# 7. Структура управления с повтором.

Во время обсуждения принципов построения алгоритмов были перечислены основные структуры управления — присвоение, последовательность, условный выбор и повтор. В предыдущих главах уже были даны описания всех структур, кроме последней. В данной главе рассмотрим структуру с повтором. Уже в алгоритмах было использовано три вида структур с повтором -- *Для (For), Пока (While)* и *Повторять (Repeat).*

Алгоритмически эти структуры можно представить в следующем виде:

*Для параметр – цикла* =n1..n2 *Делать:* { Тело цикла }

Тело цикла выполняется, пока *параметр —цикла* пробегает все значения от n1 до n2 (n1≤n2).

*Пока* (логическое – условие) *Делать:* { Тело цикла }

Тело цикла выполняется повторно до тех пор, пока значение *логического -- условия* остается истинным.

*Повторять*
{ Тело цикла }
*Пока – не* (*логическое —условие*)

Цикл повторяется до тех пор, пока значение *логического – условия* остается ложным.

Количество повторов зафиксировано только в первом варианте цикла. В двух других вариантах условие прерывания цикла нужно контролировать вручную. Это означает, что где-то в теле цикла

переменная, величина, которая проверяется в условии выполнения/прерывания цикла, должна изменять свое значение с тем, чтобы гарантировать завершение цикла после конечного числа шагов и избежать *зацикления.*

Все три структуры можно непосредственно реализовать в Турбо-Паскале. Рассмотрим каждую из них по-отдельности.

## *7.1. Структура повтора и оператор цикла «For».*

В программировании часто может понадобиться повторно выполнять оператор или группу операторов фиксированное количество раз или же осуществить повторное выполнение, пока определенная контрольная переменная пробегает значения из упорядоченного множества. Эта самая простая форма структуры повтора и алгоритмически может быть записана в следующем виде:

*Делать* n *раз:* { Тело цикла }

Или

*Для параметр— цикла* =n1..n2 *Делать:* { Тело цикла }

Цикл повторяется (n2-n1+1) раз (n1≤n2), пока *параметр—цикла* пробегает все значения от n1 до n2. Обычно, если условие n1≤n2 не соблюдается, цикл не выполнится вообще.

Не все проблемы, связанные с повторным выполнением каких-либо инструкций, можно реализовать с помощью данной структуры, но в некоторых случаях она достаточно удобна. Для ее реализации в Турбо-Паскале определен оператор «For». Например, пусть требуется вывести на экран последовательность целых чисел от нуля до десяти, Алгоритмически это запишется в виде:

*Для* i = { от 1 до 10 } *Делать:*
{ Вывести значение i}

Программный фрагмент будет иметь вид:

For i:=l to l0 Do Write(i,' ');

Здесь *i* является параметром цикла, который инициирует повторы. После «Do» стоит один оператор, который составляет тело цикла. Тело цикла может быть либо простым, либо составным оператором, и, следовательно, в последнем случае группа операторов должна быть заключена в программные скобки «Begin End».

Общий формат заголовка оператора имеет вид:

For *параметр —цикла:* = nl to n2 Do оператор; { n1<=n2 }

где nl и n2 — соответственно, начальное и конечное значения параметра цикла.

Рассмотрим пошаговое выполнение цикла для приведенного примера:

Первый шаг: *i*=l, следовательно, за первый раз
выполнится оператор: Write(l,' ');
Второй шаг: i=2, выполняемый оператор: Write(2,' ');
... и т.д.

Десятый, последний шаг: *i=10,* Write(10,' ');

Результат вывода:

1 2 3 4 5 6 7 8 9 10

После выполнения последнего шага цикл завершится и управление передается следующему оператору по тексту программы.

В Турбо-Паскале определена другая версия оператора «For», которая позволяет индексацию параметра цикла в убывающем порядке. Формат заголовка имеет вид:

For *параметр – цикла:* = n2 DownTo nl Do оператор; {n2>=nl}

Рассмотренный пример в этой версии:

```
For i:=10 DownTo 1 Do Write(i,' ');
```

выдаст результат:

10 9 8 7 6 5 4 3 2 1

Заметим, что в качестве параметра цикла запрещается использовать действительную переменную, а, кроме того, менять его значение в теле цикла.

### 7.1.1. Примеры использования оператора «For».

Рассмотрим несколько примеров использования оператора цикла «For». Некоторые из них будут реализованы с помощью других операторов цикла в следующих разделах.

*Программа- пример: Вычисление факториала.*

Пусть надо найти факториал числа *n*, которое считывается с клавиатуры. Факториал вычисляется по формуле:

$$f(n) = \prod_{i=1}^{n} 1 \cdot 2 \ldots \cdot i \ldots \cdot n.$$

Здесь внутри тела используется значение параметра цикла. Начальное значение для факториала инициируется единицей, как это и подразумевается по умолчанию при ручном вычислении:

```
Program fact;
Var n,i,f:Integer;
Begin
    Write('Enter number:')
    Read(n)
    f:=1;
    For i:=1 to n Do f:=f*i;
    WriteIn ('Factorial of, ' n,' =, 'f,'.')
```

End.

Можно привести таблицу значений переменных в процессе пошагового выполнения цикла для *n=5:*

| Шаг | *i* | *f* |
|---|---|---|
| 0 | - | 1 |
| 1 | 1 | 1 |
| 2 | 2 | 2 |
| 3 | 3 | 6 |
| 4 | 4 | 24 |
| 5 | 5 | 90 |

*Программа-пример: Таблица соответствия для температуры.*

Пусть надо вывести таблицу соответствия для температуры в Цельсиях и в Фаренгейтах. Преобразование осуществляется по формуле:

$$F = \frac{9}{5} C + 32.$$

При оформлении вывода надо проследить, чтобы значения разместились в табличной форме. В примере дается очень простая реализация:

```
Program temperature_conversion;
Var celsius,i: Integer;
      fahrenheit: Real;
Begin
        Writeln('Celsius':15,'Fahrenheit':15);
        For i:==1 to 30 Do Write('*');
Writeln;
For celsius:=1 to 30 Do
Begin
        fahrenheit:= (9/5)*celsius + 32;
       Writeln(celsius:8, fahrenheit:20:2)
```

```
End
For i:= 1 to 30 Do Write('*')
Writeln;
End.
```

*Программа-пример: Нахождение среднего значения.*

С клавиатуры вводятся *n* целых чисел. Надо найти среднее значение. При этом количество вводимых чисел также определяется пользователем. Начальное значение для переменной суммы, в отличие от случая факториала, инициируется нулевым значением. Программа будет иметь вид:

```
Program average;
    Var n,i, sum integer;
    average: Real;
Begin
    Write('Enter number of numbers:')
    Read(n)
    sum:=0;
    For i:=1 to n Do sum:=sum+i;
    If n<> Then average :=sum/n;
   Writeln('Mean value = ', average:8:2)
End.
```

Здесь если на первый запрос пользователь введет 0, то нижний предел диапазона для значений параметра цикла окажется больше верхнего предела, и, следовательно, цикл не выполнится.

## 7.2. Структура повтора — Пока (While).

Так как данная структура повтора является самой важной, рассмотрим ее алгоритмическое представление более детально. Следующий пример иллюстрирует несколько важных приемов. Программе задается множество положительных величин, требуется вывести минимальную, максимальную и среднюю величину.

Сначала нужно определить условие прерывания ввода чисел. Было бы утомительно запрашивать пользователя после каждого ввода: «продолжать или нет?» Иногда неудобно также определить

количество чисел наперед. Пусть все числа положительны, и за признак того, что данные больше не поступят, принять ввод отрицательного числа.

Надо также учесть случай, если данные не поступили вовсе, т.е. если пользователь за первый же раз вводит отрицательное число. Кроме того, программе нужно знать, какого типа числа вводятся, целые или действительные. Таким образом, детальная формулировка задачи будет иметь вид:

Пользователь должен ввести положительные действительные числа с клавиатуры на запрос программы. Признак конца — ввод отрицательного числа. Когда ввод чисел завершится, программа должна вывести минимальное и максимальное положительные числа, а также среднее значение введенных чисел. Если данные не поступили (т.е. пользователь первым числом вводит отрицательное число), тогда программа должна выдать сообщение об этом.

В программе, при вводе каждого числа, его нужно сравнивать с нулем, чтобы определить, выполняется ли условие прерывания ввода. Каждое положительное число должно добавляться к аккумулируемой сумме и счетчик количества вводимых чисел должен увеличиться на единицу [чтобы в конце вычислить среднее значение). Кроме того, каждое положительное число надо сравнивать с минимумом/максимумом, определенным на этом этапе, и в случае надобности, обновить их. Это означает, что пока вводимое число положительно, его нужно обрабатывать. Таким образом, в первом приближении алгоритм запишется так:

Инициализация;
Ввести первое *число;*
*Пока* (*число* положительно) *Делать*
{
Обработать *число;*
Ввести *число;*
}
*Если* (данных пет )
*Тогда* вывести « Данные не поступили.»

*Иначе*
{ Вычислить среднее значение;
Вывести результаты.
}

Как обычно, первый шаг и в этом алгоритме — инициализация. Его можно расширить позже, после реализации основных деталей.

В инструкции «Обработать число» реализуется несколько задач — аккумулируется сумма вводимых чисел и увеличивается счетчик количества вводимых чисел. Считанное число надо сравнить с текущим минимумом/максимумом и, в случае надобности, обновить. Таким образом, расширение инструкции «Обработать число» принимает вид:

Обработать *число.*
Добавить *число* к аккумулируемой *сумме.*
Добавить единицу к *счетчику* количества чисел;
*Если число* больше *текущего максимума*
*Тогда текущему максимуму* присвоить *число.*
*Если число* больше *текущего минимума*
*Тогда текущему минимуму* присвоить *число*

Очевидно, здесь предварительно должны быть определены следующие величины:

- Переменной, представляющей аккумулирующую *сумму,* присвоить ноль;
- Переменной, представляющей *количество* чисел, присвоить также ноль;
- Переменным, представляющим текущие *минимум* и *максимум,* присвоить значение первого из введенных чисел.

Следовательно, блок инициализации можно расширить следующим образом:

Инициализация:
Присвоить *сумме* ноль;
Присвоить *количеству* ноль;
Ввести первое *число;*
Присвоить текущему *минимуму* и *максимуму число.*

Если данные не поступили, это должно означать, что значение *количества* после цикла будет нулевым. Для вычисления средней величины сумму надо разделить на *количество*. Таким образом, в конечной форме алгоритм будет иметь вид:

Инициализация:
Присвоить *сумме* ноль;
Присвоить *количеству* ноль;
Ввести первое *число*;
*Присвоить* текущему *минимуму* и *максимуму* это *число*;
*Пока* (число положительно) *Делать*
{
Обработать *число:*
Добавить *число* к аккумулируемой *сумме;*
Добавить единицу к *счетчику* количества чисел;
*Если число* больше *текущего максимума*
*Тогда* текущему *максимуму* присвоить *число.*
*Если число* больше *текущего минимума*
Тогда текущему *минимуму* присвоить *число;*
Ввести *число;*
}
*Если* (данных нет!)
*Тогда* вывести « Данные не поступили!»
*Иначе*
{
Присвоить *среднему* значению *сумма/количество;*
Вывести: *количество, среднее* значение,
*минимум* и *максимум.*
}

Заметим, что если данные не поступают, текущему *минимуму* и *максимуму* присвоится отрицательное значение, но это никак не отразится на фактическом результате программы, так как в этом случае эти переменные не будут использоваться.

### 7.2.1. Оператор цикла «While».

Оператор повтора «While» состоит из заголовка и тела цикла, и синтаксически записывается как:

While (*Логическое— условие)* Do Оператор;

До тех пор, пока *Логическое—условие* остается истинным, будет выполняться «Оператор», который является либо простым, либо составным оператором, и следовательно, в последнем случае, группа выполняемых в цикле операторов должна быть заключена в программные скобки «Begin End». Фрагмент вывода натуральных чисел от 0 до 10 в одну строку с помощью этого оператора запишется в виде:

```
i:=l;
While (i<=10) Do
Begin
    Write(i, ' ');
    i:=i+l;
End;
```

Здесь переменная *i* имеет смысл параметра цикла. Оператор «While» не гарантирует контроля над условием выполнения цикла, как это организовано в операторе «For», ответственность за это управление полностью ложится на программиста. *Логическое—условие* проверяет значение параметра *i*, следовательно, именно от этой величины зависит, будет ли выполняться тело цикла, и на каждом этапе пользователь должен сам следить за ней: правильно определить ее исходное значение перед началом цикла (оператор *i*:=l;) и обеспечить её изменение в теле цикла (оператор *i*:=*i*+1;), с тем, чтобы избежать зацикления.

Фрагмент вывода чисел в обратном порядке будет иметь вид:

```
i:=10;
While (i>=l) Do
Begin
    Write(i,' ');
```

```
    i:=i-1;
End;
```

Рассмотрим еще один пример. Следующий фрагмент считывает числа с клавиатуры и суммирует их до тех пор, пока пользователь введет ноль или отрицательное число:

```
sum := 0;
Read(x);
While (x>0) Do
Begin
    sum := sum + x;
    Read(x);
End;
```

Отметим несколько моментов:

- ✓ Надо иметь возможность проверить *Логическое — условие* перед первым запуском цикла, т.е. все участвующие в нем переменные в этот момент должны быть определены. В первом примере это переменная *i*, которая инициируется единицей, а во втором — переменная *x*, которую задает пользователь с клавиатуры.
- ✓ Хотя бы одна переменная, тестируемая в условии, должна изменять свое значение в теле цикла. Иначе мы потеряем возможность контролировать условие выполнения цикла, и если оно было истинным с самого начала, оно останется истинным и в процессе выполнения цикла, следовательно, цикл невозможно будет прервать и он будет выполняться бесконечно. В первом примере переменная *i* увеличивает свое значение на единицу, гарантируя завершение цикла за конечное количество шагов, а во втором – пользователь сам определяет каждый раз новое значение переменной *x* и, следовательно, может завершить цикл на любом шаге.
- ✓ *Логическое – условие* проверяется до запуска цикла, следовательно, если оно изначально ложно, оператор не выполнится вообще.

### 7.2.2. Примеры использования оператора «While».

*Программа-пример: Вычисление факториала.*

```
Program repeatl;
Var n,i,f:Integer;
Begin
    Write('Enter number:')
    Read(n)
    i:=0;
    f:=l;
    While i<=n Do
    Begin
            f:=f*i;
            i:=i+l
    End;
    WriteIn ('Factorial of, 'n,' =', f,'.')
End.
```

*Программа-пример: Таблица соответствия для температуры.*

```
Program repeat2;
Var celsius,i:Integer;
    fahrenheit:Real;
Begin
    Writeln('Cetsius': 15, 'Fahrenheit': 15);
    For i:=l to 30 Do Write('*');
    Writeln;
    celsius:=0;
    While celsius<=30 Do
    Begin
            fahrenheit:= (9/5)*celsius + 32;
            Writeln(celsius:8, fahrenheit:20:2);
            celsius:= celsius+1
      End;
      For i:=l to 30 Do Write('*');
      Writeln;
End.
```

*Программа-пример: Нахождение среднего значения.*

```
Program repeat3;
Var n,i,sum:Integer;
    average:Real;
Begin
    Write('Enter number of numbers:')
    Read(n)
    sum:=0;
    i:=0;
    While i<=n Do
    Begin
            sum:=sum+i;
            i:=i+1
    End;
    average:=sum+i;
    Writeln('Mean value=', average:8:2)
End.
```

*Программа-пример: Таблица значений функции Sin.*

Следующая программа выводит на экран таблицу значений $x$ и sin(x), где $x$ принимает значения от 0° до 180° с шагом 10°. Отметим, что аргумент функции «sin» измеряется в радианах и, следовательно, $x$ предварительно нужно преобразовать из градусов в радианы, а затем вычислить значение функции.

```
Program sinTable;
{ Tabulates x and sin(x) }
Const koeff=Pi/180; { Коэффициент преобразования }
Var degree:Integer;
radian:Real;
Begin
    {Вывод заголовка}
Writeln('Degrees': 10, 'sin(degrees)': 15);
For i:=1 to 30 Do Write('*'); Writeln;
degree :=0;
    { Начало цикла }
While (degree <= 180) Do
```

```
Begin
    { Преобразование градусов в радианы }
 radian := degree * koeff;
    { Вывод строки значений }
 Writeln(degree:7, sin(radian):10:3);
 degree := degree + 10;
 End; {While}
 For i:=l to 30 Do Write('*'); Writeln;
 End.
```

Отметим, что если в этой программе цикл вывода значений был бы организован с помощью оператора «For», то потребовался бы дополнительный шаг вычисления полного количества повторов «n», которое зависит от диапазона аргумента и от шага, т.е. n=(180-0) Div 10.

В таких случаях всегда удобнее использовать структуру повтора с условием.

## *7.3. Структура — Повторять (Repeat).*

Оператор «While» выполняет цикл до тех пор, пока некоторое условие остается истинным. Иногда, естественно, возникает потребность выполнять операторы до тех пор, пока некоторое условие не станет истинным.

Для рассмотренного алгоритмического примера с циклом «While» новая версия структуры повтора дает:

Инициализация;
Ввести перое число;
*Повторять (Repeat)*
{
Обработать *число*;
Ввести *число*;
}
*Пока –не* (*Until*) *число* отрицательное;
(Т.е. до тех пор, пока число не станет отрицательным)

Если при выполнении данной версии цикла (Repeat) задавать последовательность положительных чисел, замыкающуюся отрицательным числом, то результат будет таким же, как и в случае цикла «While», но его поведение будет другим, если первым ввести отрицательное число, поскольку условие прерывания цикла проверяется после выполнения тела. Таким образом, так как эта структура выполняет тело хотя бы один раз, первое число должно быть протестировано отдельно, и в зависимости от того, положительно оно или нет, надо выполнить те или иные действия. Следовательно, использование «Repeat» может оказаться не таким удобным для случаев, когда наперед неизвестно, будет ли вообще выполняться тело цикла. Тогда лучше использовать цикл «While», который проверяет условие до начала выполнения тела цикла.

### 7.3.1. Оператор цикла «Repeat».

Структура повтора «Repeat» организована таким образом, что группа операторов, заключенная между ключевыми словами «Repeat Until», выполняется до тех пор, пока логическое условие прерывания цикла не станет истинным. Общий формат оператора имеет вид:

```
Repeat
    оператор 1;
    оператор 2;
    …
    оператор n
    Until логическое —условие,
```

Перед ключевым словом «Until» точку с запятой можно опустить.

Таким образом, тело цикла выполняется по крайней мере один раз. Следовательно, нужно проследить, чтобы начальные значения параметров, составляющих логическое условие прерывания цикла, не вызывали конфликт или логическую ошибку в процессе его выполнения.

В данной структуре повтора, как и в структуре «While», параметр цикла явно не определен. Следовательно, здесь так же нужно ввести величину, значение которой будет проверяться с

помощью логического условия, и проследить за изменением ее значения в теле цикла.

«Repeat» — единственная структура повтора, в теле которого количество выполняемых операторов не ограничено, и следовательно, отпадает потребность в программных скобках «Begin End»:

```
i:=0;     { Инициализация параметра цикла }
Repeat
    i:=i+l; { Изменение значения параметра цикла }
    Write(i,' ')
Until i<=10;
```

## 7.3.2. Примеры использования оператора «Repeat».

*Программа-пример: Вычисление факториала.*

```
Program repeatl;
Var n,i,f:Integer;
Begin
    Write('Enter number: ')
    Read(n)
    i:=0;
    f:=l;
    Repeat
            i:=i+l;
            f:=f*i
    Until i<=n;
    WriteIn('Factorial of', n,' =', f,'.')
End.
```

*Программа-пример: Таблица соответствия для температуры.*

```
Program repeat2;
Var celsius,i:Integer;
    fahrenheit:Real;
Begin
```

```
    Writeln('Celsius': 15, 'Fahrenheit': 15);
    For i:=1 to 30 Do Write('*');
    Writeln;
    celsius:=0;
Repeat
    Inc(celsius);
    fahrenheit:= (9/5)*celsius + 32;
    Writeln(celsius:8,fahrenheit:20:2)
    Until celsius<=30;
    For i:=1 to 30 Do Write('*');
    Writeln;
End.
```

*Программа-пример: Нахождение среднего значения.*

```
Program repeat3;
Var n,i,sum:Integer;
    average:Real;
Begin
    Write('Enter number of numbers:')
    Read(n)
    sum:=0;
    i:=0;
    Repeat
            i:=i+1;
            sum:=sum+i
    Until i<=n;
    average:=sum/n;
    Writeln('Mean value =', average:8:2)
End.
```

Обратите внимание, что в зависимости от того, используется оператор «While» или «Repeat», порядок операторов в теле циклов меняется. Постарайтесь объяснить причину сначала теоретически, а затем протестируйте на компьютере.

## 7.4. Организация итерации на примере вычисления квадратного корня.

Итерация — один из важнейших методов решения задач. В этом подходе сначала генерируется первое приближение решения, а затем, используя какой-нибудь метод, на каждом шаге находится улучшенное, с точки зрения точности, решение. Процесс повторяется до тех пор, пока разность между двумя последовательными аппроксимациями не достигнет величины требуемой точности. Рассмотрим алгоритм процесса итерации на примере структуры *Пока:*

Найти первую аппроксимацию (в переменной *old);*
Найти улучшенную аппроксимацию *(new),*
как функцию от старой *(old),*
*Пока* (|*old-new*| > требуемой *точности) Делать*
{
*old: =new,*
Найти новую лучшую аппроксимацию *(new),*
как функцию от *old;*
}

Здесь величина *точности* определяет продолжительность выполнения цикла

Рассмотрим задачу нахождения квадратного корня с помощью итерации. Задается положительное число $x$. Если через old обозначить первую аппроксимацию квадратного корня от $x$, то новая аппроксимация вычисляется по формуле:

$$new = \frac{old + \frac{x}{old}}{2}$$

Рассмотрим этапы вычислений для случая $\sqrt{15}$, и за первое приближение возьмем число 4:

| *old* | *New* |
|---|---|
| 4 | (4+15/4)/2=3.875 |
| 3.875 | (3,875+15/3.875)/2=3. 872983871 |
| 3,873 | (3,873+15/3.873)/2=3.872983346 |

Можно проверить, что это число довольно точно соответствует величине $\sqrt{15}$. В итерационных методах сходимость не всегда достигается так легко, а иногда и вовсе невозможна, но в случае квадратного корня, если первое приближение положительно, итерация всегда сходится.

В условии цикла упоминается «требуемая точность», которая должна быть определена исходя из диапазона обрабатываемых данных. В случае вычисления квадратного корня это может быть величина 0.0001, а условие можно записать как

*Пока* ( | old-new | > 0.0001) *Делать...*

что означает точность до трех знаков после десятичной точки. Отметим, что проверяется именно абсолютное значение разности между двумя последовательными итерациями, так как ряд итерации может не быть монотонным, т.е. строго убывающим или возрастающим, и тогда на определенном шаге разность может оказаться отрицательной, что вызовет преждевременное прерывание цикла итерации.

Однако здесь может возникнуть сложность другого характера, связанная с относительной оценкой погрешности. Рассмотрим два приближения:

| Точное значение | Приближенное значение |
|---|---|
| 100 | 100.1 |
| 0.1 | 0.2 |

Абсолютное значение погрешности в обоих случаях 0.1, но в первом случае это составляет 0.1% от приближенного значения, а во втором — 100%. Таким образом, если в качестве меры точности

использовать просто абсолютную погрешность, в первом случае это даст малую долю ошибки, но во втором — ошибка будет того же порядка, что и приближенное решение. Следовательно, резонно в качестве меры погрешности использовать относительную меру точности, что в нашем случае дает условие:

*Пока* { (|old-new |/ new) > 0.0001} *Делать...*

Кроме этого, в алгоритме нужно учесть граничные случаи:

* Если входное число отрицательно, квадратный корень не определен и, следовательно, в этом случае итерационный процесс не должен начаться вообще;

* Если поступает число с нулевым значением, то здесь ответ также известен наперед, но если процесс итерации будет задействован, в вычислениях будет производиться деление на ноль, следовательно, этот случай также нужно учесть отдельно. В итоге, программа будет иметь вид:

```
Program square_Root;
Const accuracy=0.0001;
Var number,old,new:Real;

BEGIN
    Write('Enter a positive number: ');
    Read(number);
     If number < 0
     Then WriteLn('Cannot find square root of a negative number')
     Else
     Begin
          If number = 0
       Then WriteLn('Square root of 0 is 0!')
       Else
       While Abs((old-new)/new) > accuracy Do
       Begin
       old:=number; { первое приближение }
       new:=(old+number/old)/2; {улучшенное приближение}
     End; {While}
```

```
    WriteLn(' Square root of, 'number,' is ', new,'.')
  End; {Else}
END.
```

## 7.5. Вложенные циклы.

Оператор цикла может стоять в теле такого же или другого цикла. В этом случае получаем т.н. вложенные циклы, т.е. внутренний цикл выполняется при каждом выполнении внешнего цикла.

Здесь должны быть соблюдены следующие правила:

• Каждый цикл, как внешний, так и вложенный, должен иметь различный параметр цикла.

• Вложенный цикл должен полностью располагаться внутри внешнего цикла.

Рассмотрим пример. Пусть надо организовать следующий вывод:

```
12345
1234
123
12
1
```

С использованием оператора «For» программа будет иметь вид:

```
Const n=5;
Var i,j: Integer;
Begin
    For i:==l to n Do
    Begin
            For j:=l to n-i Do Write(j, ' ');
            Writeln;
    End;
End.
```

Тот же фрагмент с помощью циклов «While» и «Repeat» будет иметь вид:

While:

```
Const n=5;
Var i, j:Integer;
Begin
    i:=l;
      While i<=n Do
Begin
    j:=1;
    While j<=n-i Do
    Begin
Write(j, ' ');
Inc(j)
End;
            Writeln;
            Inc(i)
        End;
    End.
```

Repeat:

```
    Const n=5;
    Var i, j: Integer;
    Begin
            j:=l;
            Repeat
            Begin
                    J:=1;
                    Repeat
                        Write(j, ' ');
                        Inc(j)
                    Until j>n-i;
                    Writeln;
                    Inc(i)
            Until i>n;
    End.
```

Другие примеры со вложенными циклами будут приведены в разделе описания многомерных массивов.