Практическая работа № 30

Тема: ***Организация циклов в программе.***

Цель: Научиться составлять простейшие программы с использованием операторов цикла для решения финансовых, инженерных и научных задач (в среде Delphi).

Время: 60 мин.

Задание: Решить задачу согласно варианта - составить блок-схему алгоритма, написать программу, отладить её и выполнить на ЭВМ.
Задачу решить с использованием двух-трёх операторов цикла.

Литература:
1. Фаронов В.В. Программирование на языке высокого уровня: Учебник для вузов. – СПб.: Питер, 2003.
2. Бобровский С.И. Delphi 7. Учебный курс. – СПб.: Питер, 2005.

Содержание отчёта:
- Ответы на вопросы, поставленные в пунктах описания последовательности выполнения работы.
- Блок-схема алгоритма и текст программы.
- Результаты вычислений.
- Выводы по работе (что изучили, чему научились).

Варианты задания:

Варианты задач сведены в таблицу. Необходимо найти значения функции Y(x) для всех X, изменяющихся от **Xн** до **Xк** с шагом **h**. Вывод результатов оформить в виде таблицы с двумя столбцами и следующим заголовком:

Результаты вычислений.

Аргумент	**Функция**

№ варианта	**Функция Y(x)**	**Xн**	**Xк**	**h**
1	$e^x + \sqrt{x}$	0	5	0.5
2	$\sqrt[3]{x} + \sin^2 x$	π/2	2π	0.47
3	$tg\ x$	-π/2	π/2	π/10
4	$\frac{1}{x-0.5}$	0	1	0.1
5	$2^{-\sin(x)}$	0	2π	π/10
6	$\frac{\sqrt{x+2}}{\sqrt{x-2}}$	0	10	1
7	$\frac{1}{x} + \sqrt[3]{x}$	0	15	1
8	$\frac{\cos(2x)}{\sin x}$	1.1	6.35	0.35
9	$\frac{\sin x}{1-x}$	π/2	3π/2	π/10

№ варианта	Функция Y(x)	Xн	Xк	h
10	$\frac{x}{\sqrt[3]{1-x^2}}$	0	2	0.2
11	$\frac{1+e^x}{1-e^x}$	0	5	0.5
12	$\frac{1-e^x}{1+e^x}$	0	5	0.25
13	$tg(\pi/2-(x-2))$	0	π	π/15
14	$\sqrt[4]{2x-3}$	1.5	15	1.35
15	$\frac{1+\sin x}{1-\cos x}$	0	2π	π/10

Методические указания.

Практически любой алгоритм содержит ряд операторов, которые нужно выполнить несколько раз подряд. Такая операция называется *циклом*. *(**Циклом** называется участок программы, который выполняется многократно при различных значениях аргументов.)* Операторы, которые выполняются циклически (повторяются), называются ***телом цикла.*** Цикл может иметь одну или несколько точек входа и обязательно один или несколько выходов. Если цикл не имеет выхода, то алгоритм составлен неправильно.

Для всех операторов цикла характерна следующая особенность: повторяющиеся вычисления записываются всего лишь один раз. Они и называются телом цикла. Вход в цикл возможен только через его начало. Переменные оператора цикла должны быть определены до входа в циклическую часть (т.е. переменным должны быть присвоены какие-либо начальные значения). Не забывайте про условие продолжения цикла. С каждым повторением операторов циклической части (тела цикла) переменная цикла должна увеличиваться (уменьшаться) на заданную величину (шаг). Если не задать приращения переменной цикла или не предусмотреть выход из цикла, то циклические вычисления будут повторяться бесконечно, произойдёт «зацикливание» программы.

Циклы широко применяются для решения самых разнообразных задач:

- табулирование функции (нахождение значения функции для аргумента, изменяющегося от начального до конечного значения с заданным шагом);
- нахождение суммы ряда;
- вычисление суммы n слагаемых;
- вычисление произведения n сомножителей (вычисление факториала);
- приближённое вычисление определённого интеграла (площади фигуры) и т.д.

В **Delphi** имеются три оператора цикла:

- "простой" оператор цикла
- условный оператор цикла
- условный оператор повторения

Простой оператор цикла применяется, когда известно количество повторений цикла. Он записывается так:

for *счётчик* **:=** *выражение-1* **to** *выражение-2*
do *действие* **;**

Счётчик - это переменная, которая должна быть объявлена перед логическим блоком, в котором оператор цикла расположен, и её тип должен относиться к одному из перечислимых

типов, обычно **Integer**.
Выражение-1 и *выражение-2* могут быть как константой или идентификатором, так и вызовом функции.
Действие - один или несколько операторов **Delphi**. Если это группа операторов, то они должны быть заключены в логические скобки **begin/end**.

В начале работы оператора переменная-счётчик получает значение *выражения-1*. Если при этом значение *счётчика* окажется **меньше или равно** значению *выражения-2*, то выполняются операторы, входящие в *действие*. Это и есть один цикл. Затем переменная-счётчик принимает значение, следующее за текущим, и начинается новый цикл, то есть сравнение *счётчика* и *выражения-2*, выполнение *действия*, и так далее, до тех пор, пока значение переменной-счётчика не превысит значение *выражения-2*.

Возможна работа оператора цикла, при котором переменная-счётчик будет не увеличиваться, а уменьшаться. В этом случае ключевое слово **to** заменяется на **downto**:

for *счётчик* **:=** *выражение-1* **downto** *выражение-2* **do** *действие* **;**

Соответственно, *выражение-1* должно быть **больше или равно** *выражению-2*.

В приведенном ниже примере оператор **For** используется для создания строки, содержащей 10 наборов по 10 цифр (от 0 до 9); каждый набор отделяется от следующего одним пробелом. Внешний цикл использует переменную-счётчик, которая уменьшается на единицу при каждом выполнении цикла.

```
Var
 Words, Chars: byte;
 MyString: string;
Begin
For Words := 10 downto 1 do                    ' Цикл выполняется 10 раз.
 begin
For Chars := 0 To 9 do                         ' Цикл выполняется 10 раз
        MyString := MyString + IntToStr(Chars);  ' Добавляет цифру в конец строки.
 MyString := MyString + ' '   ;                ' Добавляет пробел.
 end;
 ShowMessage (MyString);

End;
```

Условный оператор цикла удобно использовать в том случае, когда количество повторений заранее не известно:

while *условие* **do**
тело цикла **;**

Этот цикл будет выполняться до тех пор, пока истинно ***условие*** (логическое выражение, возвращающее значение типа **Boolean**). При этом если это выражение сразу равно **false**, *тело цикла* не будет выполнено ни разу.

Нужно очень внимательно следить за написанием *условия* и контролем завершения цикла, так как в результате ошибки цикл **while** будет повторяться бесконечное количество раз, что приведёт к "зацикливанию" и "зависанию" программы.

Условный оператор повторения сначала выполняет *тело цикла*, а затем уже проверяет выполнение *условия*:

Repeat
тело цикла
until *условие* **;**

Таким образом, этот вариант цикла гарантирует, что *тело цикла* будет выполнен по крайней мере один раз. И будет выполняться до тех пор, пока *условие* не станет истинным (т.е. **true**). Стоит отметить, что это единственный оператор **Delphi**, в котором *тело цикла* не требуется заключать в логические скобки **begin/end**. Начало и конец *тела цикла* определяются по ключевым словам **repeat** и **until**.

Вместе с операторами цикла используются специальные команды:

- команда прерывания цикла,
- команда продолжения цикла.

Команда прерывания цикла применяется, если в процессе выполнения операторов тела цикла выясняется необходимость его завершения. Вот эта команда:

Break ;

При её выполнении управление передаётся на первый оператор, следующий за оператором цикла.

Команда продолжения цикла позволяет немедленно продолжить выполнение цикла, пропустив все оставшиеся операторы в теле цикла, то есть начать следующую ***итерацию***.
Вот эта команда:

Continue ;

Справедливости ради стоит рассказать об ещё одном операторе, позволяющем изменить последовательность выполнения программы. Это оператор **перехода**:

goto *метка* **;**

В качестве *метки* может использоваться любой допустимый идентификатор или число в диапазоне от 0 до 9999. *Метку* предварительно необходимо объявить в разделе описания переменных, но с помощью не ключевого слова **var**, а ключевого слова **label**:

label *метка* **;**

или

label *список меток* **;**

Переходить можно как вниз, так и вверх по программе. Двоеточие отделяет метку от оператора, на который производится переход. Пример использования оператора перехода:

```
var X, Y: Integer;
label A, B;

begin
 A: X:=5 ;
...
операторы программы
 goto B;
...
 B: Y:=25;
 goto A;
end;
```

Из этого примера видно, что оператор **end ;** завершающий программу, никогда не будет выполнен, то есть программа зациклится. Именно поэтому, вообще, использование оператора перехода является **плохим стилем программирования**, и без его использования вполне можно обойтись использованием условных операторов и операторов цикла. Единственный случай, когда использование оператора **goto** может быть оправдано - это выход из нескольких вложенных циклов, что иначе требует применения нескольких операторов **Break**.

Задача (пример №1).

Найти сумму первых 15-и натуральных чисел.

Задача сводится к организации цикла по i. Для циклического накапливания сумм при составлении соответствующих алгоритмов используется предписание стандартного вида:

Сумма = сумма + слагаемое

Перед началом цикла сумма должна иметь нулевое значение. Понимать эту формулу следует так:

Пусть значение переменной «сумма» хранится в ячейке памяти № 1, а значение переменной «слагаемое» – в ячейке памяти № 2. Все выражения выполняются по правилам приоритета арифметических операций справа налево, т.е. из ячейки памяти № 2 извлекается значение переменной «слагаемое», из ячейки № 1 извлекается значение переменной «сумма», оба числа складываются (за такие операции отвечает процессор, а именно, арифметико – логическое устройство), а результат помещается в ячейку № 1 вместо старого значения переменной «сумма». Таким образом, в ячейке № 1 происходит накопление суммы. Если в

качестве слагаемого используется переменная цикла, то с каждой итерацией (шагом) цикла значение этой переменной будет меняться.

Словесная запись этого алгоритма (цикл «До», с постусловием):

1. i = 1, S = 0
2. S = S + i
3. i = i + 1
4. если i ≤ 15, перейти к шагу 2
5. вывести на экран значение S.
6. конец

Блок-схема алгоритма, соответствующая этой записи, изображена на рис.1.Согласно ГОСТ 19.701-90 схему этого алгоритма можно изобразить так, как на рис. 2.

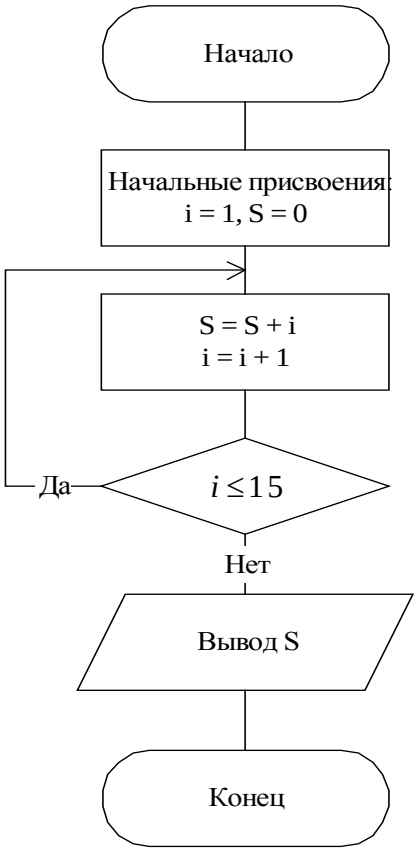

Рис. 1

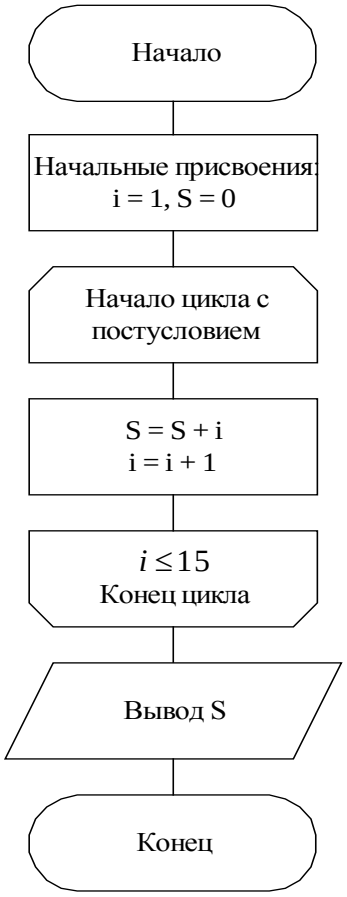

Рис. 2

Текст программы:

```
Procedure example();
Var
i, S: integer;
begin
i := 1;
S := 0;
Repeat
  S := S + I;
  i := i + 1;
until  i  > 15;
ShowMessage (‘ S = ‘ + IntToStr(S));
End.
```

Результат выполнения программы: S = 120.

Для решения этой задачи можно использовать и цикл с предусловием. Словесная запись этого алгоритма (цикл «Пока»):

1. i = 1, S = 0
2. если i > 15, перейти к шагу 6
3. S = S + i
4. i = i + 1
5. вернуться к шагу 2
6. вывести на экран значение S.
7. конец

Блок-схема алгоритма, соответствующая этой записи, изображена на рис.3. Согласно ГОСТ 19.701-90 схему этого алгоритма можно изобразить так, как на рис. 4.

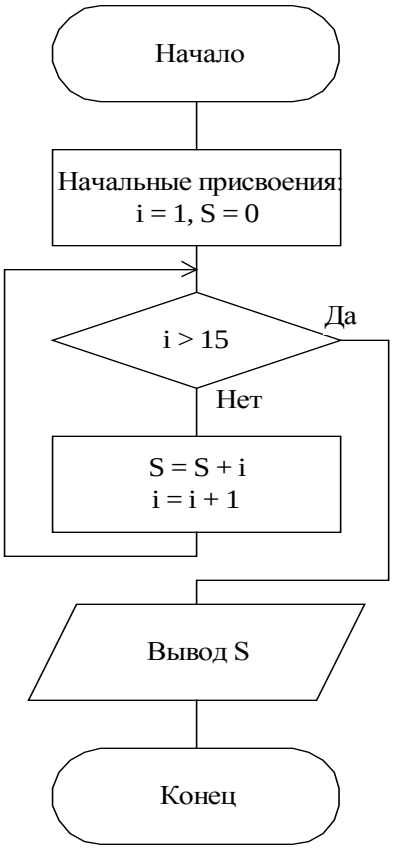

Рис.3

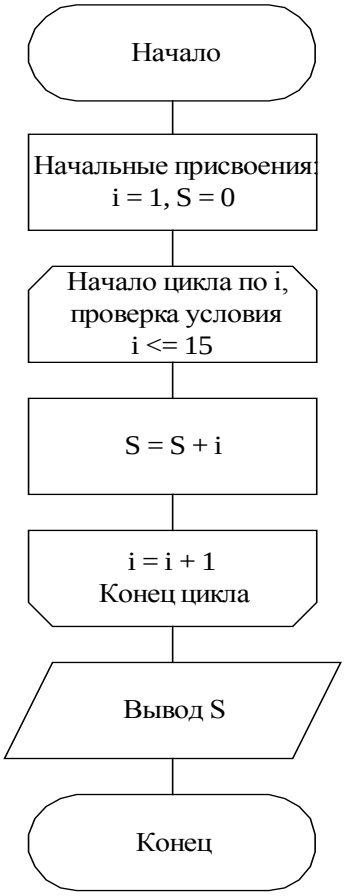

Рис.4

Текст программы: *(будет позже)*

При использовании цикла с параметром блок-схема алгоритма изображена на рис. 5.

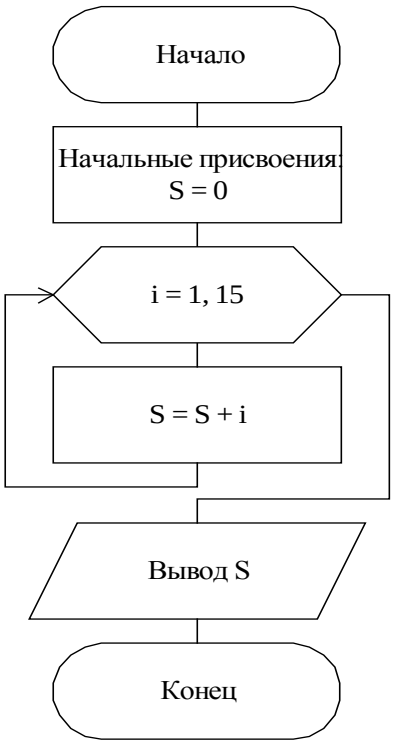

Рис. 5

Текст программы: *(будет позже)*

Ещё раз кратко об операторах циклов:

Циклы можно использовать для повторного выполнения блока кода в случае соблюдения определенного условия (или до тех пор, пока условие не соблюдается. В Delphi существуют следующие конструкции циклов: **for**, **while**..., **repeat**...**until**. Ниже рассмотрим их более подробно.

Конструкция цикла **for** удобна, когда заранее точно известно, сколько раз будет выполняться цикл. Рассмотрим пример:

```
procedure TForm1.Button1Click(Sender: TObject);
var
    i,p: Integer; {объявляем переменные типа Integer}
begin
 p:=0;
   for i:=0 to 100 do     {организовываем цикл повторений равный 100}
      begin
        p:=p+1;  {при каждом выполнении переменная p будет увеличиваться на 1}
      end;
   ShowMessage(inttostr(p));  {результат равен 101}
end;
```

Цикл также может организовываться от большего к меньшему: for i:=100 **downto** 0 **do**
Конструкция цикла **while** повторяет оператор или группу операторов, пока определенное условие не нарушится. Цикл **while** должен использоваться, когда неизвестно, сколько раз должен использоваться цикл.

```
procedure TForm1.Button1Click(Sender: TObject);
var
    i: Integer; {объявляем переменную типа Integer}
begin
 i:=0;
{организовываем цикл с условием выполнения пока i не будет равно 100}
```

```
   while i <> 100 dobegin
        i:=i+1; {при каждом выполнении переменная i будет увеличиваться на 1}
      end;
   ShowMessage(inttostr(i));  {результат равен 100}
end;
```

Цикл **repeat**...**until** используется для организации циклического выполнения совокупности операторов, до тех пор, пока не выполнится некоторое условие.

```
procedure TForm1.Button1Click(Sender: TObject);
var
    i: Integer; {объявляем переменную типа Integer}
begin
 i:=0;
{организовываем цикл с условием выполнения пока i не будет равно 100}
repeat
begin
{при каждом выполнении переменная i будет увеличиваться на 1}
        i:=i+1;end;
   until i=100;
   ShowMessage(inttostr(i));  {результат равен 100}
end;
```