin(3\*x)-1-cos(x)-cos(2\*x);{ функция для построения} end; {графика}  begin  st1:='x';st2:='y';  st3:=' Press ENTER';  sc:=50;  driver:=9; {egavga}  mode:=2; {640х480 пикселей}  initgraph(driver,mode,'d:\bp\bgi');**{инициализация графического режима }** errorcode:=graphresult; if errorcode<>grok then {ошибка } begin writeln('Error init Graph'); closegraph;  halt; end; xm:=getmaxx div 2; ym:=getmaxy div 2; { xm=320;ym=240;центр экрана} line(xm,20,xm,460);{ось y} line(20,ym,620,ym);{ ось x} outtextxy(630,ym,st1); {маркировка оси х} outtextxy(xm,10,st2); {маркировка оси у} pi:=3.1415926; pi300:=pi/300; x1:=-pi; for i:=0 to 24 do {разметка оси х вертикальными черточками} begin line(xm+round(80\*x1),230,xm+round(80\*x1),250); x1:=x1+pi300\*25; end; x1:=-pi; {собственно построение графика отрезками прямых} while x1<pi do begin y1:=f(x1);x2:=x1+pi300; y2:=f(x2); line(xm+round(80\*x1), ym-round(sc\*y1), xm+round(80\*x2), ym-round(sc\*y2)); x1:=x2; end; outtextxy(270,470,st3); readln; closegraph; end.  Программа компилируется и выдает график функции. Показать картинту я не смогу, потому что он выполняется только в полноэкранном режиме |

**Урок 12 — Модули Паскаля(Pascal). Модуль Graph в Паскале**

|  |  |
| --- | --- |
| |  | | --- | |  |   **Модуль** не представляет собой исполняемой программы, а только содержит набор средств для использования в исполняемой программе: типы данных, переменные, процедуры и функции. Исходный текст модуля имеет расширение .pas. Модуль компилируется отдельно от основной программы, откомпилированный модуль имеет расширение  .tpu (Turbo Pascal Unit). Готовый откомпилированный модуль может быть использован в основной программе с помощью предложения uses, которое записывается сразуже после имени программы.  В Турбо Паскале имеются стандартные модули :**DOS, CRT (Cathode Ray Tube, электронно-лучевая трубка), Printer, Graph, System** и другие. Их не надо описывать, а можно сразу включать в программу предложением uses. Модуль System используется во всех программах, поэтому для него сделано исключение, его можно не включать в предложение uses, он будет подключен автоматически. Остальные модули требуют включения в предложение uses,  если в программе используются ресурсы модуля, например, процедуры и функции.  Назначение стандартных модулей следующее. Модуль DOS использует системные ресурсы операционной системы MS-DOS . Возможно использование регистров, обслуживание прерываний, вызов других программ из программы на Паскале. Модуль CRT используется для обслуживания видеомонитора, клавиатуры, встроенного динамика. Процедуры очистки экрана clrscr и ожидание нажатия клавиши readkey содержатся именно в модуле CRT.  **Модуль Graph** содержит богатейший набор графических **процедур и функций**: проведение линий, работа с цветом, вызов графических примитивов, работа с текстом в графическом режиме, закраска замкнутых областей и т.п. Экран в графическом режиме рассматривается как набор пикселей. Модуль Printer служит для вывода информации на принтер. Кроме стандартных модулей, в Турбо Паскале возможны модули пользователя, Они требуют описания, которое начинается ключевым словом языка unit и содержат 3 раздела: раздел интерфейса (interface), раздел реализации (implementation), раздел инициализации (необязательный).  После введения модулей Турбо Паскаль стал одним из наиболее распространенных языков профессионального программирования. С помощью модулей оказалось возможным выполнять большие работы коллективом программистов, разбив задачу на отдельные части, каждая из которых разрабатывается одним программистом.  За счет параллельной работы многих программистов срок разработки сокращается во много раз. Руководитель проекта распределяет работу и обеспечивает стыковку отдельных модулей в единое целое. При этом каждый программист может пользоваться своими идентификаторами, независимо от остальных. Совпадение идентификаторов в различных модулях не оказывает влияния на программу, т.к. каждый идентификатор используется в своем модуле.  Рассмотрим **пример модуля для работы с комплексными числами**. Комплексное число состоит из двух вещественных чисел, действительной и мнимой частей соответственно. Пусть имеются два комплексных числа  x и y: x=a+bi; y=c+di. Здесь i квадратный корень из минус единицы (мнимая единица), причем i2= - 1. Если комплексное число z=e+fi, и z есть результат операции над  x и y, то  z=x+y; e=a+c: f=b+d;  (сложение) z=x-y; e=a-c: f=b-d;  (вычитание) z=x\*y; e=a\*c-b\*d: f=a\*d+b\*c;  (умножение) z=x/y; e=(a\*c+b\*d)/(c2+d2): f=(c\*b-a\*d)/(c2+d2):  (деление)  Легко видеть, что при сложении и вычитании складываются и вычитаются соответственно действительные и мнимые части комплексных чисел, умножение и деление более сложны.  Для организации операций над комплексными числами представим каждое комплексное число в виде записи из двух полей вещественного типа: Re –действительная часть, Im – мнимая часть. Создадим модуль complex, в котором опишем комплексный тип com с двумя полями. В этом же модуле в разделе интерфейса опишем процедуры ad (сложение), su (вычитание), mu (умножение), di (деление) комплексных чисел (только заголовки). Тела процедур поместим в раздел реализации модуля. Раздел инициализации имеет вид:  begin    операторы end.  Если раздел инициализации не содержит операторов, как в нашем примере, достаточно написать только end. Пример модуля complex.pas.  unit complex;{модуль арифметики комплексных чисел} {$N+} interface uses dos,crt; type com=record   re,im:real; end; var a,b,c:com; procedure ad(var a,b,c:com);{сложение,  c=a+b} procedure su(var a,b,c:com);{вычитание, c=a-b} procedure mu(var a,b,c:com);{умножение, c=a\*b} procedure di(var a,b,c:com);{деление,   c=a/b, случай  b =0 не предусмотрен }                    procedure wr\_com(var p:com); {печать комплексного числа} implementation procedure ad(var a,b,c:com);  begin c.re:=a.re+b.re;  c.im:=a.im+b.im; end;{ad} procedure su(var a,b,c:com); begin c.re:=a.re-b.re; c.im:=a.im-b.im; end;{su} procedure mu(var a,b,c:com); begin c.re:=a.re\*b.re-a.im\*b.im; c.im:=a.re\*b.im+a.im\*b.re; end;{mu} procedure di(var a,b,c:com); var d:real; begin  d:=b.re\*b.re+b.im\*b.im; c.re:=(a.re\*b.re+a.im\*b.im)/d; c.im:=(b.re\*a.im-a.re\*b.im)/d; end;{di} procedure wr\_com(var p:com);   begin     if p.im<0 then writeln(p.re:9:5,'-',-p.im:9:5,'i') else     writeln(p.re:9:5,'+',p.im:9:5,'i');   end;{wr\_com} begin end.{unit}  Как уже было указано, модуль не может непосредственно выполняться. Нужно разработать основную программу, которая использует ресурсы модуля. В данном случае это программа com\_arif.pas (комплексная арифметика). В программе вводятся 2 комплексных числа, затем с ними выполняются действия сложения, вычитания, умножения и деления, результаты выводятся на экран.  program com\_arif; uses complex; var x,y,z:com; {тип com определен в модуле complex} begin writeln('Введите дейст. и мн. части первого числа'); read(x.re,x.im); writeln('Введите дейст. и мн. части второго числа'); read(y.re,y.im); write('x='); wr\_com(x); write('y='); wr\_com(y); ad(x,y,z); {z=x+y} write('x+y=');wr\_com(z); su(x,y,z); {z=x-y} write('x-y=');wr\_com(z); mu(x,y,z); {z=x\*y} write('x\*y=');wr\_com(z); di(x,y,z); {z=x/y} write('x/y=');wr\_com(z); end. |