5. ЦИКЛЫ

Циклический алгоритм – это описание действий, которые в зависимости от исходных данных могут повторяться многократно. Последовательность действий, предназначенная для многократного исполнения, называется **телом цикла**.

Оглавление

§5.1 Цикл while

Итерационные циклы – это алгоритмы, в которых тело цикла выполняется в зависимости от какого-либо условия. Различают циклы с **предусловием** и **постусловием**.

Цикл с предусловием (цикл while) – цикл, который выполняется, пока истинно некоторое условие, указанное перед его началом. Это условие проверяется до выполнения тела цикла, поэтому последнее может быть ни разу не выполнено (если условие с самого начала ложно).

Цикл `while` является самым простым циклов, которые есть в языке C++. Он очень похож на ветвление `if/else`:

```
while (<условие>)
{
    <тело цикла>;
}
```

Цикл `while` объявляется с использованием ключевого слова `while`. В начале цикла обрабатывается условие. Если его значением является true (любое ненулевое значение), то тогда выполняется тело цикла.

Однако, в отличие от оператора `if`, после завершения выполнения тела цикла, управление возвращается обратно к `while` и процесс проверки условия повторяется. Если условие опять является `true`, то тогда тело цикла выполняется еще раз.

Например, программа, представленная в листинге 5.1, выводит все числа от 0 до 9.

Листинг 5.1

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5  int count = 0;
6  while (count < 10)
7  {
8  cout << count << " ";
9  ++count;
10 }
11 cout << "done!";
12 return 0;
13 }
```

Результат выполнения программы:

0 1 2 3 4 5 6 7 8 9 done!

Рассмотрим детально эту программу. Во-первых, инициализируется переменная-счетчик: `int count = 0;`. Условие 0 < 10 имеет значение true, поэтому выполняется тело цикла. В первом выражении (строка 8) мы выводим 0, а во втором (строка 9) — выполняем инкремент переменной `count`. Затем управление возвращается к началу цикла while для повторной проверки условия. Условие 1 < 10 имеет значение true, поэтому тело цикла выполняется еще раз. Тело цикла будет повторно выполняться до тех пор, пока переменная `count` не будет равна 10, только в том случае, когда результат условия 10 < 10 будет false, цикл завершится.

Если условие цикла всегда принимает значение true, то и сам цикл будет выполняться бесконечно. Это называется бесконечным циклом. Пример программы с бесконечным циклом представлен в листинге 5.2.

Листинг 5.2

```
1  #include <iostream>
2  int main()
3  {
4      int count = 0;
5      while (count < 10)
6          cout << count << " ";
```

```
7     return 0;
8 }
```

Поскольку переменная `count` не увеличивается на единицу в этой программе, то условие `count < 10` всегда будет true. Следовательно, цикл никогда не будет завершен, и программа будет постоянно выводить нули.

Мы можем преднамеренно объявить бесконечный цикл следующим образом:

```
while (1) // или while (true)
{
// Этот цикл будет выполняться бесконечно
}
```

Единственный способ выйти из бесконечного цикла — использовать операторы `return`, `break`, `goto`, выбросить исключение или воспользоваться функцией `exit(0)`.

Программы, которые работают до тех пор, пока пользователь не решит остановить их, иногда преднамеренно используют бесконечные циклы вместе с операторами `return`, `break` или функцией `exit()` для завершения цикла. Распространена такая практика в серверных веб-приложениях, которые работают непрерывно и постоянно обслуживают веб-запросы.

Часто нам нужно будет, чтобы цикл выполнялся определенное количество раз. Для этого обычно используется переменная в виде счетчика цикла. **Счетчик цикла** — это целочисленная переменная, которая объявляется с единственной целью: считать, сколько раз выполнился цикл.

Счетчикам цикла часто дают простые имена, такие как i, j или k. А еще лучше использовать «реальные» имена для переменных, например, count или любое другое имя, которое предоставляет контекст использования этой переменной.

Также для счетчиков цикла лучше использовать тип `signed int`. Использование `unsigned int` может привести к неожиданным результатам, например, как в листинге 5.3.

Листинг 5.3

```
1 #include <iostream>
2 using namespace std;
3 int main()
4 {
```

```
5     unsigned int count = 10;
6     // Считаем от 10 к 0
7     while (count >= 0)
8     {
9         if (count == 0)
10            cout << "blastoff!";
11        else
12            cout << count << " ";
13        --count;
14    }
15    return 0;
16 }
```

Оказывается, эта программа представляет собой бесконечный цикл. Она начинается с вывода 10 9 8 7 6 5 4 3 2 1 blastoff! как и предполагалось, но затем начинает отсчет с 4294967295. Условие цикла `count >= 0` никогда не будет ложным. Когда `count = 0`, то и условие `0>=0` имеет значение true, выводится blastoff, а затем выполняется декремент переменной `count,` происходит переполнение и значением переменной становится 4294967295. И так как условие `4294967295>=0` является истинным, то программа продолжает свое выполнение. А поскольку счетчик цикла является типа `unsigned`, то он никогда не сможет быть отрицательным, а так как он никогда не сможет быть отрицательным, то цикл никогда не завершится.

Каждое выполнение цикла называется **итерацией** (или «повтором»).

Поскольку тело цикла обычно является блоком, и поскольку этот блок выполняется по новой с каждым повтором, то любые переменные, объявленные внутри тела цикла, создаются, а затем и уничтожаются по новой. В листинге 5.4 переменная `z` создается и уничтожается 6 раз.

Листинг 5.4

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5      int count = 1;
6      int result = 0;
7      while (count <= 6) // итераций будет 6
8      {
9          int z; // z создается здесь по новой с каждой итерацией
10         cout << "Enter integer #" << count << ':';
```

```
11          cin >> z;
12          result += z;
13          // Увеличиваем значение счетчика цикла на единицу
14          ++count;
15      } // z уничтожается здесь по новой с каждой итерацией
16      cout << "The sum of all numbers entered is: " << result;
17      return 0;
18  }
```

Переменная `count` объявлена вне тела цикла. Это важно и необходимо, поскольку нам нужно, чтобы значение переменной сохранялось на протяжении всех итераций (не уничтожалось по новой с каждым повтором цикла).

Иногда нам может понадобиться выполнить что-то при достижении определенного количества итераций, например, вставить символ новой строки. Это легко осуществить, используя оператор остатка от деления со счетчиком цикла, как представлено в листинге 5.5.

Листинг 5.5

```
1   #include <iostream>
2   using namespace std;
3   int main()
4   {
5       int count = 1;
6       while (count <= 50)
7       {
8           // Выводим числа до 10 (перед каждым числом добавляем 0)
9           if (count < 10)
10              cout << "0" << count << " ";
11          else
12              cout << count << " ";
13  // Если счетчик цикла делится на 10 без остатка, то тогда вставляем символ новой строки
14          if (count % 10 == 0)
15              cout << "\n";
16  // Увеличиваем значение счетчика цикла на единицу
17          ++count;
18      }
19      return 0;
20  }
```

Результат выполнения программы:

01 02 03 04 05 06 07 08 09 10

11 12 13 14 15 16 17 18 19 20

21 22 23 24 25 26 27 28 29 30

31 32 33 34 35 36 37 38 39 40

41 42 43 44 45 46 47 48 49 50

Также одни циклы `while` могут быть вложены внутри других циклов `while`. В листинге 5.6 внутренний и внешний циклы имеют свои собственные счетчики. Однако, обратите внимание, условие внутреннего цикла использует счетчик внешнего цикла.

Листинг 5.6

```
#include <iostream>
using namespace std;
int main()
{
    int outer = 1;
    while (outer <= 5)
    {
        int inner = 1;
        while (inner <= outer)
            cout << inner++ << " ";
// Вставляем символ новой строки в конце каждого ряда
        cout << "\n";
        ++outer;
    }
    return 0;
}
```

Результат выполнения программы:

1

1 2

1 2 3

1 2 3 4

1 2 3 4 5

§5.2 Цикл do-while

Цикл с постусловием (цикл do-while) – цикл, в котором условие проверяется после выполнения тела цикла. Отсюда следует, что тело данного цикла всегда выполняется хотя бы один раз.

Синтаксис цикла имеет следующий вид:

```
do
{
<тело цикла>;
}
while (<условие>);
```

Тело цикла do while всегда выполняется хотя бы один раз. После выполнения тела цикла проверяется условие. Если оно истинно, то выполнение переходит к началу блока do и тело цикла выполняется снова. Если оно ложно, то происходит выход из цикла.

В листинге 5.7 приведен пример использования цикла do while для отображения меню.

Листинг 5.7

```
#include <iostream>
using namespace std;
int main()
{
// Переменная choice должна быть объявлена вне цикла do while
int choice;
do
{
cout << "Please make a selection: \n";
cout << "1) Addition\n";
cout << "2) Subtraction\n";
cout << "3) Multiplication\n";
cout << "4) Division\n";
cin >> choice;
}
while (choice != 1 && choice != 2 &&
choice != 3 && choice != 4);
// Что-то делаем с переменной choice, например, используем оператор switch
cout << "You selected option #" << choice << "\n";
return 0;
}
```

Переменная `choice` должна быть объявлена вне блоков `do while`. Если бы переменная `choice` была объявлена внутри блока `do`, то она была бы уничтожена при завершении этого блока еще до проверки условия `while`. Но нам нужна переменная,

которая будет использоваться в условии while, следовательно, переменная choice должна быть объявлена вне блока do.

§5.3 Цикл for

Цикл со счетчиком (цикл for) – это циклический алгоритм, в котором тело цикла выполняется заранее известное число раз.

Параметр цикла (счетчик) изменяет свое значение от заданного начального до заданного конечного, и для каждого значения этой переменной тело цикла выполняется один раз.

Цикл for в языке C++ идеален, когда известно необходимое количество итераций. Выглядит он следующим образом:

```
for(<объявление переменных>; <условие>;
    <инкремент/декремент счетчика>)
                {
        <тело цикла>;
                }
```

Переменные, определенные внутри цикла for, имеют специальный тип области видимости: область видимости цикла. Такие переменные существуют только внутри цикла и недоступны за его пределами.

Цикл for в C++ выполняется в 3 шага:

Шаг №1: Объявление переменных. Как правило, здесь выполняется определение и инициализация счетчиков цикла, а точнее — одного счетчика цикла. Эта часть выполняется только один раз, когда цикл выполняется впервые.

Шаг №2: Условие. Если оно равно false, то цикл немедленно завершает свое выполнение. Если же условие равно true, то выполняется тело цикла.

Шаг №3: Инкремент/декремент счетчика цикла. Переменная увеличивается или уменьшается на единицу. После этого цикл возвращается к шагу №2.

Рассмотрим пример (листинг 5.8) цикла for и разберемся детально, как он работает.

Листинг 5.8

```
1  #include <iostream>
2  using namespace std;
```

```
3  int main()
4  {
5  for (int count = 0; count < 10; ++count)
6  {
7  cout << count << " ";
8  }
9  return 0;
10 }
```

Сначала мы объявляем переменную count и присваиваем ей значение 0. Далее проверяется условие count < 10, а так как count равен 0, то условие 0 < 10 имеет значение true. Следовательно, выполняется тело цикла, в котором мы выводим в консоль переменную count (т.е. значение 0).

Затем выполняется выражение ++count, т.е. инкремент переменной. Затем цикл снова возвращается к проверке условия. Условие 1 < 10 имеет значение true, поэтому тело цикла выполняется опять. Выводится 1, а переменная count увеличивается уже до значения 2. Условие 2 < 10 является истинным, поэтому выводится 2, а count увеличивается до 3 и так далее.

В конце концов, `count` увеличивается до 10, а условие 10 < 10 является ложным, и цикл завершается. Следовательно, результат выполнения программы:

0 1 2 3 4 5 6 7 8 9

Для наглядности, давайте преобразуем в листинге 5.9 цикл `for`, приведенный выше, в эквивалентный цикл `while`.

Листинг 5.9

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5          int count = 0;
6          while (count < 10)
7          {
8              cout << count << " ";
9              ++count;
10         }
11     return 0;
12 }
```

Давайте, используя цикл `for`, напишем функцию вычисления значений в степени n (листинг 5.10).

Листинг 5.10

```
1  int pow(int base, int exponent)
2  {
3      int total = 1;
4      for (int count=0; count < exponent; ++count)
5          total *= base;
6      return total;
7  }
```

Функция возвращает значение `base^exponent` (число в степени n). `base` — это число, которое нужно возвести в степень, а `exponent` — это степень, в которую нужно возвести `base`.

- Если экспонент равен 0, то цикл `for` выполняется 0 раз, и функция возвращает 1.
- Если экспонент равен 1, то цикл `for` выполняется 1 раз, и функция возвращает 1 * base.
- Если экспонент равен 2, то цикл `for` выполняется 2 раза, и функция возвращает 1 * base * base.

Хотя в большинстве циклов используется инкремент счетчика, мы также можем использовать и декремент счетчика.

Также в циклах можно пропускать одно или сразу все выражения, как это представлено в листинге 5.11.

Листинг 5.11

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5       int count = 0;
6       for (; count < 10; )
7       {
8            cout << count << " ";
9            ++count;
10      }
11      return 0;
```

```
12  }
```

Результат:

0 1 2 3 4 5 6 7 8 9

Инициализацию счетчика мы прописали вне тела цикла, а инкремент счетчика — внутри тела цикла. В самом операторе `for` мы указали только условие. Иногда бывают случаи, когда не требуется объявлять счетчик цикла (потому что у нас он уже есть) или увеличивать его (так как мы увеличиваем его каким-то другим способом).

Хоть это и не часто можно наблюдать, но в операторе `for` можно вообще ничего не указывать. Стоит отметить, что подобное приведет к бесконечному циклу:

```
for (;;)
тело цикла;
```

Хотя в циклах `for` обычно используется только один счетчик, иногда могут возникать ситуации, когда нужно работать сразу с несколькими переменными. Для этого используется оператор Запятая, который позволяет обрабатывать несколько выражений. Пример использования данного оператора представлен в листинге 5.12.

Листинг 5.12

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5      int aaa, bbb;
6      for (aaa = 0, bbb = 9; aaa < 10; ++aaa, --bbb)
7          cout << aaa << " " << bbb << endl;
8      return 0;
9  }
```

Этот цикл присваивает значения двум ранее объявленным переменным: `aaa = 0` и `bbb = 9`. Только с каждой итерацией переменная `aaa` увеличивается на единицу, а `bbb` — уменьшается на единицу.

Подобно другим типам циклов, одни циклы `for` могут быть вложены в другие циклы `for`. В листинге 5.13 мы разместили один `for` внутри другого `for`:

Листинг 5.13

```
1  #include <iostream>
2  using namespace std;
```

```
3  int main()
4  {
5      for (char c = 'a'; c <= 'e'; ++c)
6      {
7          cout << c;
8          for (int i = 0; i < 3; ++i)
9              cout << i;
10         cout << '\n';
11     }
12     return 0;
13 }
```

С одной итерацией внешнего цикла выполняется три итерации внутреннего цикла. Следовательно, результат выполнения программы:

a012

b012

c012

d012

e012

§5.4 Операторы break и continue

С оператором `break` мы познакомились, когда изучали оператор множественного выбора `switch`. В теле оператора множественного выбора `switch` оператор `break` прерывал исполнение оператора `switch`.

Когда оператор `break` выполняется в цикле, то досрочно прерывается исполнение оператора цикла, и управление передаётся следующему оператору после цикла.

Разработаем программу (листинг 5.14) с использованием оператора `break`. Программа печатает таблицу степеней двойки.

Листинг 5.14

```
1  #include <iostream>
2  #include <cmath>
3  using namespace std;
4  int main()
5  {
6  for (int count = 0; count <= 10; count++) // начало цикла for
7  {
8  if (count == 8)
9  break; // выход из цикла for
10 cout << "2^" << count << " = " << pow(2.0,count) << endl;
```

```
11  }
12  return 0;
13  }
```

В строке 6 записан заголовок цикла `for`. В цикле `for` объявлена переменная-счётчик `count`, значение которой меняется от 0 до 10 включительно. В строке 8 записан оператор условного выбора `if`, истинность условия которого запускает оператор `break`, который, в свою очередь, приводит к выходу из цикла `for`. В строке 10 запускается функция возведения в степень `pow()`. Условие продолжения цикла `for` будет истинно до тех пор, пока значение в переменной `count <= 10`. Тогда как, выход из цикла `for` произойдёт раньше, чем условие продолжения цикла станет ложным. Выход из цикла `for` выполнится, когда значение в переменной `count` станет равным шести.

Оператор `continue` используется только в циклах. В операторах `for`, `while`, `do while`, оператор `continue` выполняет пропуск оставшейся части кода тела цикла и переходит к следующей итерации цикла. Рассмотрим фрагмент кода (листинг 5.15) с оператором `continue`.

Листинг 5.15

```
1  // пример использования оператора continue:
2  int count = 0;
3  do // начало цикла do while
4  {
5  continue;
6  count++;
7  }
8  while (count  <  10)
```

Посмотрите внимательно на вышеприведенный пример, и Вы увидите, что `do while` бесконечный, так как каждая итерация цикла приводит к выполнению оператора `continue`, который пропускает операцию инкремента переменной-счётчика `count` и переходит на следующую итерацию цикла. Таким образом значение в переменной `count` не меняется, а значит и условие всегда будет истинным.

Разработаем программу (листинг 5.16) с оператором `continue`. Программа должна работать циклически. Внутри цикла необходимо организовать ввод чисел. Если введено число от 0 до 10 включительно, то необходимо напечатать квадрат этого числа, иначе используя оператор `continue` пропустить оператор возведения в квадрат введенного числа. При введении отрицательного числа осуществить выход из цикла.

Листинг 5.16

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5  int in_number; // переменная для хранения введённого числа
6  do
7  {
8  cout << "Enter number: ";
9  cin >> in_number; // считываем введённое число в переменную in_number
10 if (in_number > 10 || in_number < 0) // если введённое число не входит в заданный интервал
11 continue; // переход на следующую итерацию цикла do while
12 cout << "square = " << in_number * in_number << endl; // возводим в квадрат введённое число
13 } while (in_number >= 0); // пока введённое число больше либо равно нулю цикл будет работать
14 return 0;
15 }
```

Цикличность в программе организуем циклом с постусловием — `do while`. В цикле сначала считываем введённое число в переменную `in_number`, после чего, выполняется проверка условия в операторе `if`. Условие оператора условного выбора `if` будет истинным в том случае, если введённое число будет строго меньше нуля или строго больше 10. Заданный интервал — $[0; 10]$, число, взятое из этого интервала, возводится в квадрат. Истинность условия оператора `if` приводит к выполнению оператора `continue` в строке 11. А оператор `continue` пропускает операторы в строке 12 и переходит к проверке условия продолжения цикла `do while`. Условие в цикле будет истинно, пока вводимые числа будут строго больше 0.