Циклы являются управляющими конструкциями, позволяя в зависимости от определенных условий выполнять некоторое действие множество раз. В C# имеются следующие виды циклов:

* for
* foreach
* while
* do...while

### Цикл for

Цикл for имеет следующее формальное определение:

|  |  |
| --- | --- |
| 1  2  3  4 | for ([действия\_до\_выполнения\_цикла]; [условие]; [действия\_после\_выполнения])  {  // действия  } |

Объявление цикла **for** состоит из трех частей. Первая часть объявления цикла - некоторые действия, которые выполняются один раз до выполнения цикла. Обычно здесь определяются переменные, которые будут использоваться в цикле.

Вторая часть - условие, при котором будет выполняться цикл. Пока условие равно true, будет выполняться цикл.

И третья часть - некоторые действия, которые выполняются после завершения блока цикла. Эти действия выполняются каждый раз при завершении блока цикла.

После объявления цикла в фигурных скобках помещаются сами действия цикла.

Рассмотрим стандартный цикл for:

|  |  |
| --- | --- |
| 1  2  3  4 | for (int i = 1; i < 4; i++)  {  Console.WriteLine(i);  } |

Здесь первая часть объявления цикла - int i = 1 - создает и инициализирует переменную i.

Вторая часть - условие i < 4. То есть пока переменная i меньше 4, будет выполняться цикл.

И третья часть - действия, выполняемые после завершения действий из блока цикла - увеличение переменной i на единицу.

Весь процесс цикла можно представить следующим образом:

1. Определяется переменная int i = 1
2. Проверяется условие i < 4. Оно истинно (так как 1 меньше 4), поэтому выполняется блок цикла, а именно инструкция Console.WriteLine(i), которая выводит на консоль значение переменной i
3. Блок цикла закончил выполнение, поэтому выполняется треться часть объявления цикла - i++. После этого переменная i будет равна 2.
4. Снова проверяется условие i < 4. Оно истинно (так как 2 меньше 4), поэтому опять выполняется блок цикла - Console.WriteLine(i)
5. Блок цикла закончил выполнение, поэтому снова выполняется выражение i++. После этого переменная i будет равна 3.
6. Снова проверяется условие i < 4. Оно истинно (так как 3 меньше 4), поэтому опять выполняется блок цикла - Console.WriteLine(i)
7. Блок цикла закончил выполнение, поэтому снова выполняется выражение i++. После этого переменная i будет равна 4.
8. Снова проверяется условие i < 4. Теперь оно возвражает false, так как значение переменной i НЕ меньше 4, поэтому цикл завершает выполнение. Далее уже выполняется остальная часть программы, которая идет после цикла

В итоге блок цикла сработает 3 раза, пока значение i не станет равным 4. И каждый раз это значение будет увеличиваться на 1. Однократное выполнение блока цикла называется **итерацией**. Таким образом, здесь цикл выполнит три итерации. Результат работы программы:

1  
2  
3

Если блок цикла for содержит одну инструкцию, то мы можем его сократить, убрав фигурные свобки:

|  |  |
| --- | --- |
| 1  2  3  4  5 | for (int i = 1; i < 4; i++)  Console.WriteLine(i);    // или так  for (int i = 1; i < 4; i++) Console.WriteLine(i); |

При этом необязательно именно в первой части цикла объявлять переменную, а в третий части изменять ее значение - это могут быть любые действия. Например:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | var i = 1;    for (Console.WriteLine("Начало выполнения цикла"); i < 4; Console.WriteLine($"i = {i}"))  {  i++;  } |

Здесь опять же цикл срабатывает, пока переменная i меньше 4, только приращение переменной i происходит в блоке цикла. Консольный вывод данной программы:

Начало выполнения цикла  
i = 2  
i = 3  
i = 4

Нам необязательно указывать все условия при объявлении цикла. Например, мы можем написать так:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | int i = 1;  for (; ;)  {  Console.WriteLine($"i = {i}");  i++;  } |

Формально определение цикла осталось тем же, только теперь блоки в определении у нас пустые: for (; ;). У нас нет инициализированной переменной, нет условия, поэтому цикл будет работать вечно - бесконечный цикл.

Мы также можем опустить ряд блоков:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | int i = 1;  for (; i<4;)  {  Console.WriteLine($"i = {i}");  i++;  } |

Этот пример по сути эквивалентен первому примеру: у нас также есть переменная-счетчик, только определена она вне цикла. У нас есть условие выполнения цикла. И есть приращение переменной уже в самом блоке for.

Также стоит отметить, что можно определять несколько переменных в объявлении цикла:

|  |  |
| --- | --- |
| 1  2 | for (int i = 1, j = 1; i < 10; i++, j++)  Console.WriteLine($"{i \* j}"); |

Здесь в первой части объявления цикла определяются две переменных: i и j. Цикл выполняется, пока i не будет равна 10. После каждой итерации переменые i и j увеличиваются на единицу. Консольный вывод программы:

1  
4  
9  
16  
25  
36  
49  
64  
81

### Цикл do..while

В цикле do сначала выполняется код цикла, а потом происходит проверка условия в инструкции while. И пока это условие истинно, цикл повторяется.

|  |  |
| --- | --- |
| 1  2  3  4  5 | do  {  действия цикла  }  while (условие) |

Например:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | int i = 6;  do  {  Console.WriteLine(i);  i--;  }  while (i > 0); |

Здесь код цикла сработает 6 раз, пока i не станет равным нулю. Но важно отметить, что цикл do гарантирует хотя бы единократное выполнение действий, даже если условие в инструкции while не будет истинно. То есть мы можем написать:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | int i = -1;  do  {  Console.WriteLine(i);  i--;  }  while (i > 0); |

Хотя у нас переменная i меньше 0, цикл все равно один раз выполнится.

### Цикл while

В отличие от цикла do цикл **while** сразу проверяет истинность некоторого условия, и если условие истинно, то код цикла выполняется:

|  |  |
| --- | --- |
| 1  2  3  4 | while (условие)  {  действия цикла  } |

Например:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | int i = 6;  while (i > 0)  {  Console.WriteLine(i);  i--;  } |

### Цикл foreach

Цикл foreach предназначен для перебора набора или коллекции элементов. Его общее определение:

|  |  |
| --- | --- |
| 1  2  3  4 | foreach(тип\_данных переменная in коллекция)  {  // действия цикла  } |

После оператора foreach в скобках сначала идет определение переменной. Затем ключевое слово **in** и далее коллекция, элементы которой надо перебрать.

При выполнении цикл последовательно перебирает элементы коллекции и помещает их в переменную, и таким образом в блоке цикла мы можем выполнить с ними некоторые действия.

Например, возьмем строку. Строка по сути - это коллекция символов. И .NET позволяет перебрать все элементы строки - ее символы с помощью цикла **foreach**.

|  |  |
| --- | --- |
| 1  2  3  4 | foreach(char c in "Tom")  {  Console.WriteLine(c);  } |

Здесь цикл foreach пробегается по всем символам строки "Tom" и каждый символ помещает в символьную переменную c. В блоке цикла значение переменной c выводится на консоль. Поскольку в строке "Tom" три символа, то цикл выполнится три раза. Консольный вывод программы:

T  
o  
m

Стоит отметить, что переменная, которая определяется в объявлении цикла, должна по типу соответствовать типу элементов перебираемой коллекции. Так, элементы строки - значения типа char - символы. Поэтому переменная c имеет тип char. Однако в реальности не всегда бывает очевидно, какой тип представляют элементы коллекции. В этом случае мы можем определить переменную с помощью оператора **var**:

|  |  |
| --- | --- |
| 1  2  3  4 | foreach(var c in "Tom")  {  Console.WriteLine(c);  } |

В дальнейшем мы подробнее рассмотрим, что представляют собой коллекции в .NET и какие именно коллекции можно перебирать с помощью цикла foreach.

### Операторы continue и break

Иногда возникает ситуация, когда требуется выйти из цикла, не дожидаясь его завершения. В этом случае мы можем воспользоваться оператором **break**.

Например:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | for (int i = 0; i < 9; i++)  {  if (i == 5)  break;  Console.WriteLine(i);  } |

Хотя в условии цикла сказано, что цикл будет выполняться, пока счетчик i не достигнет значения 9, в реальности цикл сработает 5 раз. Так как при достижении счетчиком i значения 5, сработает оператор break, и цикл завершится.

0  
1  
2  
3  
4

Теперь поставим себе другую задачу. А что если мы хотим, чтобы при проверке цикл не завершался, а просто пропускал текущую итерацию. Для этого мы можем воспользоваться оператором **continue**:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | for (int i = 0; i < 9; i++)  {  if (i == 5)  continue;  Console.WriteLine(i);  } |

В этом случае цикл, когда дойдет до числа 5, которое не удовлетворяет условию проверки, просто пропустит это число и перейдет к следующей итерации:

0  
1  
2  
3  
4  
6  
7  
8

Стоит отметить, что операторы break и continue можно применять в любом типе циклов.

### Вложенные циклы

Одни циклы могут быть вложенными в другие. Например:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | for (int i = 1; i < 10; i++)  {  for (int j = 1; j < 10; j++)  {  Console.Write($"{i \* j} \t");  }  Console.WriteLine();  } |

В данном случае цикл for (int i = 1; i < 10; i++) выполняется 9 раз, то есть имеет 9 итераций. Но в рамках каждой итерации выполняется девять раз вложенный цикл for (int j = 1; j < 10; j++). В итоге данная программа выведет таблицу умножения.