

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

- for
- foreach
- while
- do...while

Цикл for

Цикл for имеет следующее формальное определение:

```
for ([инициализация счетчика]; [условие]; [изменение счетчика])
{
    // действия
}
```

Рассмотрим стандартный цикл for:

```
for (int i = 0; i < 9; i++)
{
    Console.WriteLine($"Квадрат числа {i} равен {i*i}");
}
```

Первая часть объявления цикла - `int i = 0` - создает и инициализирует счетчик `i`. Счетчик необязательно должен представлять тип **int**. Это может быть и другой числовой тип, например, `float`. И перед выполнением цикла его значение будет равно 0. В данном случае это то же самое, что и объявление переменной.

Вторая часть - условие, при котором будет выполняться цикл. Пока условное выражение возвращает `true`, будет выполняться цикл. В данном случае цикл будет выполняться, пока счетчик `i` не достигнет 9.

И третья часть - приращение счетчика на единицу. Опять же нам необязательно увеличивать на единицу. Можно уменьшать: `i--`.

В итоге блок цикла сработает 9 раз, пока значение `i` не станет равным 9. И каждый раз это значение будет увеличиваться на 1.

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

```
int i = 0;
for (; ; )
{
    Console.WriteLine($"Квадрат числа {++i} равен {i * i}");
}
```

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

Мы также можем опустить ряд блоков:

```
int i = 0;
for (; i < 9; )
{
```

```
        Console.WriteLine($"Квадрат числа {++i} равен {i * i}");  
    }  
}
```

Этот пример по сути эквивалентен первому примеру: у нас также есть счетчик, только создан он вне цикла. У нас есть условие выполнения цикла. И есть приращение счетчика уже в самом блоке `for`.

Цикл `do`

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

```
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:

```
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:

```
int i = 6;  
while (i > 0)  
{  
    Console.WriteLine(i);  
    i--;  
}
```

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:

```
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`**:

```
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
```

Цикл `foreach`

Цикл `foreach` перебирает коллекции, например, массивы, и будет рассмотрен далее в теме массивов.