Лекция 4: Управление ходом выполнения программы. Ветвления.

От последовательности к выбору

На предыдущих занятиях мы с вами научились создавать переменные, производить над ними операции и писать простые линейные программы, которые выполняются строго сверху вниз, от первой команды к последней. Но мир нелинеен, и задачи требуют принятия решений. Сегодня мы научимся управлять ходом выполнения нашей программы, заставляя её делать то или иное действие в зависимости от условий.

«Простота — великое достоинство, но требуется упорный труд для её достижения и образование для её осознания. Увы, сложность продаётся лучше.» — Эдсгер Дейкстра.

1. Алгоритмическая основа: Принятие решений

Представьте себе обычное утро. Ваш алгоритм «Собраться в школу» может иметь развилку:

1. Проснуться.
2. **Если на улице дождь, то взять зонт.**
3. **Иначе, если солнечно, надеть солнцезащитные очки.**
4. Пойти в школу.

Ключевой момент здесь — **проверка условия**. В зависимости от его истинности (true) или ложности (false) выполняется та или иная ветка действий.

В программировании этот механизм называется **ветвлением** или **условными операторами**.

2. Базовый строительный блок: Оператор `if`

Самый фундаментальный инструмент для создания ветвлений в C# — это оператор `if` (с английского «если»).

Синтаксис:

```
if (условие)
{
    // Блок кода, выполняемый, если условие ИСТИННО (true)
}
```

Как это работает:

1. Программа доходит до ключевого слова `if`.
2. Вычисляется логическое выражение в круглых скобках. Оно должно возвращать `true` или `false`.
3. **Если результат `true`**, выполняется блок кода в фигурных скобках `{}`.
4. **Если результат `false`**, блок кода пропускается, и программа выполняется дальше.

Простой пример:

```
int temperature = 25;

if (temperature > 20)
{
    Console.WriteLine("На улице тепло. Можно идти в футболке.");
}
```

В этом примере сообщение выведется только если температура больше 20 градусов.

3. Расширяем выбор: Оператор `if-else`

Часто требуется предусмотреть не только действие для истинного условия, но и альтернативное действие на случай, если условие ложно. Для этого используется связка `if-else` («если-иначе»).

Синтаксис:

```
if (условие)
{
    // Блок кода, если условие ИСТИННО (true)
}
else
{
    // Блок кода, если условие ЛОЖНО (false)
}
```

Пример с погодой:

```
bool isRaining = true;

if (isRaining)
{
    Console.WriteLine("Идет дождь. Возьми зонт!");
}
else
{
    Console.WriteLine("Дождя нет. Зонт можно оставить дома.");
}
```

4. Множественный выбор: Конструкция `if - else if - else`

Что делать, если вариантов больше двух? Например, нам нужно оценить успеваемость студента по баллам. Для этого мы можем выстроить цепочку условий с помощью `else if`.

Синтаксис:

```
if (условие1)
{
    // Выполняется, если условие1 истинно
}
else if (условие2)
{
    // Выполняется, если условие1 ложно, но условие2 истинно
}
else if (условие3)
{
    // Выполняется, если условие1 и условие2 ложны, но условие3 истинно
}
else
{
    // Выполняется, если ВСЕ вышеперечисленные условия ложны
}
```

Важно: Проверка условий идет **сверху вниз**. Как только находится первое истинное условие, выполняется соответствующий ему блок кода, а все последующие `else if` и `else` **игнорируются**.

Пример с оценками:

```
int score = 85;


if (score >= 90)
{
    Console.WriteLine("Оценка: Отлично (5)");
}
else if (score >= 70)
```

```
{
    Console.WriteLine("Оценка: Хорошо (4)");
}
else if (score >= 50)
{
    Console.WriteLine("Оценка: Удовлетворительно (3)");
}
else
{
    Console.WriteLine("Оценка: Неудовлетворительно (2)");
}
// В данном случае будет выведено: "Оценка: Хорошо (4)"
```

«Сложность — враг исполнения.» — Тони Роббинс.

5. Продвинутые условия: Составные логические выражения

Условия внутри `if` далеко не всегда простые. Мы можем комбинировать проверки с помощью логических операторов `&&` (И), `||` (ИЛИ) и `!` (НЕ), которые мы изучали на прошлой лекции.

Пример: Идеальная погода для прогулки

```
int temperature = 22;
bool isSunny = true;
bool isWindy = false;

// Идеальная погода: тепло, солнечно и не ветрено.
if (temperature > 20 && isSunny && !isWindy)
{
    Console.WriteLine("Идеальный день для пикника!");
}
else
{
    Console.WriteLine("Может, лучше остаться дома?");
}
```

6. Особенности синтаксиса: Фигурные скобки `{}`

- **Если блок кода содержит только ОДНУ команду**, фигурные скобки `{}` можно опустить. Компилятор поймет, что условие относится только к следующей строке.

```
int x = 10;
if (x > 5)
    Console.WriteLine("X больше пяти"); // Одна команда - скобки не нужны
else
    Console.WriteLine("X меньше или равен пяти");
```

- **Однако, для читаемости и избежания ошибок, особенно новичкам, рекомендуется СТАВИТЬ скобки ВСЕГДА.** Это предотвращает трудноуловимые баги, когда вы добавляете вторую строку, забывая, что она уже не относится к условию.

```
// Опасный пример без скобок
if (isReady)
    Console.WriteLine("Всё готово!");
    StartEngine(); // ЭТА КОМАНДА ВЫПОЛНИТСЯ ВСЕГДА, независимо от isReady!
```

Рассуждение программиста про стили скобок.

7. Оператор выбора `switch`

Когда у нас много условий, которые проверяют **одну и ту же переменную** на равенство разным значениям, длинная цепочка `if - else if` может стать громоздкой.
Альтернатива — оператор `switch`.

Синтаксис:

```
switch (выражение)
{
    case значение1:
        // Код, если выражение == значение1
        break;
    case значение2:
        // Код, если выражение == значение2
        break;
```

```
    ...
    default:
        // Код, если ни один case не подошел (аналог else)
        break;
}
```

Ключевые слова:

- `case` — ветка, соответствующая определенному значению.
- `break` — обязательная команда, которая прерывает выполнение `switch` и передает управление дальше.
- `default` — необязательная ветка, которая выполняется, если ни один из `case` не сработал.

Пример с днями недели:

```
int dayOfWeek = 3;

switch (dayOfWeek)
{
    case 1:
        Console.WriteLine("Понедельник");
        break;
    case 2:
        Console.WriteLine("Вторник");
        break;
    case 3:
        Console.WriteLine("Среда"); // Выполнится этот блок
        break;
    case 4:
        Console.WriteLine("Четверг");
        break;
    case 5:
        Console.WriteLine("Пятница");
        break;
    default:
```

```
        Console.WriteLine("Выходной день");
        break;
}
```

Современный `switch` в C# (начиная с версии 8.0)

Начиная с C# 8.0, появились более компактные и выразительные формы записи `switch`:

```
string weatherDescription = "Sunny";
string action = weatherDescription switch
{
    "Rainy" => "Take an umbrella",
    "Sunny" => "Wear sunglasses",
    "Snowy" => "Wear a coat",
    _ => "Stay home" // _ - аналог default
};

Console.WriteLine(action); // Выведет: Wear sunglasses
```

Пока мы будем использовать классический синтаксис, чтобы лучше понять основы, но знать о новых возможностях полезно.

«Искусство программирования — это искусство организации сложности.» — Эдсгер Дейкстра.

8. Практикуемся: от простого к сложному

Давайте закрепим материал на практических примерах. (Примеры кода можно найти на гитхабе)

9. Итог и выводы

Сегодня мы с вами сделали огромный шаг вперед, перестав быть простыми наблюдателями и став настоящими «повелителями кода». Мы научились управлять ходом его выполнения.

Что мы узнали:

1. **Алгоритмическую суть ветвлений** — принятие решений в программе.
2. **Базовый оператор** `if` для выполнения кода при истинности условия.
3. **Конструкцию** `if-else` для реализации двух альтернативных путей.
4. **Цепочку** `if - else if - else` для множественного выбора.
5. **Использование логических операторов** (`&&`, `||`, `!`) для составления сложных условий.
6. **Оператор выбора** `switch` как альтернативу длинным цепочкам `if-else` при проверке на равенство.

Главные принципы хорошего кода с ветвлениями:

- **Читаемость:** Делайте отступы для блоков кода внутри `if`/`else`.
- **Простота (KISS):** Не создавайте слишком сложных вложенных условий. Если условие стало громоздким, вынесите его в отдельную логическую переменную с понятным именем.

```
// Плохо:
if (age > 18 && hasLicense == true && isDrunk == false && carIsWorking == true) { ... }

// Лучше:
bool canDrive = age > 18 && hasLicense && !isDrunk && carIsWorking;
if (canDrive) { ... }
```

- **Использование `default` в `switch`:** Всегда предусматривайте ветку по умолчанию для обработки неожиданных значений.

На следующей лекции мы изучим второй китовый инструмент управления потоком выполнения — **циклы**. Мы научимся повторять одни и те же действия многократно, что откроет перед нами поистине безграничные возможности.