Языки программирования

Лекция 1

Метод Main

- Точкой входа в программу на языке C# является метод Main. При создании проекта консольного приложения в Visual Studio, например, создается следующий метод Main:

```
class Program
{
    static void Main(string[] args)
    {
        // здесь помещаются выполняемые инструкции
    }
}
```

Инструкции

- Базовым строительным блоком программы являются **инструкции**.
- Инструкция представляет некоторое действие, например, арифметическую операцию, вызов метода, объявление переменной и присвоение ей значения.
- В конце каждой инструкции в C# ставится точка с запятой (;). Данный знак указывает компилятору на конец инструкции.

```
using System;

namespace HelloApp
{
    /*
        программа, которая спрашивает у пользователя имя
        и выводит его на консоль
    */
    class Program
    {
        // метод Main - стартовая точка приложения
        static void Main(string[] args)
        {
            Console.Write("Введите свое имя: ");
            string name = Console.ReadLine();       // вводим имя
            Console.WriteLine($"Привет {name}");    // выводим имя на консоль
            Console.ReadKey();
        }
    }
}
```

Переменные

- Для хранения данных в программе применяются **переменные**. Переменная представляет именованную область памяти, в которой хранится значение определенного типа. Переменная имеет тип, имя и значение. Тип определяет, какого рода информацию может хранить переменная.
- Перед использованием любую переменную надо определить. Синтаксис определения переменной выглядит следующим образом:

Типы данных

- Как и во многих языках программирования, в C# есть своя система типов данных, которая используется для создания переменных. Тип данных определяет внутреннее представление данных, множество значений, которые может принимать объект, а также допустимые действия, которые можно применять над объектом.
- В языке C# есть следующие примитивные типы данных:

- **bool**: хранит значение `true` или `false` (логические литералы). Представлен системным типом `System.Boolean`

```
1  bool alive = true;
2  bool isDead = false;
```

- **byte**: хранит целое число от `0` до `255` и занимает 1 байт. Представлен системным типом `System.Byte`

```
1  byte bit1 = 1;
2  byte bit2 = 102;
```

- **sbyte**: хранит целое число от `-128` до `127` и занимает 1 байт. Представлен системным типом `System.SByte`

```
1  sbyte bit1 = -101;
2  sbyte bit2 = 102;
```

Типы данных

- **short**: хранит целое число от `-32768` до `32767` и занимает 2 байта. Представлен системным типом `System.Int16`

```
short n1 = 1;
short n2 = 102;
```

- **ushort**: хранит целое число от `0` до `65535` и занимает 2 байта. Представлен системным типом `System.UInt16`

```
ushort n1 = 1;
ushort n2 = 102;
```

- **int**: хранит целое число от `-2147483648` до `2147483647` и занимает 4 байта. Представлен системным типом `System.Int32`. Все целочисленные литералы по умолчанию представляют значения типа int:

```
int a = 10;
int b = 0b101;  // бинарная форма b =5
int c = 0xFF;   // шестнадцатеричная форма c = 255
```

- **uint**: хранит целое число от `0` до `4294967295` и занимает 4 байта. Представлен системным типом `System.UInt32`

```
uint a = 10;
uint b = 0b101;
uint c = 0xFF;
```

- **long**: хранит целое число от `–9 223 372 036 854 775 808` до `9 223 372 036 854 775 807` и занимает 8 байт. Представлен системным типом `System.Int64`

```
long a = -10;
long b = 0b101;
long c = 0xFF;
```

Типы данных

- **ulong**: хранит целое число от `0` до `18 446 744 073 709 551 615` и занимает 8 байт. Представлен системным типом `System.UInt64`

```
ulong a = 10;
ulong b = 0b101;
ulong c = 0xFF;
```

- **float**: хранит число с плавающей точкой от $-3.4*10^{38}$ до $3.4*10^{38}$ и занимает 4 байта. Представлен системным типом `System.Single`
- **double**: хранит число с плавающей точкой от $\pm5.0*10^{-324}$ до $\pm1.7*10^{308}$ и занимает 8 байта. Представлен системным типом `System.Double`
- **decimal**: хранит десятичное дробное число. Если употребляется без десятичной запятой, имеет значение от $\pm1.0*10^{-28}$ до $\pm7.9228*10^{28}$, может хранить 28 знаков после запятой и занимает 16 байт. Представлен системным типом `System.Decimal`
- **char**: хранит одиночный символ в кодировке Unicode и занимает 2 байта. Представлен системным типом `System.Char`. Этому типу соответствуют символьные литералы:

```
char a = 'A';
char b = '\x5A';
char c = '\u0420';
```

- **string**: хранит набор символов Unicode. Представлен системным типом `System.String`. Этому типу соответствуют символьные литералы.

```
string hello = "Hello";
string word = "world";
```

- **object**: может хранить значение любого типа данных и занимает 4 байта на 32-разрядной платформе и 8 байт на 64-разрядной платформе. Представлен системным типом `System.Object`, который является базовым для всех других типов и классов .NET.

```
object a = 22;
object b = 3.14;
object c = "hello code";
```

Арифметические операции языка C#

- В C# используется большинство операций, которые применяются и в других языках программирования. Операции представляют определенные действия над операндами - участниками операции. В качестве операнда может выступать переменной или какое-либо значение (например, число). Операции бывают унарными (выполняются над одним операндом), бинарными - над двумя операндами и тернарными - выполняются над тремя операндами.

Бинарные арифметические операции

- +

Операция сложения двух чисел:

```
int x = 10;
int z = x + 12; // 22
```

- -

Операция вычитания двух чисел:

```
int x = 10;
int z = x - 6; // 4
```

- *

Операция умножения двух чисел:

```
int x = 10;
int z = x * 5; // 50
```

- /

операция деления двух чисел:

```
int x = 10;
int z = x / 5; // 2

double a = 10;
double b = 3;
double c = a / b; // 3.33333333
```

Унарные арифметические операции

- ++

  Операция инкремента

  Инкремент бывает префиксным: `++x` - сначала значение переменной x увеличивается на 1, а потом ее значение возвращается в качестве результата операции.

  И также существует постфиксный инкремент: `x++` - сначала значение переменной x возвращается в качестве результата операции, а затем к нему прибавляется 1.

```
int x1 = 5;
int z1 = ++x1; // z1=6; x1=6
Console.WriteLine($"{x1} - {z1}");

int x2 = 5;
int z2 = x2++; // z2=5; x2=6
Console.WriteLine($"{x2} - {z2}");
```

- --

  Операция декремента или уменьшения значения на единицу. Также существует префиксная форма декремента (`--x`) и постфиксная (`x--`).

```
int x1 = 5;
int z1 = --x1; // z1=4; x1=4
Console.WriteLine($"{x1} - {z1}");

int x2 = 5;
int z2 = x2--; // z2=5; x2=4
Console.WriteLine($"{x2} - {z2}");
```

Порядок выполнения арифметических операций

При выполнении сразу нескольких арифметических операций следует учитывать порядок их выполнения. Приоритет операций от наивысшего к низшему:

- Инкремент, декремент
- Умножение, деление, получение остатка
- Сложение, вычитание

Для изменения порядка следования операций применяются скобки.

Условные выражения

- **Операции сравнения**

  В операциях сравнения сравниваются два операнда и возвращается значение типа **bool** - **true**, если выражение верно, и **false**, если выражение неверно.

- ==

  Сравнивает два операнда на равенство. Если они равны, то операция возвращает **true**, если не равны, то возвращается **false**:

```
1  int a = 10;
2  int b = 4;
3  bool c = a == b; // false
```

- **!**=

  Сравнивает два операнда и возвращает true, если операнды не равны, и false, если они равны.

```
1  int a = 10;
2  int b = 4;
3  bool c = a != b;    // true
4  bool d = a!=10;     // false
```

Условные выражения

Операции сравнения

- **<**

Операция "меньше чем". Возвращает true, если первый операнд меньше второго, и false, если первый операнд больше второго:

```
int a = 10;
int b = 4;
bool c = a < b; // false
```

- **>**

Операция "больше чем". Сравнивает два операнда и возвращает true, если первый операнд больше второго, иначе возвращает false:

```
int a = 10;
int b = 4;
bool c = a > b;     // true
bool d = a > 25;    // false
```

- **<=**

Операция "меньше или равно". Сравнивает два операнда и возвращает true, если первый операнд меньше или равен второму. Иначе возвращает false.

```
int a = 10;
int b = 4;
bool c = a <= b;     // false
bool d = a <= 25;    // true
```

- **>=**

Операция "больше или равно". Сравнивает два операнда и возвращает true, если первый операнд больше или равен второму, иначе возвращается false:

```
int a = 10;
int b = 4;
bool c = a >= b;     // true
bool d = a >= 25;    // false
```

Логические операции

- **||**

  Операция логического сложения. Возвращает true, если хотя бы один из операндов возвращает true.

```
bool x1 = (5 > 6) || (4 < 6); // 5 > 6 - false, 4 < 6 - true, поэтому возвращается true
bool x2 = (5 > 6) || (4 > 6); // 5 > 6 - false, 4 > 6 - false, поэтому возвращается false
```

- **&&**

  Операция логического умножения. Возвращает true, если оба операнда одновременно равны true.

```
bool x1 = (5 > 6) && (4 < 6); // 5 > 6 - false, 4 < 6 - true, поэтому возвращается false
bool x2 = (5 < 6) && (4 < 6); // 5 < 6 - true, 4 < 6 - true, поэтому возвращается true
```

- **!**

  Операция логического отрицания. Производится над одним операндом и возвращает true, если операнд равен false. Если операнд равен true, то операция возвращает false:

```
bool a = true;
bool b = !a;    // false
```

Условные конструкции

- **Конструкция if/else**

Конструкция if/else проверяет истинность некоторого условия и в зависимости от результатов проверки выполняет определенный код.

- **Конструкция switch/case**

Конструкция switch/case аналогична конструкции if/else, также позволяет обработать сразу несколько условий.