Опыт не требуется

Книги этой серии наглядно свидетельствуют, что новичка можно научить языку и хорошему стилю программирования, не повергая его в тоску и уныние.
– Лу Гринзоу, обозреватель Dr. Dobb's Journal.

Исходные коды находятся на дискете

Сэм А. Аболрус

Программирование на PASCAL

СИМВОЛ

Learn Pascal in Three Days

Third Edition

Sam A. Abolrous

Wordware Publishing, Inc.

Опыт не требуется

Программирование на Pascal

Третье издание

Сэм А. Аболрус

Символ®

Санкт-Петербург — Москва
2003

Серия «Опыт не требуется»

Сэм А. Аболрус

Программирование на Pascal, 3-е издание

Перевод А. Петухова

Главный редактор	*А. Галунов*
Зав. редакцией	*Н. Макарова*
Научный редактор	*В. Озеров*
Редактор	*В. Овчинников*
Корректура	*С. Беляева*
Верстка	*Н. Гриценко*

Аболрус С.

Программирование на Pascal, 3-е издание. – Пер. с англ. – СПб: Символ-Плюс, 2003. – 328 с., ил.

ISBN 5-93286-057-X

Книга Сэма Аболруса «Программирование на Pascal» – третье, обновленное издание одного из лучших руководств по Паскалю, ставшего бестселлером. Она идеально подходит для приобретения базовых знаний о структурном программировании. Легкий стиль изложения и большое количество примеров и упражнений помогут начинающим программистам изучить язык за очень короткий срок. Начав с простейших программ обработки числовых данных и вывода на печать, вы научитесь писать реальные приложения, которые будут выполняться на любой платформе.

Прилагаемая дискета содержит исходный код примеров, ответы к упражнениям и файлы, необходимые для их выполнения.

ISBN 5-93286-057-X
ISBN 1-55622-805-8 (англ)

Издательство «Символ-Плюс». 199034, Санкт-Петербург, 16 линия, 7,
тел. (812) 324-5353, edit@symbol.ru. Лицензия ЛП N 000054 от 25.12.98.
Налоговая льгота – общероссийский классификатор продукции
ОК 005-93, том 2; 953000 – книги и брошюры.
Подписано в печать 15.08.2003. Формат 70x100$^{1}/_{16}$. Печать офсетная.
Объем 20,5 печ. л. Тираж 3000 экз. Заказ N
Отпечатано с диапозитивов в Академической типографии «Наука» РАН
199034, Санкт-Петербург, 9 линия, 12.

Оглавление

Об авторе

Сэм Аболрус (Sam Abolrous), программист с большим опытом дизайна и разработки программного обеспечения, получил степень бакалавра инженера-электрика в университете Александрии, Египет.

Аболрус публиковался в ведущих журналах по программированию, написал более 50 книг в компьютерной области от Кобола до C++, включая «Learn C in Three Days» и «Learn Pascal», опубликованные издательством Wordware Publishing. Он разработал множество программ в области исследования слуха в медицинском центре университета штата Луизиана (Louisiana State University Medical Center, LSUMC). В настоящее время работает программистом в корпорации Microsoft.

Благодарности

Я бы хотел поблагодарить мою дочь Салли Аболрус (Sally Abolrous) за помощь при редактировании этой книги.

Предисловие

На примерах, приведенных в этой книге, можно изучить Паскаль за очень короткий срок. Вы начнете с простейших программ обработки числовых данных и вывода на печать, а закончите созданием реальных приложений с использованием структурного программирования.

Паскаль был разработан в начале 70-х годов Николасом Виртом (Niklaus Wirth), швейцарским программистом, и назван в честь французского математика Блеза Паскаля (Blaise Pascal, 1623–1662). Стандарт языка появился в 1983 году и был утвержден Институтом инженеров по электротехнике и электронике (IEEE) и Американским национальным институтом стандартов (ANSI). С ростом использования микрокомпьютеров язык подвергся изменениям и дополнениям, из которых наиболее популярны UCSD Pascal (разработан Калифорнийским университетом в Сан Диего) и Turbo Pascal фирмы Borland International.

Цель книги – научить писать переносимые, платформо-независимые программы с использованием, главным образом, стандартов IEEE/ANSI. Мы обсудим также новые возможности языка, ссылаясь на источники. Здесь не рассматриваются подробно непереносимые части языка (такие как графика), но зато уделяется внимание мощным возможностям современных реализаций, полезным при обработке данных. Программы, приведенные в книге, компилировались с помощью Turbo Pascal, но вы вправе использовать компилятор по своему выбору. Лишь в редких случаях потребуются небольшие изменения, к которым есть соответствующие комментарии.

Книга содержит ряд специальных обозначений:

Примечания. Содержат дополнительную информацию по сложным темам.

Предупреждения. Предостерегают о возможной опасности совершения ошибки.

Советы. Содержат конкретные рекомендации, облегчающие понимание материала и правильное выполнение примеров.

4

Циклы

4–1. Построение циклов

В предыдущей главе вы научились строить повторяющиеся циклы с помощью следующих инструментов:

- Оператор перехода, такой как GOTO, многократно передающий управление в начало программы.
- Условие выхода из цикла при необходимости.

Условие может быть использовано для проверки начального значения и выхода из цикла при достижении определенного значения. Для того чтобы повторить цикл некоторое число раз, необходимо организовать счетчик. В этом случае условие используется для проверки счетчика при каждом выполнении цикла. Этот тип цикла называется *арифметическим циклом*. В следующей программе эти простые приемы применяются для пятикратного вывода на экран сообщения «Прошу прощения, повторите еще раз». Алгоритм, используемый в программе, следующий:

1. Счетчик получает нулевое начальное значение.
2. Приращение счетчика на 1.
3. Проверяется, меньше или равно 5 значение счетчика.
4. Вывод сообщения.
5. Переход на шаг 2.

```
{ ------------------------------ Пример 4-1 ------------------------------ }
PROGRAM GoToLoop(OUTPUT);
LABEL
  1000; { объявление метки}
VAR
  Kounter :INTEGER;
BEGIN
```

```
  Kounter := 0;
1000:
  Kounter := Kounter + 1;
  IF Kounter <= 5 THEN
    BEGIN
      WRITELN(' Прошу прощения, повторите еще раз..');
      GOTO 1000 { перезапуск }
    END;
  WRITELN;
  WRITELN('Для продолжения нажмите ENTER..');
  READLN
END.
```

В этой программе перед входом в цикл, начинающийся с метки «1000», счетчику присваивается нулевое начальное значение. Внутри цикла счетчик увеличивается, а затем проверяется, меньше ли он или равен 5. Если да, то выполняется оператор WRITELN, и цикл повторяется с помощью оператора GOTO. Если условие не выполняется (то есть значение счетчика больше 5), программа заканчивает работу. Вывод программы выглядит так:

```
Прошу прощения, повторите еще раз..
Прошу прощения, повторите еще раз..
Прошу прощения, повторите еще раз..
Прошу прощения, повторите еще раз..
Прошу прощения, повторите еще раз..
Для продолжения нажмите ENTER..
```

Паскаль предоставляет готовые к использованию управляющие структуры для построения циклов, поэтому можно избежать такого нескладного кода. В одной конструкции управляющей структуры содержится и оператор ветвления, и условие.

В этой главе мы рассмотрим следующие конструкции:

- Цикл FOR
- Цикл WHILE
- Цикл REPEAT

Каждый из трех циклов имеет свои возможности, подходящие для различных приложений.

4–2. Цикл FOR

Цикл FOR представляет собой арифметический цикл, повторяющий выполнение оператора или блока операторов определенное число раз. Взгляните на пример:

```
{ ----------------------------- Пример 4-2 ----------------------------- }
PROGRAM ForLoop(OUTPUT);
```

```
VAR
  Kounter :INTEGER;
BEGIN
  FOR Kounter := 1 TO 5 DO
    WRITELN('Прошу прощения, повторите еще раз..');
  WRITELN;
  WRITELN('Для продолжения нажмите ENTER..');
  READLN
END.
```

Эта программа приводит к тому же результату, что и предыдущая, но она проще и лучше организована. Цикл FOR выполняет ту же работу, что и предыдущая программа. Он присваивает *управляющей переменной* (Kounter) *начальное значение* = 1, затем выполняет оператор, увеличивает значение управляющей переменной на единицу и повторяет процесс, пока значение Kounter не достигнет *конечного значения* = 5.

Основная форма конструкции FOR следующая:

```
FOR управляющая переменная := выражение-1 TO выражение-2 DO
     оператор;
```

где управляющая переменная – это счетчик цикла, выражение-1 – начальное значение, а выражение-2 – конечное значение.

Управляющая переменная, выражение-1 и выражение-2 могут иметь любой тип, за исключением REAL.[1] Все они должны быть одного типа.

Помните, что конструкция FOR – это один оператор, заканчивающийся точкой с запятой. Если по ошибке вы добавите еще одну точку с запятой, как в следующем цикле:

```
FOR Kounter := 1 TO 1000 DO;
          WRITELN('Прошу прощения, повторите еще раз..');
```

не удивляйтесь, если цикл выполнится только один раз независимо от конечного значения счетчика. Точка с запятой после ключевого слова DO прекращает выполнение цикла в этом месте.

Внутри цикла нельзя изменять значение управляющей переменной.[2] Взгляните на оператор присваивания, расположенный внутри цикла:

```
FOR K := 1 TO 10 DO
  K := 2
...
```

---

[1] Современные компиляторы понимают десятки встроенных (предопределенных) типов. Между тем, не все типы, кроме REAL, могут быть использованы в конструкции FOR. Для получения дополнительных сведений обратитесь к справочной информации по используемому вами компилятору. – *Примеч. науч. ред.*

[2] Компилятор это допускает, однако это строго не рекомендуется, разве только вы точно знаете, что делаете. – *Примеч. науч. ред.*

Даже если компилятор примет такой оператор, он отменит действие цикла, поскольку счетчик цикла все время равен 2. То же самое правило действует и для начального и конечного значений управляющей переменной.

Как обычно, использование блоков BEGIN-END позволяет включить в цикл столько операторов, сколько хотите.

Пример: Степени двойки

Число 2 и его степени играют важную роль в компьютерной сфере. Некоторые числа, такие как 1024 байт (эквивалентно 1 Кбайт) и 65 536 байт (64 Кбайт), имеют широкое применение. В следующей программе для вывода на экран степеней двойки применяется цикл FOR, а степень вычисляется по алгоритму из примера 2-2. Программа выводит степень и результат возведения числа 2 в эту степень. Начальное и конечное значения счетчика задаются пользователем в процессе выполнения программы. Таким образом, вы можете определять интервал чисел, который хотели бы протестировать:

```
{ ------------------------------ Пример 4-3 ------------------------------ }
PROGRAM ForLoop(INPUT, OUTPUT);
VAR
  Base, Power, Start, Final :INTEGER;
BEGIN
  Base := 2;
  WRITE('Введите начальную степень:');
  READLN(Start);
  WRITE('Введите конечную степень:');
  READLN(Final);
  WRITELN;
  WRITELN('Число     Степень двойки');
  FOR Power := Start TO Final DO
    BEGIN
      WRITE(Power:3);
      WRITELN(EXP(LN(Base)*Power):20:0)
    END;
  WRITELN;
  WRITELN('Для продолжения нажмите ENTER..');
  READLN
END.
```

Результат выполнения примера с использованием значений степени от 1 до 20:

```
Введите начальную степень:1
Введите конечную степень:20
Число    Степень двойки
  1                 2
  2                 4
  3                 8
  4                16
```

```
 5              32
 6              64
 7             128
 8             256
 9             512
10            1024
11            2048
12            4096
13            8192
14           16384
15           32768
16           65536
17          131072
18          262144
19          524288
20         1048576
Для продолжения нажмите ENTER..
```

Задание 4-1

Напишите программу определения високосного года в пределах от 1900 до 2000 года. Выведите на экран год и результат проверки в виде:

```
Год 1990 невисокосный год.
Год 1991 невисокосный год.
Год 1992 високосный год.
Год 1993 невисокосный год.
Год 1994 невисокосный год.
Год 1995 невисокосный год.
Год 1996 високосный год.
Год 1997 невисокосный год.
Год 1998 невисокосный год.
Год 1999 невисокосный год.
Год 2000 високосный год.
```

Пример: Среднее значение

Следующая программа демонстрирует ввод данных с применением цикла. Она принимает с клавиатуры последовательность чисел и вычисляет сумму и среднее значение чисел. В начале выполнения программы вас просят ввести число элементов, которое представляет конечное значение счетчика. Внутри цикла сумма накапливается в переменной `Sum` с помощью оператора:

```
Sum := Sum + Number;
```

Во время выполнения цикла среднее значение определяется по сумме и количеству элементов оператором:

```
Average := Sum / N;
```

Вот код программы:

```
{ ------------------------------ Пример 4-4 ------------------------------ }
PROGRAM AverageProg1(INPUT,OUTPUT);
VAR
  Average, Sum, Number :REAL;
  N, Kounter           :INTEGER;
BEGIN
  Sum := 0;
  WRITE('Введите количество элементов:');
  READLN(N);
  FOR kounter := 1 TO N DO
    BEGIN
      WRITE('Введите элемент #',kounter,': ');
      READLN(Number);
      Sum := Sum + Number { Точка с запятой необязательна}
    END;
  Average := Sum / N;
  WRITELN;
  WRITELN('Сумма чисел = ', Sum:0:2);
  WRITELN('Среднее значение чисел = ', Average:0:2);
  WRITELN;
  WRITELN('Для продолжения нажмите ENTER..');
  READLN
END.
```

Результат выполнения программы:

```
Введите количество элементов: 5
Введите элемент #1: 1
Введите элемент #2: 2
Введите элемент #3: 3
Введите элемент #4: 4
Введите элемент #5: 5
Сумма чисел = 15.00
Среднее значение чисел = 3.00
Для продолжения нажмите ENTER..
```

Обратите внимание, как количество элементов было выведено на экран внутри цикла с использованием значений управляющей переменной `Kounter`.

4–3. Пошаговое выполнение вверх и вниз

В предыдущих примерах счетчик цикла FOR всегда увеличивался. Это означает, что конечное значение должно быть больше начального, в противном случае цикл никогда не будет выполнен.

Альтернативная форма цикла FOR позволяет уменьшать значение счетчика путем замены ключевого слова TO на ключевое слово DOWNTO:

```
FOR управляющая переменная := выражение-1
            DOWNTO выражение-2 DO
    оператор;
```

Теперь можно начинать считать с большего значения и идти вниз, пока не будет достигнуто конечное значение.

Пример: Факториал

Факториал положительного целого числа определяется так:

```
N! = N * (N-1) * (N-2) ... * 3 * 2 * 1
```

Таким образом, факториал 4 равен 4*3*2*1, а факториал 3 равен 3*2*1. Можно написать следующие соотношения для факториала:

```
4! = 4 * 3!
3! = 3 * 2!
2! = 2 * 1!
1! = 1
```

Вообще говоря, можно написать следующий оператор Паскаля для вычисления факториала с использованием счетчика:

```
Factorial := Factorial * Kounter;
```

Переменной Kounter можно давать или положительное приращение от 1 до N, или отрицательное – от N до 1. Следующая программа использует этот алгоритм в цикле с отрицательным приращением:

```
{ ------------------------------ Пример 4-5 ------------------------------- }
PROGRAM FactorialProg1(INPUT,OUTPUT);
VAR
  Factorial       :REAL;
  Kounter, Number :INTEGER;
BEGIN
  WRITE('Дайте мне число, и я скажу вам его факториал: ');
  READLN(Number);
  Factorial := 1;
  FOR kounter := Number DOWNTO 1 DO
    Factorial := Factorial * Kounter;
  WRITELN('Факториал', Number,' равен ', Factorial:0:0);
  WRITELN;
  WRITELN('Для продолжения нажмите ENTER..');
  READLN
END.
```

Обратите внимание, что начальное значение 1 должно быть присвоено переменной Factorial до начала итерации. Результат выполнения программы:

```
Дайте мне число, и я скажу вам его факториал: 8
Факториал 8 равен 40320
Для продолжения нажмите ENTER..
```

Хотя факториал всегда является целым числом, использование типа REAL (или LONGREAL в Турбо Паскале) для переменной Factorial предоставляет больше места для хранения быстро растущих результатов вычисления факториала. При использовании типа INTEGER уже после 7! программа начнет выдавать странные результаты.

Задание 4–2

Измените предыдущую программу так, чтобы она проверяла входные значения. При нулевом значении программа должна заканчивать работу, не начиная выполнение цикла. Можно использовать оператор GOTO или функцию Турбо Паскаля EXIT.

4–4. Вложенные циклы

Подобно любому другому оператору, оператор цикла FOR может быть использован внутри другого оператора. В соответствии с нуждами приложения можно вкладывать сколь угодно много операторов цикла друг в друга. Следующая программа выводит на экран массив чисел:

```
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
```

Массив состоит из трех строк и пяти столбцов. Количеством строк и столбцов можно управлять с помощью счетчиков двух вложенных циклов. Как видите, для каждого прохода счетчика внешнего цикла (`Row`) счетчик внутреннего цикла (`Column`) делает пять проходов. Значения, которые появляются на выходе, – это значения счетчика `Column`. Заметьте, что с помощью счетчика внешнего цикла после вывода всей строки на экран выводится пустая строка.

```
{------------------------------ Пример 4-6 ------------------------------}
PROGRAM NestedLoops(OUTPUT);
VAR
  Row, Column :INTEGER;
BEGIN
  FOR Row := 1 TO 3 DO { Начало внешнего цикла }
    BEGIN
      FOR Column := 1 to 5 DO { Начало внутреннего цикла }
        WRITE(Column, ' '); { Конец внутреннего цикла}
      WRITELN { Этот оператор принадлежит внешнему циклу }
    END { Конец внешнего цикла}
END.
```

Обратите внимание на два ключевых слова END в предыдущей программе, первое из которых без точки с запятой, поскольку это последний оператор главного блока (главное тело программы). Ключевое слово WRITELN, стоящее перед этим END, также не заканчивается точкой с запятой, так

как это последний оператор в блоке цикла. И хотя они не обязательны, вы можете поставить точки с запятой. Если вы добавите в конец другой оператор (например, задержки вывода на экран), то ситуация изменится.

Задание 4–3

Измените последнюю программу так, чтобы она рисовала 50 звездочек в пяти строчках, как показано ниже:

```
* * * * * * * * * *
* * * * * * * * * *
* * * * * * * * * *
* * * * * * * * * *
* * * * * * * * * *
```

4–5. Цикл WHILE

Конструкция цикла WHILE содержит необходимое условие завершения цикла, но, в отличие от цикла FOR, в ней отсутствует счетчик. Ее основной вид:

```
WHILE условие DO
    оператор;
```

Эта форма говорит: «Выполняйте следующий оператор, пока условие принимает значение TRUE».

После входа в цикл проверяется условие (логическое выражение). Если его значение равно TRUE, выполняется оператор, стоящий за ключевым словом DO. Цикл будет повторяться, а оператор будет повторно выполняться, пока условие не примет значение FALSE. Программа должна иметь соответствующую логику, чтобы в нужное время условие приняло значение FALSE. В этом цикле можно использовать счетчик, но вам нужно самим заботиться о его положительном или отрицательном приращении.

Следующая программа демонстрирует тот же самый алгоритм вычисления среднего значения множества чисел, введенных с клавиатуры, но с помощью цикла WHILE. Условие состоит в сравнении значения счетчика Kounter с максимальным количеством элементов N. При достижении максимума цикл завершается.

```
{ ------------------------------ Пример 4-7 ------------------------------ }
PROGRAM AverageProg2(INPUT,OUTPUT);
VAR
  Average, Sum, Number :REAL;
  Kounter, N           :INTEGER;
BEGIN
  Sum := 0;
  Kounter := 1;
```

```
  WRITE('Введите число элементов:');
  READLN(N);
  WHILE Kounter <= N DO
    BEGIN
      WRITE('Введите элемент #',Kounter,': ');
      READLN(Number);
      Sum := Sum + Number;
      Kounter := Kounter + 1
    END;
  Average := Sum / N;
  WRITELN;
  WRITELN('Сумма чисел = ', Sum:0:2);
  WRITELN('Среднее значение чисел = ', Average:0:2);
  WRITELN;
  WRITELN('Для продолжения нажмите ENTER..');
  READLN
END.
```

Обратите внимание, что счетчик получает начальное значение в начале программы, а приращение – внутри цикла. Для первого прохода используется начальное значение (`Kounter := 1`), потому что приращение выполняется в конце. Это один из способов, но в нескольких следующих программах используются другие комбинации. Не забудьте также, что если в цикле надо выполнить более одного оператора, их нужно поместить в блок BEGIN-END.

Результат выполнения программы:

```
Введите количество элементов: 3
Введите элемент #1: 1
Введите элемент #2: 2
Введите элемент #3: 3
Сумма чисел = 6.00
Среднее значение чисел = 2.00
Для продолжения нажмите ENTER..
```

Можно не вводить количество элементов заранее, а подсчитывать их в цикле. В этом случае необходим ключ для завершения цикла, например ввод отрицательного числа. Взгляните на модифицированную версию программы, где каждое вводимое число проверяется на равенство –1:

```
{ ----------------------------- Пример 4-8 ----------------------------- }
PROGRAM AverageProg3(INPUT,OUTPUT);
VAR
  Average, Sum, Number :REAL;
  Kounter              :INTEGER;
BEGIN
  Sum := 0;
  Average := 0;
  Number := 0;
  Kounter := 0;
```

```
  WHILE Number<>-1 DO
    BEGIN
      Kounter := Kounter + 1;
      Sum := Sum + Number;
      WRITE('Введите элемент #',Kounter,' (или -1 для выхода): ');
      READLN(Number)
    END;
  IF Kounter > 1 THEN
  Average := Sum / (Kounter - 1);
  WRITELN;
  WRITELN('Сумма чисел = ', Sum:0:2);
  WRITELN('Среднее значение чисел = ', Average:0:2);
  WRITELN;
  WRITELN('Для продолжения нажмите ENTER..');
  READLN
END.
```

Результат выполнения программы:

```
Введите элемент #1 (или -1 для выхода): 1
Введите элемент #2 (или -1 для выхода): 2
Введите элемент #3 (или -1 для выхода): 3
Введите элемент #4 (или -1 для выхода): -1
Сумма чисел = 6.00
Среднее значение чисел = 2.00
Для продолжения нажмите ENTER..
```

Обратите внимание на следующие моменты в этой программе:

- Оператор цикла стоит в конце блока цикла, так что входное значение может быть проверено перед любой обработкой.
- Среднее значение определяется путем деления суммы на уменьшенное на единицу значение счетчика. Это нужно для того, чтобы учесть лишний проход, когда `Number` равно –1.
- Среднее значение вычисляется, только если переменная `Kounter` не равна 1. Это сделано для того, чтобы избежать ошибки «деления на ноль» в случае выхода из программы до введения данных. В таком случае будет получен следующий ответ:

```
Введите элемент #1 (или -1 для выхода): -1
Сумма чисел = 0.00
Среднее значение чисел = 0.00
Для продолжения нажмите ENTER..
```

Задание 4–4

Составьте программу, содержащую конструкцию цикла WHILE, которая выводит на экран таблицу умножения в виде:

1 * X = Y
2 * X = Y
3 * X = Y
4 * X = Y
5 * X = Y
6 * X = Y
7 * X = Y
8 * X = Y
9 * X = Y
...

Значение X принимается с клавиатуры, а значение Y представляет собой результат умножения.

4–6. Цикл REPEAT

Этот цикл применяется для выполнения группы операторов, пока не будет выполнено определенное условие. Он имеет вид:

```
REPEAT
     оператор-1;
     оператор -2;
     ...
     оператор -n;
UNTIL условие;
```

Как видно из этой формы, цикл может выполнить более одного оператора без применения блоков BEGIN-END. Другое отличие между циклом WHILE и циклом REPEAT в том, что цикл REPEAT выполняется хотя бы один раз независимо от условия, так как каждый проход начинается выполнением оператора и заканчивается проверкой условия. В некоторых приложениях эта возможность необходима.

Взгляните на алгоритм вычисления факториала с использованием цикла REPEAT:

```
  ...
  Factorial := 1;
  Kounter := Number;
REPEAT
  Factorial := Factorial * Kounter;
  Kounter := Kounter - 1;
UNTIL Kounter = 0;
```

Когда Kounter достигает нуля (показывает, что значение 1 уже использовано), то другие проходы уже не нужны, и цикл завершается. Можно также реализовать алгоритм пошагового увеличения, например:

```
...
Factorial := 1;
Kounter := 1;
REPEAT
  Factorial := Factorial * Kounter;
  Kounter := Kounter + 1;
UNTIL Kounter = Number + 1;
```

В этом случае цикл завершается, когда значение Kounter достигает Number+1, и это говорит о том, что значение Number уже было использовано.

В следующей программе цикл REPEAT вложен в цикл WHILE. Программа будет повторно выполняться, пока не будет введен 0 для ее остановки.

```
{ ------------------------------ Пример 4-9 ---------------------------- }
PROGRAM FactorialProg2(INPUT,OUTPUT);
VAR
  Factorial       :REAL;
  Kounter, Number :INTEGER;
BEGIN
  WRITE('Дайте мне число (или 0 для выхода): ');
  READLN(Number);
  WHILE Number<>0 DO { Начало цикла WHILE }
    BEGIN
      Factorial := 1;
      Kounter := 1;
      REPEAT { Начало цикла REPEAT }
        Factorial := Factorial * Kounter;
        Kounter := Kounter + 1;
      UNTIL Kounter = Number + 1; { End of the REPEAT loop }
      WRITELN('Факториал ', Number,' равен ', Factorial:0:0);
      WRITE('Дайте мне число (или 0 для выхода): ');
      READLN(Number)
    END; { Конец цикла WHILE }
  WRITELN('Я вышла!')
END.
```

Обратите внимание, что здесь используются два одинаковых оператора ввода – один перед циклом WHILE, а другой внутри. Первый нужен для присвоения начального значения переменной Kounter перед входом в цикл, чтобы было с чем сравнивать внутри цикла. Результат выполнения программы:

```
Дайте мне число (или 0 для выхода): 3
Факториал из 3 равен 6
Дайте мне число (или 0 для выхода): 5
Факториал из 5 равен 120
Дайте мне число (или 0 для выхода): 0
Я вышла!
```

Задание 4-5

Перепишите предыдущую программу, используя цикл FOR в качестве внутреннего цикла, а цикл WHILE – в качестве внешнего.

Заключение

В этой главе мы рассмотрели три управляющие структуры, предназначенные для построения циклов:

- Цикл FOR
- Цикл WHILE
- Цикл REPEAT

1. Цикл FOR применяется для повторного выполнения оператора или блока операторов определенное число раз:

```
FOR управляющая переменная := выражение-1 TO выражение-2 DO
      оператор;
```

где управляющая переменная – это счетчик цикла, выражение-1 – начальное значение, а выражение-2 – конечное значение.

2. Отрицательное приращение счетчика реализуется при помощи альтернативной формы оператора FOR:

```
FOR управляющая переменная := выражение-1
                DOWNTO выражение-2 DO
      оператор;
```

3. Цикл WHILE предназначен для выполнения оператора или блока операторов, пока определенное условие принимает значение TRUE. Основная форма конструкции:

```
WHILE условие DO
    оператор;
```

4. И в цикле FOR, и в цикле WHILE можно выполнить несколько операторов, помещая их в блок BEGIN-END.

5. Цикл REPEAT служит для выполнения группы операторов, пока определенное условие не примет значение FALSE. Его основная форма:

```
REPEAT
     оператор-1;
     оператор-2;
     оператор-n;
UNTIL условие;
```

6. Теперь вы знаете, что главное отличие цикла REPEAT от двух других операторов в том, что операторы внутри цикла REPEAT выполняются хотя бы один раз независимо от условия.
7. Вы также понимаете, что цикл REPEAT может выполнить несколько операторов без применения блоков BEGIN-END.
8. Наконец, вы изучили в этой главе, что конструкции циклов могут быть вложены в другие конструкции (включая другие циклы).

Упражнения

1. Определите, истинны или ложны следующие утверждения:
   a. Тело цикла WHILE выполняется хотя бы один раз.
   b. Тело цикла REPEAT выполняется хотя бы один раз.
   c. Тело цикла REPEAT, содержащее более одного оператора, не должно быть заключено между BEGIN и END.
   d. Тело цикла WHILE, содержащее более одного оператора, не должно быть заключено между BEGIN и END.
2. Напишите программу, печатающую нечетные (или четные) числа от 1 до 20.
3. Напишите программу, которая читает с клавиатуры 100 целых чисел и прекращает работу, если введен ноль или сто первое число, независимо от того, что произошло сначала.
4. Напишите программу, определяющую население города в конце каждого года с 1990 по 2000 год. Используйте следующие предположения:
   a. Начальная популяция в конце 1989 года = 5 миллионам
   b. Ежегодный процент смертности = 3%
   c. Ежегодный процент рождаемости = 7%

   Программа должна выводить данные на экран следующим образом:

```
 Год     Народонаселение
1990          5349999.97
1991          5724499.94
1992          6125214.90
1993          6553979.92
1994          7012758.48
1995          7503651.54
1996          8028907.12
1997          8590930.59
1998          9192295.70
1999          9835756.37
2000         10524259.29
```

Ответы

1. a. Ложно b. Истинно c. Истинно d. Ложно
2. Следующий фрагмент кода представляет алгоритм печати нечетных чисел от 1 до 20:

```
FOR I := 1 TO 20 DO
  IF (I MOD 2)<>0 THEN
    WRITELN(I);
```

   Для печати четных чисел замените оператор <> на оператор =.

3. Следующий фрагмент кода представляет основной алгоритм программы. Добавьте объявления и инициализацию переменных.

```
WHILE (K < 100) AND (DONE = FALSE) DO
  BEGIN
    READLN(n);
    K := K+1;
    IF N = 0 THEN
      DONE := TRUE
  END;
```

4. Это основной алгоритм вычисления народонаселения. Добавьте объявления и инициализацию переменных.

```
P := 5000000.0 ;
FOR I := 1990 TO 2000 DO
     BEGIN
          B:= 0.07 * P;
          D:= 0.03;
          P:= P + B - D;
          WRITELN(I:10, P:20:2);
     END
```