Среда программирования PascalABC

1. Язык программирования Pascal

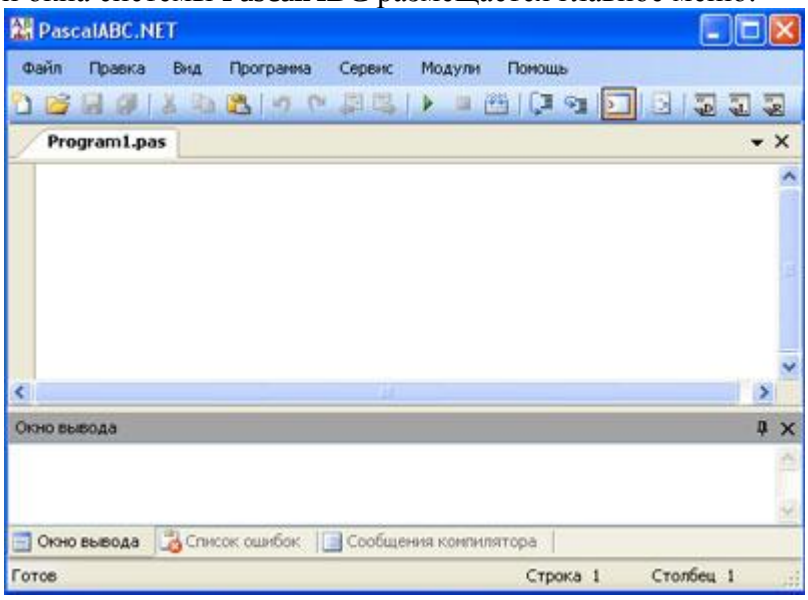
Язык Pascal (Паскаль) был разработан швейцарским профессором Никлаусом Виртом в конце 60-х - начале 70-х годов и назван в честь выдающегося французского математика и философа Блеза Паскаля. Первоначально этот язык был создан для обучения программированию. Однако благодаря заложенным в нем большим возможностям структурного программирования он стал широко применяться в различных областях: науке, технике, экономике, при создании информационных систем и т.д.

Существует множество версий языка Паскаль. Мы будем рассматривать систему программирования PascalABC, разработанную на матфаке Южного Федерального Университета. http://pascalabc.net

2. Запуск и элементы окна PascalABC

Запуск программы: *Пуск\ Программы\ PascalABC.NET\ PascalABC.NET*

В верхней части окна системы PascalABC размещается главное меню:



Центральную часть окна занимает рабочая область, предназначенная для работы с текстом программ. В нижней части окна расположено окно вывода. Помимо перечисленных элементов, имеются еще вертикальная и горизонтальная полосы прокрутки, кнопка закрытия окна и др.

3. Этапы работы с программой.

Ввод, редактирование и сохранение программы.

Сразу после запуска системы PascalABC вы видите на экране окно редактирования. Мигающий курсор как бы приглашает вас к вводу текста программы. Рассмотрим правила работы в режиме редактирования.

Вы можете начать работать с новым кодом, набирая его в окне редактирования, либо отредактировать уже существующий программный файл. Чтобы открыть файл, воспользуйтесь командой *Файл\ Открыть*. В последующем диалоговом окне откройте нужную папку и выберите файл с расширением .pas.

Каждому открытому файлу будет отвечать отдельная вкладка в окне, в этом смысле среда программирования PascalABC является многооконной. На вкладке указывается имя программного файла.

Текст программы сохраняется в виде файла с расширением .pas.

Компиляция, исправление ошибок, получение справки.

Трансляция – это процесс перевода программы с языка программирования высокого уровня на язык машинных кодов. Трансляция бывает двух видов – компиляция и

интерпретация – в зависимости от того, переводится ли программа вся сразу целиком или последовательно по одной команде.

Трансляция программы, написанной на PascalABC, осуществляется с помощью компилятора, входящего в состав системы. Для запуска программы нажмите зелёный треугольник или *Программа\ Выполнить*.

В ходе компиляции в окне вывода могут появиться сообщения об ошибках. Ошибкой является любое отклонение от синтаксиса языка Pascal, например, пропущенные разделители, необъявленная переменная или тип данных, неопределенная константа, подпрограмма и т.д. Увидеть сообщения компилятора об ошибках можно в нижней части окна программы на вкладке Сообщения компилятора или изучив Список ошибок.

При желании пользователь может записать программу на Паскале как исполняемый файл с расширением exe. Для создания exe-файла необходимо после успешного запуска программы выполнить *Программа\ Компилировать*.

Диалог программы с пользователем и результаты работы программы отображаются в Окне вывода, а для графического вывода – в отдельном окне.

4. Структура программы

В Паскале программа должна иметь определенную структуру и состоять из следующих разделов:

<заголовок> - program ...
<описание внешних модулей> - uses ...
<описание меток> - label ...
<описание констант> - const ...
<описание типов переменных> - type ...
<описание переменных> - var...
<описание процедур> - procedure ...
<описание функций> - function ...
<раздел операторов> - begin ...

Каждый из разделов начинается со служебного слова и заканчивается точкой с запятой. Эти начальные служебные слова даны в правой колонке приведенного перечня разделов программы.

Не все из перечисленных разделов обязательно должны присутствовать в программе. В простых программах могут быть только заголовок, описание переменных и раздел операторов. Вообще любой раздел, кроме раздела операторов, в программе может отсутствовать. В PascalABC не обязательно должен присутствовать заголовок.

Что касается порядка следования разделов, то некоторые разделы (описания, кроме uses) могут располагаться в произвольном месте программы и встречаться в программе любое количество раз. Главное правило, которое при этом должно соблюдаться: описание идентификатора должно предшествовать его использованию в разделе операторов.

Комментарии

Отметим, что в любое место программы можно добавлять поясняющий текст - комментарии. Комментарии должны начинаться парой символов: // или заключаться в фигурные скобки.

Описание меток

В программе, написанной на Паскале, перед любым оператором можно поставить метку. Метка отмечает обычно инструкцию, к которой можно перейти из любого места программы с помощью оператора перехода. В качестве метки может выступать любое сочетание из букв и цифр длиной до 127 символов.

Все используемые в программе метки должны быть описаны. Раздел описания меток начинается со слова label и имеет, например, вид:

```
label
  Ml, Tok, llab, 123;
```

В разделе операторов после идентификатора метки ставится двоеточие, которое

указывает компилятору, что данный идентификатор является меткой:

Ml: <оператор>

Tok: <оператор>

Количество описанных меток может превышать число использованных меток, и это не будет считаться ошибкой. Поэтому программист может заранее описать избыточное количество меток и применять их по мере расширения программы.

Описание констант

При составлении первых программ на Паскале привыкните к определенной последовательности записи разделов описаний. Это не повлияет на исполнение программы, но поможет вам в дальнейшем анализировать программу.

После описания меток обычно следует описание констант. В этом разделе идентификаторам констант присваиваются постоянные значения. Описание констант может выглядеть следующим образом:

```
Const
   Inf = 1024; //Целая константа
   Communic = ‘Связь установлена’; //Строковая константа
   Ref = 13.45; //Вещественная константа
```

Описание переменных

Все переменные, которые встречаются в программе, должны быть описаны. Описание переменных выполняется по следующей схеме:

var <идентификатор>: <тип>;

То есть описание начинается с зарезервированного слова var (от слова variable - переменная). Затем следует перечисление имен переменных, разделенных запятыми. Имена переменных отделяются двоеточием от указания их типа (о типах величин см. следующий параграф). Например:

```
var
  A, D, M: integer;
  XI, X2: real;
  L: boolean;
 Text: string;
```

Раздел операторов

Исполняемой частью программы является раздел операторов, который следует за разделом описаний. В разделе операторов выполняются действия над предварительно описанными переменными, константами, функциями и т.д. Именно в этом разделе получается результат, ради которого составлялась программа. Начинается раздел служебным словом begin и заканчивается словом end с точкой.

Операторы языка Паскаль бывают простыми и составными (или, иначе говоря, структурными).

> ***Простыми*** *называются те операторы, которые не содержат никаких других операторов. К простым операторам относятся: присваивание, оператор перехода, оператор вызова процедуры и пустой оператор.*

Оператор перехода goto (его называют еще оператором безусловного перехода) применяется в тех случаях, когда после выполнения некоторого оператора нужно выполнить не следующий по порядку в записи программы, а какой-либо другой оператор. Для выделения оператора, к которому нужно совершить переход, используется метка. Переход осуществляется следующим образом:

```
goto  Labl;
Labl:   A:=A*D;
```

Когда в программе дойдет очередь до оператора goto Labl, будет исполнен оператор A:=A*D, стоящий в строке с меткой Labl. Вслед за оператором с меткой будет выполняться следующая строка программы.

5. Типы данных и их описания

Любой элемент данных (константу, переменную) можно отнести к тому или иному

типу. Тип определяет множество значений, которые может принимать элемент данных. Все типы данных должны быть понятны компилятору, и поэтому те типы, которые вводятся программистом, необходимо описать.

Целочисленные типы

Кроме известного вам типа integer, представляющего значения целых в диапазоне от -32768 до 32767, в Pascal имеются и другие целочисленные типы:

byte - числа в диапазоне 0 . . 255;
shortint - числа в диапазоне -128 .. 127;
word - числа в диапазоне 0 . . 65535;
longint - числа от -2147483648 до 2147483647.

Границы диапазонов определяются тем, что для хранения переменных типа byte или shortint отводится 1 байт оперативной памяти, для переменных integer и word - 2 байта, а для longint - 4 байта.

Приведем пример описания целочисленных переменных:

```
var
    HI, H2: word;
    Zcolor: byte;
```

Над данными целого типа можно выполнять арифметические операции и операции отношения. К целочисленным данным применяют также стандартные функции: sin(x), cos(x), arctan(x), exp(x), ln(x), sqrt(x) и др.

Вещественные типы

В арифметических выражениях обычно используются переменные, принимающие вещественные значения. Напомним, что все вещественные числа могут изображаться в форме с фиксированной точкой (например, 0.13, 4.671, 6133.99 и т.д.) и с плавающей точкой (например, $2*10^3$, $5.17*10^{-7}$ и т.д.). Значения с плавающей точкой в языке Pascal записываются в формате <мантисса>E<порядок> например, 2E3 или 5.17E -1.

Для представления вещественных значений в Pascal чаще всего используются типы: real, single, double. Эти типы различаются диапазоном допустимых значений и объемом требуемой памяти:

real - числа от 2.9E-39 до 1.7E38 с мантиссой 11-12 знаков; отводится 6 байт памяти;

single - числа от 1.5E-45 до 3.4E38 с количеством значащих цифр 7-8; отводится 4 байта памяти;

double - числа от 5.0E-324 до 1.7E308 с количеством значащих цифр 15 - 16; отводится 8 байт памяти;

Формат описания вещественных типов аналогичен описанию целочисленных типов:

```
var
    Yl, Y2: single;
    ZZ: double;
```

К данным вещественного типа применяются те же операции, что и к целым. При записи операций присваивания нужно помнить, что переменной вещественного типа можно присвоить значение выражения целого типа, но не наоборот.

Символьный тип

Переменные, которые принимают символьные значения из таблицы ASCII, принадлежат к символьному типу - char. Для размещения таких переменных в памяти требуется всего один байт.

В программу на Паскале символьные переменные вводятся с помощью описания вида:

```
var
  Wx, Ux: char;
```

Если в программе встречаются значения символьных переменных, они должны быть заключены в апострофы, например, X='B'.

Для переменных типа char в Паскале предусмотрена стандартная функция ord (X), которая преобразует символ X в его ASCII-код. Так, для символа ‘B’ функция ord возвращает значение 66. Обратное преобразование кода в символ осуществляется функцией

chr(X). То есть функция chr(66) возвращает символ ‘B’.

Логический тип

Величины логического (булевского) типа рассматривались нами ранее. При описании этих величин в программе на Паскале используется слово boolean, например

```
var
   Pozit, Sel, boolean;
```

Величины этого типа могут принимать только 2 значения: ложь или истина.

Перечисляемый тип

Кроме рассмотренных выше типов, программист по своему желанию может вводить новые типы данных. К их числу относится *перечисляемый* тип данных, который определяется путем перечисления его элементов по следующей схеме:

type <имя типа> = <список имен>

Раздел описания типов данных в этом случае начинается служебным словом type, после которого следуют имена типов и списки значений. Отдельные значения в списке указываются через запятую, а сам список заключается в круглые скобки. При описании переменных в разделе var указывается принадлежность тому или иному типу. Например:

```
type
   Metal = (Copper, Tantal, Cobalt, Silver);
   Index = (1, 5, 8, 13);
var
   Ml, M2: Metal;
   Ix, ly, Init: Index;
   Size: {Little, Middle, Big);
```

Это описание перечисляемых типов Metal и Index. Переменные Ml, M2 типа Metal могут принимать только значения из ряда: Copper, Tantal... а переменные Ix, ly, Init типа index - целые значения: 1, 5, 8, 13. Переменная Size не имеет определенного типа, однако для нее в разделе var заданы возможные значения: Little, Middle, Big. Попытка присвоить какой-либо переменной иное значение, не указанное в ее описании, вызовет программное прерывание.

Тип-диапазон |

При задании рассмотренного выше перечисляемого типа необходимо составить список возможных значений. Однако в некоторых случаях удобнее не перечислять все значения, а просто указать границы интервала, в котором эти значения лежат. Для этого применяется *интервальный* тип данных (его еще называют *тип-диапазон).* При описании этого типа указывается интервал: от наименьшего до наибольшего значения. Эти крайние значения разделяются двумя последовательными точками, например:

```
type
   Element = 100..200;
   Letter = ( 'a'.. 'z' );
var
   Number, N1:  Element;
   Bukva: Letter;
```

В этом описании тип Element определяет множество целых чисел от 100 до 200, а тип Letter - множество букв латинского алфавита от a до z. Переменные Number и N1 принадлежат типу Element, а переменная Bukva: - типу Letter.

Структурированные типы

Все рассмотренные выше типы данных (целый, вещественный, символьный, логический, перечисляемый, тип-диапазон) не содержат составных частей и поэтому называются *простыми* или *скалярными* типами. Наряду с простыми типами, в Паскале предусмотрены *структурированные* типы, в которых данные состоят из компонентов.

Структурированные типы *данных представляют собой наборы однотипных*

или разнотипных компонентов. Типы компонентов образуются из других типов данных (простых или структурированных).

К структурированным типам относятся строки, массивы, записи, файлы и (другие типы. Представление о некоторых из них вы получили ранее. Здесь мы рассмотрим массивы и их описание в программах на Паскале. Описание массива задается следующей схемой:

type <имя типа> = array [<список индексов>] of <тип>

где <имя типа> - идентификатор типа; array и of - зарезервированные слова; <список индексов> - список диапазонов индексов или других индексных типов; <тип> - любой тип данных. Приведем пример:

```
type
   Vector = array [1..3] of real;
   Table = array [1..5,  1..9] of  integer;
   Cub = array [0..4, -2..2, N1] of  byte;
```

В данном примере Vector - это имя типа одномерного массива, состоящего из трех элементов, принимающих вещественные значения. Тип Table - тип двумерного массива размером 5x9, состоящего из целых чисел. Тип Cub - это тип трехмерного массива, состоящего из целых значений типа byte. Третья компонента массива типа Cub обозначается символьным индексом N1.

Переменные, принадлежащие указанным выше типам, описываются обычным образом:

```
var
     Al: Vector;
    Din:   Table;
    Cl, C2, C3: Cub;
```

где Al, Din, Cl, C2, C3 - идентификаторы переменных.

6. Запись и чтение в Паскале

Процедуры записи Write, Writeln

Вывод в Паскале выполняется с помощью встроенной процедуры, которая вызывается директивой вида

Write (U, V) где аргументы U, V - это выражения типа integer, byte, real и т.д.

Программа приветствия:

```
 \\Приветствие
      begin
           Write ('Привет!');
           Write ('Как дела?');
      end.
```

Наберите эту программу в окне редактора PascalABC и затем запустите ее, нажав зелёный треугольник.

Для вывода в две строки нужно вместо первого Write использовать Writeln.

Если процедура Writeln не содержит никаких аргументов, то она осуществляет просто перевод строки.

Форматы вывода

Чтобы результаты, выводимые программой на экран, имели упорядоченный вид, нужно научиться управлять выводом. Прежде всего, отметим, что в процедурах Write и Writeln можно регулировать *ширину поля вывода.* Значение ширины задается целым числом через двоеточие после аргумента, например, Write (I:20). В результате переменная I будет выведена в поле шириной 20, начиная с крайней правой позиции. Если же ширину поля не задавать, то есть записать как обычно Write (I), значение переменной будет выведено в текущую позицию курсора.

Допустим, вам нужно получить на экране список значений целых чисел, выровненных по разряду единиц:

```
  123
   12
-3467
```

67891234

Пусть этот список представляет собой значения переменных A1 - A4. Тогда для его получения запишите последовательность инструкций вида:

writeln (A1:9); writeln (A2:9); writeln (A3:9); writeln (A4:9);

Аналогичным образом вы можете регулировать вывод не только чисел, но и текста. Например, если в примере предыдущего пункта вы запишите

writeln ('Привет, Лариса!':20); writeln ('Как дела?':20);

то получите на экране две строки, выровненные по правому краю. Левые позиции будут заполнены соответствующим числом пробелов:

Привет, Лариса!
Как дела?

Когда в процедурах write (X) или writeln (X) аргумент X имеет тип real, на экран выводится число в десятичном представлении с плавающей точкой, например, 3.33333333333485E-01. Чтобы представлять числа в более удобном формате с фиксированной точкой, используйте выражения вида:

write (Y:p:q)

где p - общее число позиций, а q - число знаков после десятичной точки. Таким образом можно вывести число 13.579 с помощью выражения write (Y:6:3) или число -0.45678 с помощью write (Y:8:5). Если количества позиций, заданного значением p, не хватает для размещения выводимого числа, Pascal автоматически откроет новые позиции. Если вручную ввести дополнительные позиции (увеличить p), то число займет крайние правые позиции, а слева появятся пробелы.

Процедуры чтения Read, Readln

В Паскале для ввода данных предусмотрена процедура чтения read. С помощью этой процедуры, имеющей формат

read (U, V);

возможен ввод чисел, символов, строк и т.д. Данные набираются на клавиатуре как минимум через один пробел. После набора данных, которые высвечиваются на экране, нажимается клавиша Enter.

Вводимые данные должны соответствовать определенному для них типу. Если это соответствие будет нарушено (например, для переменной типа real будет введено значение типа char), то появится сообщение об ошибке.

В случае, когда в программе имеется несколько операторов read, данные для них будут набираться в одной строке. Переход на следующую строку произойдет, когда окончится текущая строка. Однако в Паскале предусмотрено считывание данных из отдельной строки с помощью процедуры readln. После считывания последнего значения из списка этой процедуры следующие данные будут считываться с начала новой строки. Например, в случае последовательности операторов

readln (X, Y);
readln (Z);

после набора с клавиатуры значений для X, Y курсор автоматически будет переведен на новую строку для ввода Z.

Правила записи текста программ в Паскале

С помощью приведенных выше простых программ вы изучили не только операторы ввода-вывода, но и освоили некоторые правила написания программного кода в Паскале. Перечислим их.

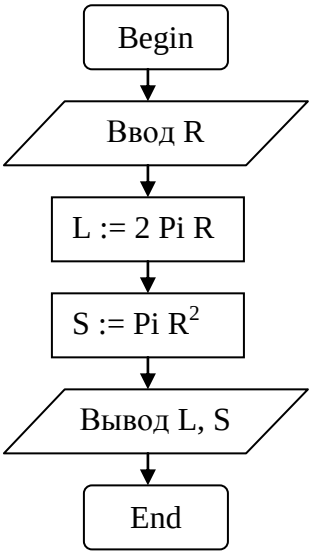
1. После каждого оператора (кроме begin и end) ставятся точка с запятой. - После слова begin ничего не ставится, а после слова end, означающего конец программы, нужно ставить точку.
2. Текстовая (строковая) переменная в Паскале заключается в апострофы ‘...’
3. Служебные слова (program, begin, end, integer, real и т.д.) обычно пишутся строчными буквами, а имена констант, переменных, процедур, функций начинаются с прописных букв.

4. В одной строке программы содержится один оператор. Логически подчиненные структуры записываются на одну позицию правее той структуры, которой они подчинены.

Эти правила написания программ не являются обязательными, то есть вы можете, например, набирать операторы прописными буквами или каждую строку программы начинать с крайней левой позиции. Кроме того, в одной строке может быть несколько операторов, разделенных точками с запятой. Однако читать и проверять такую программу будет неудобно. Поэтому рекомендуется пользоваться приведенными правилами, которые общеприняты среди программистов. В PascalABC существует возможность отформатировать текст готовой программы по этим правилам (*Сервис\ Форматировать код*).

Пример программы (линейный алгоритм) для вычисления длины окружности и площади круга. Радиус вводится с клавиатуры.

```
//Длина окружности, площадь круга
const
     Pi = 3.14159;
var
     R: byte;
     S, L: Real;
begin
    write ('введите радиус');
    readln (R);
    L:=2*Pi*R;
    S:=Pi*sqr(R);
     writeln ('длина окружности', L:6:2);
     writeln ('площадь круга', S:9:3)
end.
```



Практикум.

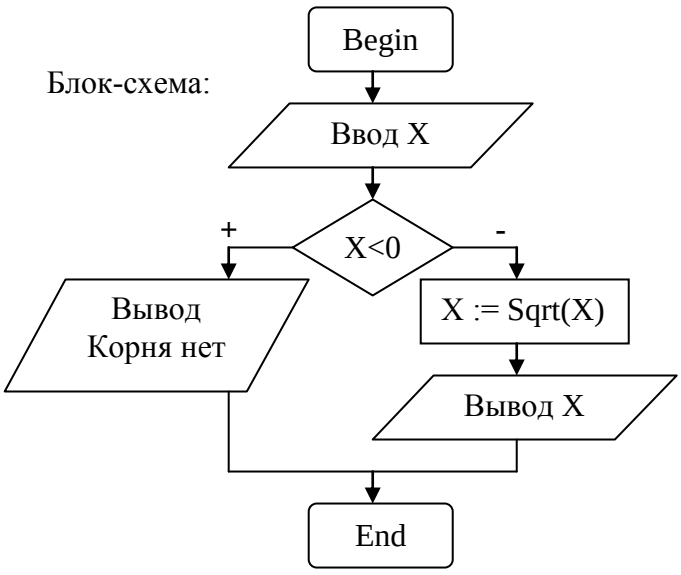
1. Вычислить гипотенузу прямоугольного треугольника, если известны его катеты.
2. Вычислить расстояние по формуле: $S = V_0t + at^2/2$

7. Операторы ветвления

Условный оператор if...then...else

Если необходимо выбрать одно из возможных действий в зависимости от некоторого условия, применяют условный оператор. Рассмотрим пример применения условного оператора при вычислении квадратного корня числа. Как известно, для получения действительного значения корня подкоренное выражение должно быть неотрицательным. В программе, кроме известных вам конструкций, используем встроенную функцию Sqrt (x) для вычисления квадратного корня и «операторные скобки» - служебные слова begin и end – т.к. в случае невыполнения условия необходимо выполнить две команды (вычисление корня и его печать).

Блок-схема:




```
//Квадратный корень
 var
    X: real;
 begin
```

```
        writeln ('Введите число X');
        read (X);
            if  X<0
            then
                 writeln ('Корня нет')
            else
                begin
                 X:=sqrt(X);
                 writeln ('Кв. корень равен', X:8:4)
                end
end.
```

Наберите эту программу и запустите на исполнение. Проверьте действие условия проверки знака подкоренного выражения, задавая отрицательные и положительные значения X.

Практикум.

1. Вычислить модуль числа m = |x|, используя его определение:

$$m = \begin{cases} x, \text{ если } x > 0, \\ -x, \text{ если } x < 0. \end{cases}$$

2. Вычислить значение функции $y = \frac{\sin x}{\sqrt{x^2 - 5}}$ с учетом области определения (если функция не существует, печатать соответствующий текст).

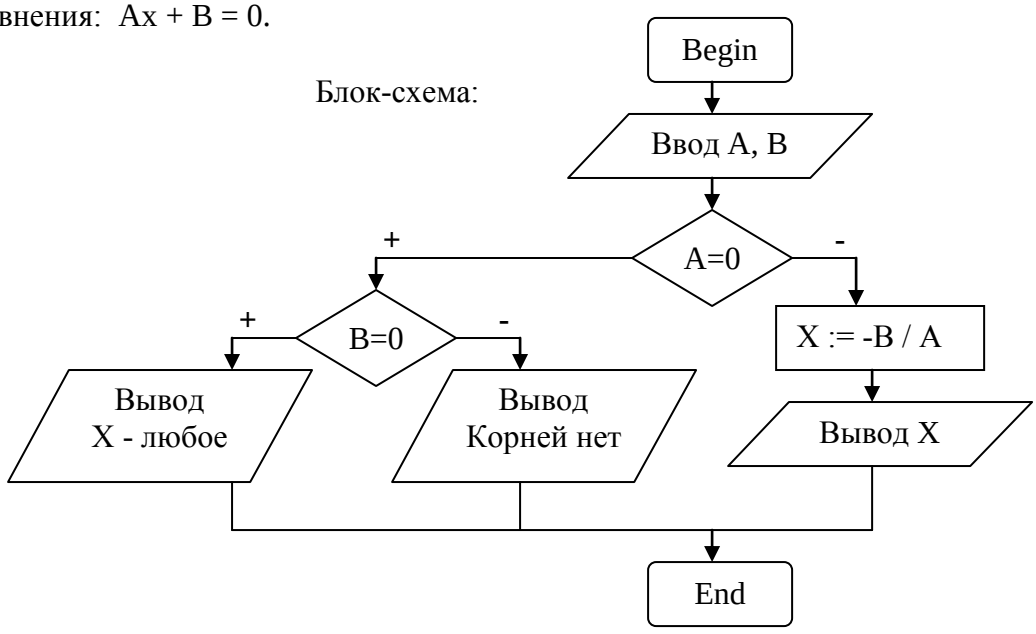
3. Вычислить значение функции для произвольного x:

$$y = \begin{cases} x^2 + \frac{1}{x}, \text{ если } x \neq 0, \\ \sin^2(x+2), \text{ если } x = 0. \end{cases}$$

Вложенные операторы

В некоторых задачах удобно применять вложенные условные операторы типа if ...then...if ...then...else...else или if ...then...else... if ...then ..else. Рассмотрим пример решения линейного уравнения: Ax + B = 0.

Блок-схема:



```
//Решение линейного уравнения
 var
    A, B, X: real;
begin
       writeln ('Введите коэффициенты A и B');
        read (A);
        read (B);
        if  A=0
        then  if  B=0
              then writeln ('X - любое')
              else  writeln ('Корня нет')
        else
             begin
               X := - B / A;
               writeln ('X =',  X:6:3)
             end;
end.
```

Оператор выбора case

Условный оператор if ...then обеспечивает ветвление только с двумя вариантами выбора. Для задания множественного ветвления используется более мощный оператор - оператор выбора. Этот оператор состоит из выражения (селектора) и списка вариантов:

```
case  <выражение>  of
<список 1>:   <оператор 1;>
<список 2>:   <оператор 2;>
. . .
<список  N>:   <оператор N>
else
<оператор>
end;
```

Схема работы оператора case такова. Сначала вычисляется значение селектора, следующего за словом case. Затем выполняется оператор с константой выбора, равной значению селектора. Если ни одна из констант не равна текущему значению селектора, то исполняется оператор, стоящий после слова else.

Часть else <оператор> в тексте программы можно опустить. Тогда, если среди констант селектора нужное значение отсутствует, выполнение оператора case ни к чему не приведет.

Примеры селекторов

Используем оператор case для вычисления функции

$Y = (1 + X + X^2)^N$, в которой степень N принимает целые значения (1, 2, 3).

```
var
     N: integer;
     X, Y: real;
begin
  writeln ('Введите значение X');
  readln (X);
  writeln ('Введите показатель степени от 1 до 3');
  readln (N);
  X:=1+X+X*X;
  case N of //Выбор варианта и вычисление селектора
       1: writeln ('Y=', X:6:3);
       2: writeln ('Y=', X*X:6:3);
```

```
            3: writeln ('Y=', X*X*X:6:3)
        else
            writeln ('Нет данных');
        end;
    end.
```

В этом примере на экран выводится запрос на ввод аргумента X и показателя степени N (значения селектора). В зависимости от него реализуется тот или иной вариант расчета функции X^N, например, для N = 2 - вариант, помеченный как 2. Если в качестве N в начале работы программы будет введено число, не значащееся среди констант выбора (например, 9), то на экран будет выведено сообщение «Нет данных».

Мы рассмотрели пример, когда каждому оператору в списке case предшествует только одна константа выбора (1, 2 или 3). Однако в общем случае перед каждым оператором может быть список констант (<список 1>, <список 2> в записи формата оператора case). Кроме того, может быть указан интервал изменения констант, который обозначается двумя точками «..».

Приведем пример селектора, содержащего списки констант выбора. Составим программу, с помощью которой можно было бы ввести номер месяца, а программа ответила бы, какому времени года соответствует этот месяц.

```
//Время года
var
   N: integer;
begin
  writeln ('Введите номер месяца');
  readln (N);
    case N of
       1,2,12: writeln ('Это зима');
       3..5:   writeln ('Это весна');
       6..8:   writeln ('Это лето');
       9..11:  writeln ('Это осень')
    else
        writeln ('Нет такого месяца')
    end;
end.
```

В качестве констант выбора могут использоваться целые числа (integer), символы (char), логические значения (boolean), а также пользовательский тип.

Практикум.

1. Вычислить корни квадратного уравнения, если заданы его коэффициенты.

Составим программу для нахождения корней квадратного уравнения $AX^2 + BX + C = 0$. Это уравнение является квадратным только при A, не равном 0. Для нахождения корней квадратного уравнения необходимо исследовать знак дискриминанта уравнения $D=B^2-4AC$. Если D>0, то имеются два различных корня, если же D<0, то действительных корней нет, если D=0, корни равны.

Программа может иметь следующий вид:

```
var
  A, B, C, D: real;
begin
    writeln ('Введите коэффициенты A, B, C');
    readln (A, B, C);
    if A=0
    then writeln ('Уравнение не квадратное')
    else
      begin
```

```
        D := B*B - 4*A*C; A := 2*A;
         if  D<0
         then writeln ('Действительных корней нет')
         else if D=0
              then writeln ('Корни равны: X1 = X2 = ', -B/A:6:3)
              else
                 begin
                  D := sqrt(D);
                  writeln ('X1 = ', (-B+D)/A:6:3);
                  writeln ('X2 = ', (-B-D)/A:6:3);
                 end;
      end;
  end.
```

Составьте блок-схему программы. Наберите программу и проверьте ее работу при различных наборах коэффициентов (A, B, C), например, (0, 1, 2); (0, 0, 5); (1, 1, -6) и т.д.

2. Определить тип треугольника, если заданы его стороны.

Треугольник не существует при невыполнении хотя бы одного из условий: a<b+c, b<a+c, c<a+b. (Из истинности этих условий следует также, что a>0, b>0 и c>0. Докажите!)

Если m – максимальная из длин сторон, а t и r – длины двух других сторон, то

- при $m^2 = t^2 + r^2$ – треугольник прямоугольный,
- при $m^2 < t^2 + r^2$ – треугольник остроугольный,
- при $m^2 > t^2 + r^2$ – треугольник тупоугольный.

Докажите!

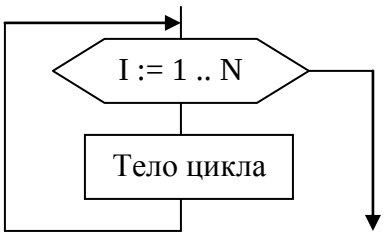
8. Операторы цикла

Оператор for (цикл со счетчиком)

Оператор for задает определенное число повторений и в Паскале имеет следующий формат:

for <параметр цикла>:= <P1> to <P2> do <тело цикла>;

где P1 и P2 - выражения для определения начального и конечного значений параметра цикла.



Часть оператора от слова for до слова do называется заголовком цикла. Тело цикла может быть представлено как простым, так и составным оператором (заключается в слова begin...end). Оператор тела цикла выполняется до тех пор, пока не будут перебраны все значения параметра цикла.

Возможно изменение параметра цикла как в сторону возрастания, так и в сторону убывания. Слово to означает, что параметр цикла меняется от начального до конечного значений в порядке возрастания. При каждом повторении параметр получает приращение +1. Чтобы параметр цикла убывал, нужно вместо to подставить слово downto (отвечает приращению -1).

Параметр цикла необходимо указать в разделе описаний программы (или текущего блока). Нельзя изменять значения параметра с помощью каких-либо присваиваний в теле цикла. После выполнения цикла параметр цикла становится неопределенным, и его идентификатор можно использовать в других операторах, в том числе - в новых циклах.

Как пример действия оператора for приведем программу вычисления значений функции

$$Y = e^{-|X|}$$

при различных значениях X в интервале от X_1 до X_2. Считаем, что аргумент X «пробегает» 10 значений, которые расположены в интервале от X_1 до X_2. Начальное и конечное значения

X_1 и X_2 задаются пользователем, а промежуточные значения X программа находит по формуле:

$$X = X_1 + (X_2-X_1)*(I-1)/9$$

где I - параметр цикла. При I=1 аргумент X равен X_1, а при I = 10 имеем $X=X_2$.

В программе мы используем встроенные функции Exp(X) и Abs(X), которые выполняют вычисление экспоненциальной функции и нахождение модуля соответственно. Программа будет иметь следующий вид:

```
var
I: integer;
X, Y, X1, X2: real;
begin
      writeln ('Значения функции Exp(-|X|)');
      writeln;
      writeln ('Введите интервал для аргумента: X1, X2');
      readln (X1, X2);
      for I:=1 to 10 do
        begin
          X:=X1+(X2-X1)*(I-1)/9;
          Y:=Exp(-Abs(X));
         writeln (Y)
        end
  end.
```

В разделе описания переменных указаны параметр цикла I (тип integer), а также переменные аргумента X и функции Y, границы интервала X1 и X2 (тип real). В начале выполнения программы на экране появляется надпись «Значения функции Exp (-Abs (x))», а затем в процессе циклических вычислений выводится столбик значений Y.

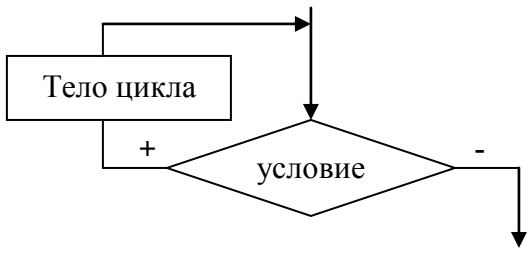
Практикум.

Протабулировать функцию $y = x^2 + 2x$ на заданном интервале в заданном количестве точек.

Оператор while («цикл Пока»)

Оператор цикла while является оператором цикла с предусловием («цикл Пока»), поскольку проверка условия производится при каждом повторении перед выполнением тела цикла. Оператор while имеет следующий формат записи:

```
while <условие> do
<тело цикла>
```



Условие представляет собой логическое выражение, а тело цикла - оператор, который может быть простым или составным. Перед каждым выполнением тела цикла вычисляется значение условия. Если значением будет True, цикл выполняется и снова вычисляется условие. Так повторяется до тех пор, пока условие не даст значение False и не произойдет выход из цикла и передача управления следующему оператору программы.

Приведем пример использования оператора while. Найдем с его помощью остаток от деления двух целых чисел A и B, не прибегая при этом к операции mod.

```
//Остаток от деления
 var
    A, B, X: integer;
    begin
    writeln ('Введите A, B');
```

```
        readln (A, B);
        X:=A;
        while X>=B do
           X := X - B;
        writeln ('Остаток от деления равен ', X);
        writeln
     end.
```

В качестве тела цикла здесь использован простой оператор X: =X-B.

Оператор while позволяет составлять компактные программы для вычисления различных сумм последовательностей чисел. Пусть требуется вычислить сумму ряда $S = X + X^2 + ... + X^N$ при произвольном значении X. Запишем для этого программу:

```
var
  I, N: integer;
  X, Y, S: real;
begin
writeln ('Введите X, N');
readln (X, N);
 I:=l;
 Y:=l;
 S:=0;
while K=N do
  begin
   Y:=Y*X;
    S:=S+Y;
     I:=I + 1;
  end;
writeln ('Сумма равна: ', S);
writeln
end.
```

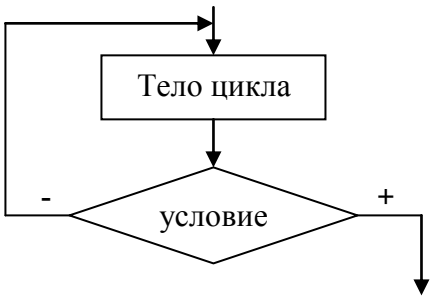
Обратите внимание, что работой цикла while «дирижирует» оператор I:=I+1, называемый счетчиком циклов. Пока значение счетчика не превышает N, цикл продолжается. Значение счетчика, равное N+1, прекращает работу оператора while.

Практикум.

1. Протабулировать функцию $y = x^2 + 2x$ на заданном интервале в заданном количестве точек.
2. Даны целое положительное число n и последовательность целых чисел a_1, a_2, .. a_n. Найти те члены a_i последовательности, которые
   а) являются четными числами,
   б) являются удвоенными нечетными числами,
   в) являются степенями двойки.
3. Дано целое число n>1. Вычислить:
   а) 3^n,
   б) n!,
   в) $\sqrt{n+\sqrt{(n-1)+...+\sqrt{2+\sqrt{1}}}}$.

Оператор repeat («цикл До»)

Еще одну конструкцию цикла представляет оператор repeat, который записывается в виде:




```
repeat
<тело цикла>
until <условие окончания цикла>
```

Этот оператор организует циклические вычисления таким образом, что условие проверяется после очередного выполнения тела цикла. Если выражение условия принимает значение True, повторения прекращаются. Поэтому оператор repeat называют еще циклом с постусловием или просто - «циклом До» (работа цикла продолжается до тех пор, пока не выполнено условие завершения).

Оператор repeat (как и оператор while) позволяет выполнять повторяющиеся действия, когда число повторений заранее не известно. Рассмотрим как пример убывающую геометрическую прогрессию (q<l):

$1, q, q^2, \ldots q^N, \ldots$

Как известно, сумма членов этой бесконечной прогрессии равна 1 / (1-q). Составим программу, с помощью которой можно было бы проверить эту формулу на конкретных примерах.

Построим алгоритм таким образом, чтобы вычисление суммы прекращалось, как только последний член в этой сумме будет меньше некоторой заранее заданной величины E (например, E=0.0001). В следующей программе выполняется ввод значений q, E и суммирование членов прогрессии с помощью оператора repeat:

```
var
    Q, Y, S, E: real;
begin
writeln ('Введите Q');
readln (Q);
writeln ('Введите границу для последнего учитываемого члена прогрессии');
readln (E);
Y:=1;
S:=0;
repeat
   begin
   Y:=Y*Q;
   S:=S+Y;
   end;
until abs(Y)<=E;
writeln ('Сумма фактически = ', S:8:6, 'По формуле = ', 1/(1-Q):8:6);
end.
```

9. Подпрограммы в Паскале

При написании больших программ широко используется структурный подход, называемый также *структурным программированием*. Программы, написанные с использованием структурного подхода, удобны в наладке и эксплуатации, а также легки для понимания. Это возможно благодаря тому, что большая программа разбивается на относительно самостоятельные модули (вспомогательные алгоритмы). Такие модули можно разрабатывать и отлаживать независимо один от другого. Поскольку независимые модули являются самостоятельным исполняемым кодом, они называются *подпрограммами.* Подпрограммы в языке Паскаль реализованы в виде процедур и функций.

Процедуры. Формальные и фактические параметры

Большие возможности повышения эффективности программирования на Паскале заложены в использовании процедур.

> ***Процедура*** *- это независимая именованная часть программы, которую можно вызвать из любой точки программы для выполнения определенных действий.*

Чтобы к процедуре можно было обратиться из основной программы, она должна быть описана. Описание процедуры размещается в программе после раздела описания переменных, в состав этого описания входят заголовок и тело процедуры.

Заголовок процедуры состоит из служебного слова procedure, идентификатора процедуры и списка формальных параметров, заключенного в круглые скобки (список параметров не обязателен). Элементы списка параметров отделяются друг от друга запятыми. Структура тела процедуры аналогична структуре программы, то есть включает в себя описания меток, констант, типов, переменных, а также раздел операторов:

```
procedure <имя> (<входные  параметры>; <var выходные параметры>);
const  ...;
type  ...;
var  ...;
begin
  <Операторы>
end;
```

Рассмотрим в качестве примера процедуру вычисления дискриминанта квадратного уравнения:

```
procedure Discr (A,B,C: real; var D: real);
begin
  D := B*B - 4*A*C
end;
```

В этой процедуре использованы переменные A, B, C, D типа real, которые являются формальными параметрами (A, B, C – входные, D - выходной). При вызове процедуры из текста программы вместо формальных подставляются фактические параметры, при этом соблюдаются следующие правила:

- Соответствие между параметрами команды вызова и формальными параметрами процедуры устанавливается не по именам, а по порядку следования: первый фактический параметр соответствует первой переменной, записанной в заголовке процедуры, второй фактический параметр – второй переменной и т.д.
- В качестве фактических параметров (аргументов) могут использоваться не только имена переменных или табличных величин, но и константы, арифметические выражения.
- Типы соответствующих параметров команды вызова и заголовка процедуры должны совпадать.

Команда вызова процедуры выполняется в три этапа: 1) вычисление фактических аргументов; 2) исполнение алгоритма-процедуры; 3) присвоение полученных значений результатов алгоритма-процедуры соответствующим фактическим переменным.

Примеры процедур

Оформим в виде процедуры программу приветствия. Ниже приведена запись процедуры и текст основной программы, из которой выполняются обращения к процедуре.

```
procedure Hello (Name: string);
begin
Writeln ('Привет,', Name, '!');
Writeln (Name, ', как дела?');
Writeln
end;
begin
  Hello ('Катя');
  Hello ('Андрей');
  Hello ('Лена')
end.
```

Эта программа выводит на экран приветствия для трех человек: Кати, Андрея и Лены. Символьная величина Name - входной параметр, выходных параметров нет.

Рассмотрим еще один пример. В некоторых расчетах приходится пользоваться функциями гиперболических синуса и косинуса: $sh(x) = ½ (e^{x} – e^{-x})$, $ch(x) = ½ (e^{x} + e^{-x})$.

Составим программу вычисления, например, выражений

$Fl=sh^2x+chx$ и $F2= shx+ch^2x$

с использованием процедуры расчета гиперболических функций.

```
var
   X: real;
   Cx, Sx, Fl, F2: real;
procedure Hpb (Y: real; var Ch, Sh: real);
   begin
   Y := Exp (Y);
   Ch := (Y+l/Y)/2;
   Sh := (Y-l/Y)/2
   end;
begin
writeln ('Введите аргумент X');
readln (X);
Hpb (X, Cx, Sx);
Fl := Sx*Sx + Cx;
F2 := Sx + Cx*Cx;
writeln ('Значение функции Fl равно', Fl);
writeln ('Значение функции F2 равно', F2)
end.
```

Функции

В структурном программировании наряду с процедурами широко применяются функции. Отличие их от процедур заключается в том, что результатом выполнения функции является некоторое единственное значение. Это исчисляемое значение присваивается идентификатору функции. Структура описания функции аналогична описанию процедуры, и подробно останавливаться на ней мы не будем. Приведем пример описания и использования функции при расчете выражения $y = |x + 1| + |x + 2|$.

Вычисление модуля оформим в виде пользовательской функции Modul:

```
var
  X, Y: real;
function Modul (X: real): real;
  begin
  if  X<0  then  X := -X;
  Modul:=X
  end;
begin
  writeln ('Введите X');
  readln (X);
  Y := Modul (X+1)+ Modul (X+2);
  writeln (Y:6:3)
end.
```

В следующем примере показано вычисление целой степени числа, то есть степенной функции $Y=X^N$. Этой функции нет среди встроенных функций Паскаля, поэтому для ее вычисления вводим пользовательскую функцию Deg. Непосредственно возведение числа в целую степень выполняем с помощью оператора цикла while:

```
var
  Z, F: real;
  M: integer;
function Deg (N: integer; X: real): real;
  var
    I: integer; Y: real;
  begin
  I:=1;
```

```
  Y:=1;
  while I<=N do
    begin
    Y:=Y*X;
    I:=I + 1
    end;
  Deg:=Y;
  end;
begin
  writeln ('Введите Z, M');
  readln (Z, M);
  F:=Deg (M, Z);
  writeln (F:8:3);
end.
```

Стандартные функции

Sin(x)	Синус
Cos(x)	Косинус
ArcTan(x)	Арктангенс
Sqr(x)	Квадрат
Sqrt(x)	Квадратный корень
Abs(x)	Модуль
Ln(x)	Натуральный логарифм
Exp(x)	e^x
Round(x)	Округление
Int(x)	Целая часть
Frac(x)	Дробная часть
Random(x)	Случайное число

10. Работа с символами и строками

Описание строк

Для работы с текстами в Паскале применяется структурированный тип string (строка). Напомним, что строка - это цепочка, составленная из символов. Символы берутся из кодовой страницы, поддерживаемой в компьютере. Строка похожа на одномерный массив, однако, в отличие от массива, количество элементов строки (символов) не фиксировано. Единственное ограничение на количество символов: оно не может превышать 255. Для строки длиной N отводится в памяти (N+1) байт (N байт - для хранения символов и 1 байт - для записи длины строки).

Строковые переменные должны быть определены в разделе описания переменных, например:

```
var
    Textl: string;
    Tl: string [20];
    T2: string [125];
```

где в квадратных скобках указывается максимальная длина строки. Если длина строки не указана, то принимается длина по умолчанию - 255 символов. Можно задать строковый тип, например:

```
type
  Atten = string [100];
Var
  Bl: Atten;
```

Строковые константы определяются текстом, заключенным в кавычки, например:

```
Const
```

Name = 'Константин';

К любому символу строки можно обратиться по его номеру (например, Atten [7]) - аналогично обращению к элементу одномерного массива. При этом нужно учесть, что первый байт строки имеет номер 0 и содержит значение длины строки. Второй байт с номером 1 содержит первый символ строки.

Операции над строками

Над строками возможны простейшие операции, которые позволяют составлять строковые выражения. Прежде всего, это операция *сложения* (называемая также *конкатенацией),* которая заключается в объединении двух слов без пробела, например, выражение 'Прилуки, ' + 'Ужгород' будет давать текст: Прилуки, Ужгород.

Другими операциями над строками являются *операции отношения* (=, <, >, >=, <=) которые проводят сравнение двух строк текста. Сравнение выполняется посимвольно слева направо до первого несовпадающего символа. Большим значением считается то, в котором первый несовпадающий символ имеет больший номер в алфавите. Строки считаются равными, если каждый символ одной строки совпадает с каждым символом другой строки в порядке следования символов. Результат операции отношения имеет булевский тип, например, выражения 'абзац' < 'абонент'; 'depend' >= 'dependence' имеют соответственно значения True и False.

Присвоение значения строковой переменной можно выполнить с помощью оператора присваивания:

T1:='Сегодня температура '; T2:=T1 + '-10 градусов';

В одном выражении можно записывать переменные как строкового, так и символьного типов.

Часто полезными операциями оказываются преобразования данных символьного типа char в целое число - код ASCII, а также обратное преобразование. Преобразование символа в число выполняется функцией Ord, а обратное преобразование - функцией Chr.

Запишем простейшую программу перевода символов, вводимых с клавиатуры, в числовой код:

```
var
   Xsymbol: char;
begin
 write ('Введите символ: ');
 readln (Xsymbol);
 writeln ('Символ  ', Xsymbol, 'имеет код', ord (Xsymbol));
end.
```

При вводе символов числовой код может принимать значения от 0 до 255.

Обработка строк с помощью процедур и функций

В Паскале имеется набор стандартных процедур и функций для работы со строками. Рассмотрим некоторые из этих процедур и функций и их действие на примере следующих строковых констант:

```
const
S1:= 'обитатель ';
S2 := 'лесов ';
S3 := 'кабан ';
```

concat (A, B, .. Z) - функция строкового типа, которая возвращает значение сцепленных строк A, B,.. Z. Исходные строки в новой строке следуют в том порядке, в каком они указаны в списке параметров. Приведем примеры с использованием определенных выше строковых констант:

Выражение	**Результат**
concat (SI, S2, S3);	'обитатель лесов кабан '
concat (S3, SI, S2);	'кабан обитатель лесов '

delete (T, Pos, N) - процедура удаления N символов в строке T, начиная с позиции под номером Pos. Например:

Выражение	Результат
delete (S1, 4, 2);	'обитель'
delete (S3, 1, 2);	'кан '

insert (T, S, Pos) - процедура вставки строки T в строку S, начиная с позиции Pos. Приведем пример действия процедуры вставки:

Выражение	Результат
insert (S3, S1, 6);	'обитакабантель '

copy (T, Pos, N) - функция строкового типа, которая возвращает значение подстроки длиной N, начиная с позиции Pos в исходной строке T. Например:

Выражение	Результат
copy (S1, 3, 2);	'ит'
copy (S2, 2, 3);	'аба'

Length (T) - функция целого типа, которая возвращает значение длины строки T, например:

Выражение	Результат
length (S1);	9
length (S2);	5
length (S3);	5

pos (Tl,T2) - функция целого типа, которая обнаруживает в строке T2 первое появление подстроки T1. Функция возвращает номер позиции, в которой находится первый символ подстроки T1. Если подстрока T1 не найдена, то результатом будет 0. Рассмотрим действие этой функции на примере двух строк: 'такт' (строка T1) и 'бестактность' (строка T2). Выражение pos (Tl, T2) будет иметь результатом число 4.

Как пример использования строковых процедур составим программу, которая будет выполнять последовательные преобразования слов «холестерин - холерик - болеро».

```
var
   S1, S2: string;
begin
   S1 := 'холестерин';
   writeln (S1);
   delete (S1,5,3);
   delete (S1,7,1);
   S2:=S1+'к';
   writeln (S2);
   S2:=copy (S2,2,4);
   S2:= 'б' + S2 +'о';
   writeln (S2);
end.
```

11. Работа с табличными величинами

Ранее вы познакомились с табличными величинами, которые в большинстве языков программирования представляются с помощью одномерных или двумерных массивов. Существует множество задач обработки информации, заданной в форме таблиц. К таким задачам относятся:

- изменение элементов таблицы (редактирование таблицы, математические действия над элементами и т.д.);
- поиск элементов, удовлетворяющих определенному критерию (максимальных и минимальных значений, фрагментов текста и т.д.);
- обработка значений таблицы (нахождение сумм и произведений элементов и т.д.);

- сортировка таблиц, то есть упорядочение элементов по возрастанию (убыванию).

В настоящем параграфе мы рассмотрим примеры таких задач.

Формирование таблиц

Массивы, как известно, упорядочены таким образом, что каждому элементу соответствует определенная совокупность индексов (линейным таблицам - один индекс, прямоугольным таблицам - два индекса). Доступ к каждому элементу осуществляется заданием индексов.

Массив в Паскале описывается словосочетанием *array of*. При этом можно задавать тип, например, с помощью конструкции:

```
type
  Table1 = array [1...10] of integer;
var
  A1, B2: Table1;
```

Но можно описывать массив и без представления типа, например,

```
var
   Table: array [1..10] of integer;
```

Таким образом, описаны одномерные массивы Al и B2 типа Table1 и массив Table. Массивы представляют собой линейные таблицы, которые составлены из 10 элементов, принимающих целые значения.

Присвоение начальных значений элементам массива называется *инициализацией.* Инициализировать массив можно с помощью оператора цикла либо путем поэлементного ввода значений. Так, таблица 5x10 с единичными значениями всех элементов A[I, J] инициализируется вложенными операторами for:

```
for I:=l to 5 do for J:=l to 10 do A[I, J] := 1;
```

В случае поэлементного задания таблицы путем ввода с клавиатуры обычно используется оператор read или readln, например, read ([2,4]) или readln ([1,8]). Эти операторы могут размещаться внутри тела цикла, если необходим ввод с клавиатуры каждого элемента.

Вывод элементов таблицы на экран проще всего осуществлять с помощью оператора write или writeln, например,

```
for I := 1 to 5 do
for J:=l to 10 do
writeln (A[I,J]);
```

Рассмотрим пример формирования линейной таблицы. Допустим, вы создаете путем ввода с клавиатуры таблицу значений суточной температуры, наблюдавшейся летом на протяжении недели. Программа определяет максимальную температуру за неделю и выводит ее на экран. Код программы может быть следующим:

```
var
   Temp: array [1. .7] of  real;
   I: integer;
   Tmax: real;
begin
  Tmax := 0;
for I := l to 7 do
   begin
      writeln ('Введите температуру ', I, ' - го дня недели');
      readln (Temp[I]);
      if  Tmax < Temp [I] then Tmax := Temp[I];
   end;
writeln ('Максимальная температура равна ', Tmax:5:2);
end.
```

Наберите этот код и запустите программу на счет. Программа последовательно предложит вам ввести температуры для каждого из 7 дней. Вы можете вводить не только

целые числа, но и десятичные с фиксированной точкой (два знака после точки). После ввода последнего числа вы увидите максимальное значение температуры за неделю.

Нахождение произведения и суммы элементов таблицы

Вычисление сумм и произведений элементов таблиц рассмотрим на следующем примере. Допустим, имеется двумерная таблица, состоящая из 3 столбцов и N строк, причем в столбцы 1 и 2 введены вещественные числа. Требуется найти произведение элементов столбцов 1, 2 и записать произведение в столбец 3. Нужно также вычислить сумму элементов столбца 3. Подобные операции встречаются при заполнении накладных, когда количество товара (столбец 1) умножается на его цену (столбец 2) и вносится в столбец стоимости товара (столбец 3). Суммирование стоимости товара каждого наименования (элементов столбца 3) дает общую сумму накладной.

В разделе описаний констант и переменных запишем:

```
const
  N = 20;
var
  Nakl: array [1.. .3, 1..N] of real;
  I: integer;
```

то есть, определим Nakl - двумерный массив, N - количество строк в массиве; i - индекс, зарезервированный для нумерации строк массива.

Умножение столбцов 1 и 2 и запись произведения в столбец 3 выполним с помощью оператора for:

```
for I:=l to N do
Nakl [3, I] := Nakl [1, I] * Nakl [2, I];
```

Сумму элементов столбца присвоим переменой S. Текст программы может быть записан так:

```
const
  N = 5;
var
  Nakl: array [1..3, 1..N] of  real;
  I, J: integer;
  S: real;
begin
for I:=l to N do
  begin
    writeln ('Введите количество товара   ', I);
    readln (Nakl [1, I]);
    writeln ('Введите цену товара ', I);
    readln (Nakl [2, I])
  end;
S := 0;
for I:=l to N do
  begin
    Nakl [3, I] := Nakl [l, I] * Nakl [2, I];
    S := S + Nakl [3,1];
  end;
writeln ('Общая сумма равна ', S: 6: 2);
end.
```

После запуска этой программы нужно последовательно ввести значения для первого и второго столбцов (количество и цена товара), после чего на экран будет выведено вычисленное значение суммы.

Поиск элементов в таблицах

Поиск значений табличных величин рассмотрим на простом примере. Допустим, задан одномерный массив A, составленный из N целых чисел;

```
const
   N = 10;
   A: array [1..N] of byte (3, 8, 7, 1, 8, 1, 4, 5, 8, 13);
```

Требуется найти первый элемент, имеющий заданное значение, и вывести номер этого элемента на экран, Значение искомого элемента обозначим X. Ввод значения искомого элемента оформим в виде приглашения:

```
   writeln ('Введите элемент для поиска ');
   readln (X);
```

Поиск нужного элемента будем выполнять путем циклического сравнения значений всех элементов массива со значением X, введенным с клавиатуры. Будем считать, что номера элементов массива начинаются с 1. Блок поиска запишем в виде:

```
for I := l to N do
   if A[I]=X then
      writeln ('Номер искомого элемента', I);
```

Поиск осуществляется с помощью условного оператора if A [I] = x then. Если условие A[I] = x выполняется, номер I выводится на экран. В целом код программы будет иметь вид:

```
const
   N = 10;
   A: array [1..N] of  byte = (3, 8, 7, 1, 8, 1, 4, 5, 8, 13);
var
   X, I: byte;
begin
   writeln ('Исходный   массив: ');
   for I := 1 to N do write (A[I],'       ');
   writeln;
   writeln ('Введите элемент для поиска');
   readln (X);
   for I := 1 to N do
      if A [I] = X then
         writeln ('Номер искомого элемента', I);
end.
```

Наберите код этого примера и сохраните его в виде файла на диске. Откомпилируйте программу и запустите программу на исполнение. Если в процессе исполнения вы введете значение для поиска 1, программа выдаст вам номера элементов: 4 и 6. Если же ввести значение 8, то программа укажет номера искомых элементов: 2, 5, 9. Исходный массив A[I] может формироваться различными способами: заданием значений в описании (как в примере), с помощью каких-либо функций, вводом значений элементов с клавиатуры. Нужно только в программе предусмотреть тот или иной способ ввода.

Сортировка таблиц

Рассмотрим простые алгоритмы упорядочения (сортировки) одномерных таблиц. Цель сортировки - облегчить последующий поиск элементов; выбор алгоритма сортировки зависит от структуры обрабатываемого списка. Критериями эффективности сортировки являются быстродействие и экономия памяти, что может быть особенно существенно в случае больших списков.

Метод прямого выбора

Допустим, вам нужно из исходной последовательности A[i], состоящей из N элементов, образовать убывающую последовательность (точнее, последовательность из невозрастающих элементов). Зафиксируем первый элемент и просмотрим остальной массив (N-1) элементов, отыскав в нем наибольший. Если этот элемент окажется больше первого, поменяем его с первым элементом местами. Затем зафиксируем элемент 2 и просмотрим оставшиеся (N-2) элемента. Найдя наибольший элемент, обменяем его с элементом 2.

Подобную процедуру будем продолжать до тех пор, пока не останется один, самый большой элемент.

Приведем программу, осуществляющую сортировку массива из 5 элементов методом прямого выбора (в качестве элементов взяты строки):

```
const
  Num = 5;
A: array[1.. Num] of string = ('ca', 'aa', 'd', ' a' ,'ab');
var
  Temp: string;
  I, J, L: integer;
begin
   writeln ('Начальный   массив');
   for I := l to Num do write ('    ', A[I]);
   writeln;
   writeln;
   for I := l to Num-1 do
     for J := I+1 to Num do
      begin
        if  A[I] < A[J] then
          begin
          Temp := A[I];
          A[I] := A[J];
          A[J] := Temp
       end;
   for L := l to Num do write  ('    ', A[L]);
   writeln
   end
end.
```

Процесс сортировки в этом примере проиллюстрируем выводом получаемой последовательности элементов после каждой операции сравнения.

```
ca  aa  d   a   ab
d   aa  ca  a   ab
d   aa  ca  a   ab
d   aa  ca  a   ab
d   ca  aa  a   ab
d   ca  aa  a   ab
d   ca  aa  a   ab
d   ca  aa  a   ab
d   ca  ab  a   aa
d   ca  ab  aa  a
```

Нетрудно подсчитать, что количество операций сравнения в методе прямого выбора будет равно числу сочетаний из Nmax по 2, то есть Nmax! / (2! (Nmax - 2)!) , где Nmax - размер исходного массива.

Метод пузырьков

Этот метод сортировки своим алгоритмом напоминает «всплывание» в процессе вычислений более «легких» элементов. Последовательность элементов просматривается от начала к концу (например, слева направо). При этом сравниваются пары соседних элементов. Если элемент справа оказывается больше элемента слева, то они обмениваются местами. Запишем программу сортировки того же массива, что и в предыдущем примере:

```
const
  Num = 5;
```

```
A: array [1..Num] of string = ('ca', 'aa', 'd', 'a', 'ab');
var
  Temp: string;
  I, J, L: integer;
begin
  writeln ('Начальный   массив');
  for I := l to Num do write ('    ', A[I]);
  writeln;
  for I := 2 to Num do
    begin
    for J := Num downto I do
      begin
      if A[J-1]<A[J] then
        begin
        Temp := A[J-l];
        A[J-1] := A[J];
        A[J] := Temp;
        for L :=l to Num do write ('    ', A[L]);
        writeln;
       end;
     end;
   end;
end.
```

Пузырьковый метод сортировки несколько эффективнее метода прямого выбора, поскольку приводит к результату за меньшее число перестановок.

ca	aa	d	a	ab
ca	aa	d	ab	a
ca	d	aa	ab	a
d	ca	ab	aa	a

Кроме рассмотренных нами методов сортировки, существуют также ускоренные методы, например, метод последовательного дробления массива на части.

Распечатка двумерного массива

Заполнить двумерный массив 5x10 целыми случайными числами из интервала от -20 до 30 и распечатать его в виде таблицы.

```
var i, j: integer;
   a: array[1..5, 1..10] of integer;
 begin
 randomize;
 for i := 1 to 5 do
 for j:= 1 to 10 do
  a[i, j] := round (random (50) - 20); //заполнение массива
for i:=1 to 5 do
begin
   for j := 1 to 10 do
   write (a[i, j]:6); //распечатка массива
writeln;
end;
readln
end.
```

Практикум.

Одномерный массив из 10 элементов состоит из случайных целых чисел из интервала от –10 до 10. Определить, сколько элементов массива меньше заданного числа.

```
var n, k, i: integer;
   a: array[1..10] of integer;
begin
randomize; //подключение генератора случайных чисел
for i:=1 to 10 do
    begin
    a[i] := round (random(20)-10); //заполнение массива
    writeln (a[i]) //распечатка массива
    end;
writeln ('введите число');
readln (k);
n := 0;
for i:=1 to 10 do
  if a[i]<k then n := n+1;
writeln (n)
end.
```

12. Работа с графикой

Подключение графического режима:

```
uses graphABC;
```

Размер графического экрана устанавливается с помощью процедуры SetWindowSize(w,h), где w и h – ширина и высота окна. Значения координат принимают только целочисленные значения: x от 0 до w, а y от 0 до h.

Расположение осей координат:

Основные операторы графики

putpixel (x, y, c) ставит точку с координатами (x, y) и цветом c.

line (x1, y1, x2, y2) рисует отрезок от точки (x1, y1) до точки (x2, y2).

circle (x, y, R) рисует окружность с центром в точке (x, y) и радиусом R.

rectangle (x1, y1, x2, y2) прямоугольник с вершинами (x1, y1) и (x2, y2), лежащими на диагонали.

Пример 1. Составить программу, которая рисует на экране графические примитивы: отрезок, прямоугольник, круг, эллипс.

Прежде всего подключим модуль graphABC. Зададим размеры графического окна setWindowSize(600,240); - ширина 600, а высота 240 пикселей.

Запишем процедуру рисования отрезка, соединяющего точки с координатами (80,40) и (500,40):

```
line(80,40,500,40);
```

Поскольку параметры пера не заданы, то по умолчанию его толщина 1 пиксель, цвет черный. Зададим толщину пера 5 пикселей setPenWidth(5); Это значение не изменится до тех пор, пока не будет задано новое. Таким образом, дальше все примитивы рисуются пером такой толщины. Для каждого примитива будем задавать цвет пера и кисти, например, для прямоугольника коричневый цвет пера setPenColor(clBrown); и желтый цвет кисти setBrushColor(clYellow);

Аналогично нарисуем круг (задаются координаты центра и радиус) и эллипс (задаются координаты противоположных вершин прямоугольника, описывающего эллипс).

Наконец, зададим параметры текста: размер символов 15 пикселей setFontSize(15); начертание жирное setFontStyle(fsBold); цвет коричневый setFontColor(clBrown); цвет фона (кисти) белый setBrushColor(clWhite).

Программа может выглядеть так:

```
uses graphABC;  //подключение модуля graphABC
begin
setWindowSize(600,240);  // размеры окна вывода
line(80,40,500,40);  // отрезок
setPenWidth(5);  // толщина пера
setPenColor(clBrown); setBrushColor(clYellow);  // цвет пера и кисти
rectangle(40,80,200,160);  // прямоугольник
setPenColor(clRed); setBrushColor(clGreen);
circle(300,120,40);  // круг
setPenColor(clBlue); setBrushColor(clRed);
ellipse(400,80,540,160);  // эллипс setFontSize(15);
setFontStyle(fsBold);  // размер и начертание шрифта
setFontColor(clBrown); setBrushColor(clWhite);
textOut(120,180,'Графические примитивы');   // вывод текста
end.
```

13. Работа с файлами

Задача нахождения суммы двух чисел (X и Y):

```
 var x, y: integer;
       fin, fout: text;
begin
  assign (fin, ‘input.txt’);
  reset (fin);
  assign (fout, ‘output.txt’);
  rewrite (fout);
  readln (fin, x);
  readln (fin, y);
  writeln (fout, x+y);
  close (fin); close (fout)
end.
```

fin – обозначение файла ввода данных input.txt
fout – обозначение файла вывода данных output.txt
assign () – команда присвоения имени файлу
reset () – открыть для чтения
rewrite () – открыть для записи
clouse () – закрыть

Файлы input.txt и output.txt должны находиться в той же папке, куда сохранён файл программы. Числа x и y в файле input.txt должны быть на разных строчках. Сумма этих чисел будет записана в файл output.txt.

14. Загрузка растрового изображения. Анимация

Пусть кораблик перемещается на фоне моря слева направо и вверх на расстояние 600 пикселей.

Загрузим изображения фона и корабликая из файлов flow.jpg и fly.jpg, поместив их описатели в переменные fon и bub. Установим прозрачность фона для изображения автомобиля. Зададим начальные координаты (x,y), ширину w и высоту h изображения автомобиля. Все переменные имеют тип integer. Процедуры рисования и стирания будем повторять в цикле с предусловием While до тех пор, пока автомобиль не переместится на 600 пикселей. На каждом шаге цикла координату x левого верхнего угла изображения увеличиваем на 10, а y уменьшаем на 1. Ширину и высоту уменьшаем на 2 пикселя для уменьшения изображения при удалении.

Заметим, что плавность и длительность полученной демонстрации зависит от выбора шага, количества кадров (повторений цикла), времени показа кадра (задержки), а также быстродействия компьютера.

Программа может выглядеть так:

```
uses GraphABC;
var  x, y, w, h : integer;
fon:= new Picture('flow.jpg');  //загрузка изображения фона
bub:= new Picture('fly.jpg');  //загрузка изображения кораблика
begin
   SetWindowSize(676,507);
   bub.Transparent:=true;  //прозрачный фон кораблика
   x:=0; y:=270; w:=184; h:=200;
   While x<600 do
   begin
      ClearWindow;
      fon.Draw(0,0);  //вывод изображения
      bub.Draw(x,y,w,h);  //вывод изображения, масштабируя его к размеру (w, h)
      x:=x+10; y:=y-1; w:=w-2; h:=h-2;  //изменение координат кораблика и масштаба
      sleep(40); Redraw;  //задержка 40 мс и перерисовка
   end;
end.
```

Приложение

Графические примитивы модуля GraphABC

SetPixel(x,y,color: integer); Закрашивает один пиксел с координатами **(x, y)** цветом **color**.

MoveTo(x,y: integer); Передвигает невидимое перо к точке с координатами (**x, y)**; работает в паре с функцией **LineTo(x, y)**.

LineTo(x,y: integer); Рисует отрезок от текущего положения пера до точки **(x, y)**; координаты пера при этом также становятся равными **(x, y)**.

Line(x1,y1,x2,y2: integer); Рисует отрезок с началом в точке **(x1, y1)** и концом в точке **(x2, y2**).

Circle(x,y,r: integer); Рисует окружность с центром в точке **(x, y)** радиусом **r**.

Ellipse(x1,y1,x2,y2: integer); Рисует эллипс, заданный описанным прямоугольником с координатами противоположных вершин **(x1,y1)** и **(x2,y2)**.

Rectangle(x1,y1,x2,y2: integer); Рисует прямоугольник, заданный координатами противоположных вершин **(x1,y1)** и **(x2,y2)**.

Arc(x,y,r,a1,a2: integer); Рисует дугу окружности с центром в точке **(x,y)** и радиусом **r**, заключенную между двумя лучами, образующими углы **a1** и **a2** с осью OX **(a1 и a2** – вещественные, задаются в градусах и отсчитываются против часовой стрелки).

Pie(x,y,r,a1,a2: integer); Рисует сектор круга, ограниченный дугой (параметры процедуры имеют тот же смысл, что и в процедуре **Arc)**.

FloodFill(x,y,color: integer); Заливает однотонную область цветом **color**, начиная с точки **(x, y)**.
TextOut(x,y: integer; s: string); Выводит строку **s** в позицию **(x,y)** (точка **(x,y)** задает верхний левый угол прямоугольника, который будет содержать текст из строки **s**).

Константы стандартных цветов:
clBlack – черный
clPurple – фиолетовый
clWhite – белый
clMaroon – темно-красный
clRed – красный
clNavy – темно-синий
clGreen – зеленый
clBrown – коричневый
clBlue – синий
clSkyBlue – голубой
clYellow – желтый
clAqua – бирюзовый
clCream – кремовый
clOlive – оливковый
clFuchsia – сиреневый
clTeal – сине-зеленый
clGray – темно-серый
clLime – ярко-зеленый
clLtGray – светло-серый
clDarkGray – темно-серый
clMedGray – серый
clSilver – серебряный

Действия с графическим окном
SetWindowSize(w,h: integer); Устанавливает ширину и высоту графического окна.
SetWindowCaption(s: string); Устанавливает заголовок графического окна.
SaveWindow(fname: string); Сохраняет содержимое графического окна в файл с именем **fname**.
LoadWindow(fname: string); Выводит в графическое окно рисунок из файла с именем **fname**.
ClearWindow; Очищает графическое окно белым цветом.
ClearWindow(c: ColorType); Очищает графическое окно цветом **c**.
Redraw; Осуществляет перерисовку окна

Действия с пером и кистью
SetPenColor(color: integer); Устанавливает цвет пера, задаваемый параметром **color**.
SetPenWidth(w: integer); Устанавливает толщину пера, равную **w** пикселам.
SetPenStyle(ps: integer); Устанавливает стиль пера, задаваемый параметром **ps**.
Стили пера задаются именованными константами:
psSolid, psClear, psDash, psDot, psDashDot, psDashDotDot
SetBrushColor(color: integer); Устанавливает цвет кисти, задаваемый параметром **color**.
SetBrushPicture(fname: string); Устанавливает в качестве образца для закраски кистью изображение, хранящееся в файле **fname**.
SetBrushStyle(bs: integer); Устанавливает стиль кисти, задаваемый параметром **bs**.
Стили кисти задаются именованными константами:
bsSolid, bsCross, bsClear, bsDiagCross, bsHorizontal, bsBDiagonal, bsVertical, bsFDiagonal.

Действия с рисунками

LoadPicture(fname: string): integer; загружает рисунок из файла с именем **fname** в оперативную память и возвращает описатель рисунка в целую переменную **n**:
n:=LoadPicture(fname)

DrawPicture(n,x,y: integer); Выводит рисунок с описателем **n** в позицию **(x,y)** графического окна.

DrawPicture(n,x,y,w,h: integer); Выводит рисунок с описателем **n** в позицию **(x,y)** графического окна, масштабируя его размеры к ширине **w** и высоте **h**.

SavePicture(n: integer; fname: string); Сохраняет рисунок с описателем **n** в файл с именем **fname** (форматы **bmp, jpg** или **gif**).

SetPictureSize(n,w,h: integer); Устанавливает размер рисунка с описателем **n** равным **w** на **h** пикселей.

SetPictureTransparent(n: integer; b: boolean); Устанавливает (**b=True**) или отключает (**b=False**) режим прозрачности при выводе изображения с описателем **n**.

Полезные ссылки:

- Основы алгоритмизации на Pascal: http://www.abc-it.lv/index.php/
- В.В. Ятис (Северный район НСО): http://kabinet-inform.narod.ru/p32.htm