Лекция 5: Управление ходом выполнения программы. Циклы.

От выбора к повторению

На прошлом занятии мы научили наши программы принимать решения, используя операторы ветвления `if`, `else` и `switch`. Теперь наша программа может пойти по тому или иному пути. Но что, если какое-то действие нужно выполнить не один, а много раз? Сегодня мы изучим второй краеугольный камень структурного программирования — **циклы**.

«Это то, что действительно радует в работе с компьютерами. Они не спорят, они все помнят, и они никогда не выпьют все ваше пиво.» — Пол Лерп.

1. Алгоритмическая основа: Зачем нужны циклы?

Представьте, что вам нужно:

- Вывести на экран 100 раз фразу "Я больше не буду опаздывать на уроки!".
- Перебрать всех студентов в списке и выставить им оценки.
- Запрашивать у пользователя пароль до тех пор, пока он не введет правильный.

Во всех этих случаях мы имеем дело с **многократным повторением одних и тех же или похожих действий**.

Цикл — это алгоритмическая конструкция, позволяющая выполнять одну и ту же последовательность действий несколько раз до тех пор, пока выполняется некоторое условие.

Ключевые компоненты любого цикла:

1. **Тело цикла** — те самые инструкции, которые нужно повторять.
2. **Условие продолжения** — логическое выражение, которое проверяется на каждой итерации (повторении). Если оно истинно (`true`), цикл выполняется снова.
3. **Счетчик или итератор** (не всегда, но часто) — переменная, которая меняется с каждым шагом цикла и в конечном счете приводит к изменению условия на `false`.

2. Цикл `for`: Цикл со счетчиком

Цикл `for` — это идеальный инструмент, когда вы заранее знаете, сколько раз нужно выполнить тело цикла.

Синтаксис:

```
for (инициализация; условие; итератор)
{
    // Тело цикла
}
```

Как это работает:

1. **Инициализация:** Выполняется один раз перед началом цикла. Здесь обычно объявляется и инициализируется счетчик.

2. **Условие:** Проверяется **перед каждой** итерацией. Если `true` — тело цикла выполняется. Если `false` — цикл завершается.
3. **Тело цикла:** Выполняются нужные инструкции.
4. **Итератор:** Выполняется **после каждой** итерации. Здесь обычно изменяется счетчик (увеличивается или уменьшается).
5. ...и снова шаг 2 (Проверка условия).

Классический пример: Вывод чисел от 1 до 10

```
for (int i = 1; i <= 10; i++)
{
    Console.WriteLine(i);
}
```

Разберем по шагам:

1. `int i = 1` — создается переменная-счетчик `i` со значением 1.
2. `i <= 10` — проверка: 1 <= 10? `true`.
3. Выполняется `Console.WriteLine(1);`.
4. `i++` — значение `i` увеличивается на 1. Теперь `i = 2`.
5. `i <= 10` — проверка: 2 <= 10? `true`.
6. Выполняется `Console.WriteLine(2);`.
7. ... и так далее, пока после шага `i++` значение `i` не станет равным 11.
8. `i <= 10` — проверка: 11 <= 10? `false`. Цикл завершается.

Пример: Вычисление суммы чисел от 1 до N

```
int sum = 0; // Переменная-аккумулятор для суммы
int N = 100;

for (int i = 1; i <= N; i++)
{
    sum += i; // sum = sum + i;
}

Console.WriteLine($"Сумма чисел от 1 до {N} равна {sum}");
```

3. Цикл `while`: Цикл с предусловием

Цикл `while` используется тогда, когда количество повторений **заранее неизвестно**, но известно **условие**, при котором их нужно продолжать.

Синтаксис:

```
while (условие)
{
    // Тело цикла
}
```

Как это работает:

1. Проверяется условие.
2. Если оно истинно (`true`), выполняется тело цикла.
3. После этого управление снова переходит к проверке условия.
4. Цикл продолжается, пока условие не станет ложным (`false`).

Пример: Игра "Угадай число"

```
Random rnd = new Random();
int secretNumber = rnd.Next(1, 101); // Случайное число от 1 до 100
int userGuess = 0;

Console.WriteLine("Угадайте число от 1 до 100!");

while (userGuess != secretNumber)
{
    Console.Write("Ваш вариант: ");
    userGuess = int.Parse(Console.ReadLine());

    if (userGuess < secretNumber)
        Console.WriteLine("Загаданное число БОЛЬШЕ.");
    else if (userGuess > secretNumber)
        Console.WriteLine("Загаданное число МЕНЬШЕ.");
}

Console.WriteLine("Поздравляем! Вы угадали!");
```

Важный момент: В теле цикла `while` должно быть что-то, что в конечном счете сделает условие ложным. Иначе цикл станет **бесконечным**.

4. Цикл `do...while`: Цикл с постусловием

Этот цикл похож на `while`, но с ключевым отличием: условие проверяется **после** выполнения тела цикла. Это значит, что тело цикла гарантированно выполнится **хотя бы один раз**.

Синтаксис:

```
do
{
    // Тело цикла
} while (условие);
```

Пример: Запрос пароля

```
string correctPassword = "12345";
string inputPassword;

do
{
    Console.Write("Введите пароль: ");
    inputPassword = Console.ReadLine();

    if (inputPassword != correctPassword)
        Console.WriteLine("Неверный пароль! Попробуйте еще раз.");

} while (inputPassword != correctPassword);

Console.WriteLine("Доступ разрешен!");
```

Сравнение `while` и `do...while`:

- `while`: "Пока здоров, делай зарядку". (Может быть, ты уже болен и не сделаешь ни разу).
- `do...while`: "Сделай зарядку, а потом проверь, здоров ли ты". (Зарядку ты сделаешь в любом случае).

5. Цикл `foreach`: Цикл для перебора коллекций

Цикл `foreach` предназначен для простого и безопасного перебора всех элементов в коллекции (например, в массиве, списке, строке). Мы подробнее изучим коллекции позже, но познакомиться с циклом стоит уже сейчас.

Синтаксис:

```
foreach (тип_элемента переменная in коллекция)
```

```
{
    // Тело цикла, где 'переменная' — текущий элемент
}
```

Пример: Перебор символов в строке

```
string message = "Hello, World!";

foreach (char symbol in message)
{
    Console.Write(symbol + " "); // Выведет: H e l l o ,   W o r l d !
}
```

Главное преимущество `foreach` — нам не нужно заботиться о индексах или размере коллекции. Цикл сам пройдет от начала до конца.

6. Управление выполнением циклов: `break` **и** `continue`

Иногда внутри цикла требуется досрочно прервать его выполнение или пропустить текущую итерацию. Для этого существуют операторы `break` и `continue`.

`break` **— немедленный выход из цикла.**

```
// Поиск первого числа, делящегося на 13 в диапазоне от 1000 до 1100
for (int i = 1000; i <= 1100; i++)
{
    if (i % 13 == 0)
    {
        Console.WriteLine($"Первое найденное число: {i}");
        break; // Как только нашли, выходим из цикла, дальше не ищем
    }
}
```

`continue` **— переход к следующей итерации цикла.**

```
// Вывод всех нечетных чисел от 1 до 10
for (int i = 1; i <= 10; i++)
{
    if (i % 2 == 0) // Если число четное...
```

```
    {
        continue; // ...пропускаем его и переходим к следующему (i+1)
    }
    Console.WriteLine(i); // Эта строка выполнится только для нечетных i
}
// Выведет: 1, 3, 5, 7, 9
```

Аналогия:

- `break` — как аварийный выход из здания. Ты выходишь и не возвращаешься.
- `continue` — как пропуск одного этажа в лифте. Ты едешь дальше к следующему.

7. Эффективность и время работы циклов

Циклы — это мощно, но они могут быть источником медленной работы программы, если их использовать неэффективно. Давайте рассмотрим основные принципы.

1. Минимизация операций внутри цикла

Все, что можно вычислить **один раз до цикла**, должно быть вычислено до него.

Плохо:

```
for (int i = 0; i < collection.Length; i++) // Length вычисляется на каждой итерации!
{
    // ...
}
```

Хорошо:

```
int length = collection.Length; // Вычислили длину ОДИН раз
for (int i = 0; i < length; i++)
{
    // ...
}
```

2. Избегайте вложенных циклов высокой сложности

Вложенные циклы (цикл внутри цикла) резко увеличивают количество операций. Если внешний цикл выполняется N раз, а внутренний M раз, то общее количество итераций будет N * M.

```
// Пример: Сложность O(n^2)
for (int i = 0; i < n; i++)
```

```
{
    for (int j = 0; j < n; j++)
    {
        // ... какая-то работа
    }
}
// Количество итераций: n * n = n²
```

Если `n = 1000`, то итераций будет 1 000 000. Если `n = 10000`, то уже 100 000 000. Это очень быстро замедляет программу. Старайтесь пересматривать алгоритмы, чтобы избегать таких "тяжелых" вложенных циклов.

3. Бесконечные циклы

Это циклы, условие выхода из которых никогда не выполняется. Обычно это ошибка.

```
// ОСТОРОЖНО! БЕСКОНЕЧНЫЙ ЦИКЛ!
while (true)
{
    Console.WriteLine("Помогите!");
}
```

Такие циклы могут "подвесить" программу. Всегда проверяйте, есть ли в цикле `while` или `do...while` механизм, который в итоге сделает условие ложным.

Сводная таблица: Какой цикл когда использовать?

Цикл	Когда использовать?	Преимущества	Недостатки
`for`	Когда известно **количество** итераций.	Полный контроль над счетчиком. Компактный синтаксис (все управление в одной строке).	Не так гибок, когда количество итераций неизвестно.
`while`	Когда количество итераций **неизвестно**, но известно **условие продолжения**.	Очень гибкий. Условие может быть любым.	Риск бесконечного цикла, если условие всегда true.
`do...while`	Когда тело цикла **должно выполниться хотя бы один раз**.	Гарантирует хотя бы одно выполнение.	Используется реже, чем `while`.

Цикл	Когда использовать?	Преимущества	Недостатки
`foreach`	Для **последовательного перебора всех элементов** коллекции.	Простота и безопасность. Не нужно управлять индексами.	Не позволяет напрямую изменять саму коллекцию. Меньше контроля над порядком (только "с начала до конца").

Практический совет: Отладка (Debug)

Лучший способ понять, как работает цикл — использовать **отладку**.

1. Поставьте **точку останова** (щелкните слева от номера строки в Visual Studio).
2. Запустите программу в режиме отладки (F5).
3. Используйте **F10** (Шаг с обходом) и **F11** (Шаг с заходом), чтобы выполнять код пошагово.
4. Наблюдайте, как меняются значения переменных в окне "Локальные" или "Контрольные значения".

Итог

Сегодня мы с вами научились заставлять компьютер повторять действия многократно, что открывает перед нами огромные возможности. Мы изучили:

1. **Алгоритмическую суть** циклов.
2. Цикл со счетчиком `for`.
3. Циклы с условием `while` и `do...while`.
4. Цикл для перебора коллекций `foreach`.
5. Операторы управления `break` и `continue`.
6. Основы **эффективности** при работе с циклами.

Главный принцип: Выбирайте тип цикла в зависимости от задачи. Четко понимайте, когда ваш цикл должен остановиться.

На следующей лекции мы наконец-то познакомимся со строками, и методами работы с ними.