

Язык программирования C#

C# Programming Language

Лекция 3. Тема. Операции

Тема 3. Операции

Содержание 3 лекции

1. Арифметические операторы (Arithmetic Operators) - +, -, *, /, %
2. Math.Pow(), Math.Sqrt() - математические функции
3. Операции сравнения и проверки на равенство. (<, <=, >, >=, ==, !=).
4. Операторы Инкремента и Декремента.
5. Правила преобразования типа результата арифметических операций
6. Операции присвоения с сложением, вычитанием, умножением, делением и присвоением остатка от деления.

Тема 3. Операции

Содержание 3 лекции (продолжение)

7. Использование локальных областей и локальных переменных
8. Символ @.
9. Операторы checked и unchecked.
10. Сцепление строк
11. Маркеры подстановки
12. Флаги форматирования строк
13. Оператор sizeof
14. Var - неявно типизированные локальные переменные
15. Сравнение значений разных типов

#1. Арифметические операторы (Arithmetic Operators) - +, -, *, /, %

К операторам, которые выполняют арифметические операции можно отнести операторы:

+ (сложения),

- (вычитания),

* (умножения),

/ (деления),

% (получения остатка от деления)

// Арифметические операторы

(Arithmetic Operators) - +, -, *, /, %

// Addition (+)

```
byte summand1 = 1, summand2 = 2; //
```

Множественное объявление.

```
int sum = 0;
```

```
sum = summand1 + summand2;
```

```
Console.WriteLine(sum);
```

// Subtraction (-)

```
byte minuend = 5, subtrahend = 3;  
int difference = 0;  
difference = minuend - subtrahend;  
  
Console.WriteLine(difference);
```

// Multiplication (*)

```
byte factor1 = 2, factor2 = 3;
```

```
int product = 0;
```

```
product = factor1 * factor2;
```

```
Console.WriteLine(product);
```

// Division (/)

```
byte dividend = 5, divisor = 2;  
int quotient = 0, remainder = 0;  
quotient = dividend / divisor;  
Console.WriteLine(quotient);
```

// Remainder after division (%)

```
remainder = dividend % divisor;  
Console.WriteLine(remainder);  
// Delay.  
Console.ReadKey();
```


- Операции умножения, деления, получения остатка от деления имеют больший приоритет, чем сложения и вычитания, поэтому выполняются в первую очередь.
- При получении результата остатка от деления - знак результата не сокращается и соответствует значению первого операнда (делимого).
- Если в правой части выражения выполнялись операции деления между целыми числами, то результат будет приведен компилятором к целому типу, даже если результат записать в переменную вещественного типа или привести все выражение к вещественному типу.

#2. `Math.Pow()` , `Math.Sqrt()` - математическая функции

Язык C# предоставляет большой набор математических функций для выполнения различных вычислений.

- `Math.Sqrt()` - математическая функция которая извлекает квадратный корень. В аргументных скобках указываем значение числа, из которого хотим извлечь квадратный корень.

- `Math.Pow()` - возведения числа в степень. В аргументных скобках через запятую указываем два аргумента (первый - число, которое хотим возвести в степень, второй – степень, в которую мы хотим

**// Math.Pow() - возведение числа в степень
(1-ый аргумент - число, которое возводим в
степень,
2-ой – степень, в которую возводим число)**

```
double x = 2, y = 8;  
double result = Math.Pow(x, y);
```

**// Math.Sqrt() - математическая функция, которая
извлекает квадратный корень**

```
double x = 256;  
double result = Math.Sqrt(x);
```

#3 Операции сравнения и проверки на равенство. (<, <=, >, >=, ==, !=).

К операциям сравнения можно отнести операции:

- > больше,
- >= больше или равно,
- < меньше,
- <= меньше или равно.

. К операциям проверки на равенство можно отнести операции:

- == равно,
- != не равно.

Результатом выполнения операций сравнения и проверки на равенство неравенство всегда будет либо false или

Для predetermined типов значений оператор равенства (==) возвращает значение true, если значения его операндов совпадают, в противном случае — значение false.

Для типа string оператор == сравнивает значения строк.

Оператор неравенства (!=) возвращает значение false, если его операнды равны, в противном случае — значение true.

Оператор сравнения "меньше или равно" (<=) возвращает значение true, если первый операнд меньше или равен второму, в противном случае возвращается значение false

// Операции сравнения и проверки на равенство

(<, <=, >, >=, ==, !=)

```
byte value1 = 0, value2 = 1;
```

```
bool result = false;
```

```
// Less than
```

```
result = value1 < value2;
```

```
Console.WriteLine(result);
```

```
// Greater than
```

```
result = value1 > value2;
```

```
Console.WriteLine(result);
```

```
// Less than or equal to
```

```
result = value1 <= value2;
```

```
Console.WriteLine(result);
```

**// Операции сравнения и проверки на равенство
(<, <=, >, >=, ==, !=)**

// Greater than or equal to

```
result = value1 >= value2;  
Console.WriteLine(result);
```

// Equals

```
result = value1 == value2;  
Console.WriteLine(result);
```

// Not equals

```
result = value1 != value2;  
Console.WriteLine(result);
```


#4. Операторы Инкремента и Декремента

Оператор инкремента (++) увеличивает свой операнд на 1. Оператор инкремента может находиться как перед операндом, так и после него: ++variable или variable++.

Префиксная операция увеличения - результатом выполнения этой операции является использование значения операнда после его увеличения.

Постфиксная операция увеличения - результатом выполнения этой операции является использование значения операнда перед его увеличением.

Оператор декремента (--) уменьшает свой операнд на 1. Оператор декремента может находиться как перед операндом, так и после него: --variable или variable--.

Префиксная операция декремента – результатом выполнения этой операции является использование значения операнда после его декремента.

Постфиксная операция декремента - результатом этой операции является использование значения операнда до его декремента.

// Операторы Инкремента и Декремента (Increment and Decrement Operators)

```
Console.WriteLine("----- Постфиксный инкремент"); // Post-increment
```

```
byte number1 = 0;
```

```
Console.WriteLine(number1++); // Сначала выводим на экран, потом  
увеличиваем на 1.
```

```
Console.WriteLine(number1);
```

```
Console.WriteLine("----- Префиксный инкремент"); // Pre-increment
```

```
byte number2 = 0;
```

```
Console.WriteLine(++number2); // Сначала увеличиваем на 1, потом  
выводим на экран.
```

```
Console.WriteLine("----- Постфиксный декремент"); // Post-decrement
```

```
    sbyte number3 = 0;
```

```
    Console.WriteLine(number3--); // Сначала выводим на экран,  
ПОТОМ уменьшаем на 1.
```

```
    Console.WriteLine(number3);
```

```
    Console.WriteLine("----- Префиксный декремент"); //
```

```
Pre-decrement
```

```
    sbyte number4 = 0;
```

```
    Console.WriteLine(--number4); // Сначала уменьшаем на 1, потом  
ВЫВОДИМ на экран.
```

#5. Правила преобразования типа результата арифметических операций

Все арифметические операции производимые над двумя значениями типа (byte, sbyte, short, ushort) в качестве результата, возвращают значение типа int.

Для типов int, uint, long и ulong, не происходит преобразования типа результата арифметических операций.

// ПРАВИЛО:

// Все арифметические операции производимые над двумя значениями типа (byte, sbyte, short, ushort) в качестве результата, возвращают значение типа int.

Assignment - присваивание

// Присвоение со сложением для типа byte.

byte variable1 = 0;

//variable1 = variable1 + 5; // ОШИБКА: Попытка неявного преобразования значения результата, тип int

//variable1 = (byte)variable1 + 5; // ОШИБКА: Происходит преобразование типа byte в тип byte, раньше выполнения операции сложения.

variable1 = (byte)(variable1 + 5); // Громоздкое решение.

variable1 += 5; // Элегантное решение.

//variable1 += 5000; // Ошибка. т.к. значение правой части выражения не должно превышать диапазон допустимых значений типа переменной

// ПРАВИЛО:

// Для типов int, uint, long и ulong, не происходит преобразования типа результата арифметических операций.

#6. Операции присвоения с сложением, вычитанием, умножением, делением и присвоением остатка от деления

// Присвоение со сложением.

```
int variable2 = 0;  
variable2 = variable2 + 5;  
variable2 += 5;
```

// Присвоение с вычитанием.

```
uint variable3 = 0;  
variable3 = variable3 - 5;  
variable3 -= 5;
```

// Присвоение с умножением.

```
long variable4 = 0;  
variable4 = variable4 * 5;  
variable4 *= 5;
```

// Присвоение с делением.

```
ulong variable5 = 0;  
variable5 = variable5 / 5;  
variable5 /= 5;
```

// Присвоение остатка от деления.

```
long variable6 = 0;  
variable6 = variable6 % 5;  
variable6 %= 5;
```

#7. Использование локальных областей и локальных переменных

Локальная область – участок кода, внутри класса или блок, который ограничен фигурными скобками.

Область видимости переменной - часть текста программы, в которой имя можно явно использовать. Чаще всего область видимости совпадает с областью действия.

Переменная созданная внутри локальной области называется локальной переменной, область ее действия - от открывающей скобки локальной области до ее окончания(закрывающей скобки) блока, включая все вложенные локальные области.

Переменная уровня класса называется глобальной переменной или полем.

В коде можно создавать локальные области и в двух разных локальных областях хранить одноименные переменные.

```
using System;
```

```
// Использование локальных областей и локальных переменных.
```

```
static void Main()
```

```
{
```

```
    // ПРАВИЛО: В коде можно создавать локальные области и в двух  
    разных локальных областях хранить одноименные переменные.
```

```
    // Локальная область 1
```

```
    {
```

```
        int a = 1;
```

```
        Console.WriteLine(a);
```

```
    }
```

```
    // Локальная область 2
```

```
    {
```

```
        int a = 2;
```

```
        Console.WriteLine(a);
```

```
    }
```

Если в коде имеются локальные области, то запрещается хранить одноименные переменные за пределами локальных областей. И наоборот, если за пределами локальных областей уже созданы переменные с каким-то именем, то в локальных областях этого уровня запрещается создавать одноименные переменные.

// ПРАВИЛО:

// Если в коде имеются локальные области, то запрещается хранить одноименные переменные за пределами локальных областей.

//int a = 3; // ОШИБКА: Переменная с таким именем уже существует в локальной области.

// Delay.

Console.ReadKey();

#8. Символ @.

Ключевые слова- это предварительно определенные зарезервированные идентификаторы, имеющие специальные значения для компилятора. Их нельзя использовать в программе в качестве идентификаторов, если только они не содержат префикс @.

Символ @, который используется в идентификаторе переменной, указывает компилятору, что это слово необходимо трактовать как идентификатор, а не как ключевое слово C# или его команду.

Символ @ не является частью идентификатора, поэтому, @myVariable - это тоже самое, что и myVariable.

// Использование ключевых слов языка C# в качестве идентификаторов.

```
static void Main()
```

```
{
```

```
    //int bool = 5;    // Illegal
```

```
    int @bool = 7;    // Legal
```

```
    Console.WriteLine(@bool);
```

// Символ @ не является частью идентификатора, поэтому, @myVariable - это тоже самое, что и myVariable.

```
    string @myVariable = "Hello";
```

```
    Console.WriteLine(myVariable);
```

```
    // Delay.
```

```
    Console.ReadKey();
```

```
}
```

#9. Операторы `checked` и `unchecked`

Операторы C# могут выполняться в проверяемом или непроверяемом контексте. В проверяемом контексте арифметическое переполнение вызовет исключение (ошибку).

В непроверяемом контексте арифметическое переполнение будет проигнорировано, а результат усечен.

Для таких действий используются следующие конструкции:

- `checked`- указание проверяемого контекста;
- `unchecked`- указание непроверяемого контекста;

Проверка переполнений применяется в следующих случаях:

- Если используются выражения, использующие predetermined операторы в целых типах с операциями
(++, --, +, -, *, /).
- Если выполняются явные числовые преобразования между целыми типами данных.

// Проверка переполнения - (checked)

sbyte a = 127;

// Проверять переполнение.

checked

{

 a++; // ОШИБКА уровня компилятора

}

// 127 + 1 = -128

// Запрет проверки переполнения - (unchecked)

```
sbyte a = 127;
    // Проверять переполнение.
    unchecked
    {
        a++; // Логическая ошибка
    }
    // 127 + 1 = -128
    Console.WriteLine(a);

    // Delay.
    Console.ReadKey();
```

// Комбинация проверки и запрета проверки переполнения.

```
sbyte a = 126;
```

```
    sbyte b = 127;
```

```
    // Проверять переполнение.
```

```
    checked
```

```
    {
```

```
        a++;
```

```
    // Не проверять переполнение.
```

```
    unchecked
```

```
    {
```

```
        b++;
```

```
    }
```

```
    a++;
```

```
}
```

#10. Сцепление строк

// Сцепление строк. (Конкатенация)

// 1 вариант.

```
string word1 = "Привет ";  
string word2 = "Мир!";  
string phrase = word1 + word2;  
Console.WriteLine(phrase);
```

// 2 вариант.

```
Console.WriteLine("Hello " + "World!");
```


Строки

Форматированный вывод

Маркер подстановки



```
Console.WriteLine("Это число {0}", 1);
```

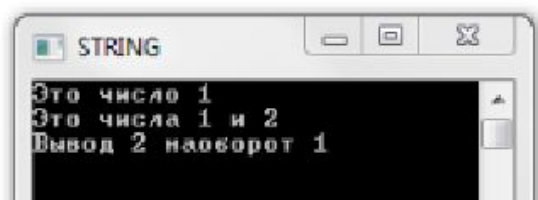
Элемент подстановки



Позиция элемента подстановки, начинается с нуля (0,1,2,3,... и т.д.), указание большей позиции приведет к ошибке.

```
Console.WriteLine("Это числа {0} и {1}", 1, 2);
```

```
Console.WriteLine("Вывод {1} наоборот {0}", 1, 2);
```



Позиция элемента подстановки, начинается с нуля (0,1,2,3,... и т.д.), указание большей позиции приведет к ошибке.

Конкатенация – сцепление строк или значений переменных типа `string`, для получения строк большего размера с помощью операции `+`.

Для форматирования числовых результатов и вывода их на экран можно использовать метод `Console.Write()` или `Console.WriteLine()`, который вызывает метод `string.Format()`.

#11. Маркеры подстановки

```
int a = 1;
```

```
    Console.WriteLine("Это число {0}", a);
```

```
int b = 2, c = 3;
```

```
    Console.WriteLine("Это числа {0} и {1}", b, c);
```

```
    Console.WriteLine("Это числа {0}, {1} и {2}", b, c);
```

```
// Delay.
```

```
    Console.ReadKey();
```

```
Это число 1
Это числа 2 и 3
Это числа наоборот 3 и 2
```

**//вывод на консоль в одной строке значения сразу
нескольких переменных.**

//прием, который называется интерполяцией:

```
string name = "Tom";
```

```
int age = 34;
```

```
double height = 1.7;
```

```
Console.WriteLine
```

```
    ($"Имя: {name} Возраст: {age} Рост: {height}м");
```

Имя: Tom Возраст: 34 Рост: 1,7м

```
// Сравните Console.WriteLine
```

```
string name = "Tom";
```

```
    int age = 34;
```

```
    double height = 1.7;
```

```
Console.WriteLine
```

```
("Имя: {0} Возраст: {2} Рост: {1}м", name, height,  
age);
```

```
Console.WriteLine
```

```
($"Имя: {name} Возраст: {age} Рост: {height}м");
```

#12. Флаги форматирования строк

Существуют следующие флаги форматирования строк:

- С или с - Валюта (Currency);
- D или d - Десятичное число (Decimal);
- E или e - Научный формат (Scientific, exponential)
- F или f - Формат с фиксированным значением после запятой (Fixed-point)
- G или g - Общие (General)
- N или n – Number (Number)
- X или x - Шестнадцатеричный формат (Hexadecimal)
- P или p –Процентный (Percent)

// Флаги форматирования строк.

```
Console.WriteLine("C format: {0:C}", 99.9);    // Вывод в денежном формате.  
Console.WriteLine("F format: {0:##}", 99.935); // Вывод значений с фиксированной точностью.  
Console.WriteLine("N format: {0:N}", 99999);  // Стандартное числовое форматирование.  
Console.WriteLine("X format: {0:X}", 255);    // Вывод в шеснадцатиричном формате.  
Console.WriteLine("D format: {0:D}", 0xFF);   // Вывод в десятичном формате.  
Console.WriteLine("E format: {0:E}", 9999);   // Вывод в экспоненциальном формате.  
Console.WriteLine("G format: {0:G}", 99.9);   // Вывод в общем формате.  
Console.WriteLine("P format: {0:P}", 99.9);   // Вывод в процентном формате.
```

```
C format: 99,90 ?  
F format: 100  
N format: 99 999,00  
X format: FF  
D format: 255  
E format: 9,999000E+003  
G format: 99,9  
P format: 9 990,00%
```

Формат задается с помощью флагов форматирования. Флаг форматирования может иметь следующую форму: `Axx`, где `A` — флаг формата (определяет тип формата), а `xx` — описатель точности (количество отображаемых цифр или десятичных знаков форматированного результата). Например: `Console.WriteLine("{0:F2}", 99.935);`

#13. Оператор sizeof

Оператор sizeof() - позволяет получить размер значения в байтах для указанного типа.

Оператор sizeof() можно применять только к типам: (byte, sbyte, short, ushort, int, uint, long, ulong, float, double, decimal, char, bool).

Возвращаемые оператором sizeof() значения имеют тип int.

// Оператор sizeof - позволяет получить размер значения в байтах для указанного типа.

// Оператор sizeof можно применять только к типам:

// byte, sbyte, short, ushort, int, uint, long, ulong, float, double, decimal, char, bool.

// Возвращаемые оператором sizeof значения имеют тип int.

```
int doubleSize = sizeof(double);
```

```
    Console.WriteLine("Размер типа double: {0} байт.",  
doubleSize);
```

```
    Console.WriteLine("Размер типа int: {0} байт.", sizeof(int));
```

```
    Console.WriteLine("Размер типа bool: {0} байт.", sizeof(bool));
```

```
    Console.WriteLine("Размер типа long: {0} байт.", sizeof(long));
```

```
    Console.WriteLine("Размер типа short: {0} байт.",  
sizeof(short));
```

Размер типа `double`: 8 байт.

Размер типа `int`: 4 байт.

Размер типа `bool`: 1 байт.

Размер типа `long`: 8 байт.

Размер типа `short`: 2 байт.

#14. Var - неявно типизированные локальные переменные

// Неявно типизированная локальная переменная.

```
var myVar = 7;
```

```
Console.WriteLine(myVar);
```

// Неявно типизированные локальные переменные не допускают множественного объявления.

```
// var a = 1, b = 2, c = 3;
```

// Неявно типизированные локальные переменные должны быть инициализированы.

```
// var a;
```

```
// Константа не может быть неявно типизированная.
```

```
// const var myVar = 3.14;
```

Неявный тип `var`. Такие переменные называют неявно типизированными локальными переменными. Таким способом можно «поручить» компилятору определить тип ваших переменных, если вы не знаете точно результат.

Правила использования неявно типизированных локальных переменных:

- Можно создать только в локальных областях;
- Должны быть проинициализированы непосредственно в месте создания;
- Не допускают множественного объявления;
- Константы не могут быть неявно типизированными

#15. Сравнение значений разных типов

// Сравнение значений разных типов.

```
bool result = false;
```

```
int a = 1;
```

```
float b = 2.0f;
```

```
result = a < b; // Сравнение значения типа int, со  
значением типа float - допустимо.
```

```
string c = "Hello";
```

```
//result = c < a; // Сравнение значения типа int, со  
значением типа string - не допустимо.
```

19. Оператор сравнения "меньше" (<) возвращает значение true, если первый операнд меньше второго, в противном случае возвращается значение false.

20. Оператор сравнения "больше" (>) возвращает значение true, если первый операнд больше второго, в противном случае возвращается значение false.

21. Оператор сравнения "больше или равно" (>=) возвращает значение true, если первый операнд больше или равен второму, в противном случае возвращается значение false.

Тема 3. Операции

Содержание отчета

(пишется от руки в лекционной тетради. Допускаются вклейки рисунков из презентации)

Примеры должны отличаться от приведенных в лекции!!

1. Арифметические операторы. (Arithmetic Operators) - +, -, *, /, %. Примеры.
2. Math.Pow() - возведение числа в степень. Пример.
3. Math.Sqrt() - математическая функция, которая извлекает квадратный корень. Пример.
4. Операции сравнения и проверки на равенство. (<, <=, >, >=, ==, !=). Примеры.
5. Операторы Инкремента и Декремента. (Increment and Decrement Operators). Префиксная и постфиксная формы. Примеры.
6. Отличие префиксной и постфиксной форм. Примеры.
7. Правило. Какое значение типа в качестве результата возвращают все арифметические операции, производимые над двумя значениями ти-па (byte, sbyte, short, ushort)?

8. Правило преобразования типа результата арифметических операций для типов `int`, `uint`, `long` и `ulong`. Примеры.
9. Операции присвоения с сложением, вычитанием, умножением, делением и присвоением остатка от деления. Примеры.
10. Использование локальных областей и локальных переменных. Правила. Примеры.
11. Символ `@`. Примеры использования.
12. Проверка переполнения. Оператор `checked`. Примеры использования.
13. Запрет проверки переполнения. Оператор `unchecked`. Примеры использования.
14. Сцепление строк. (Конкатенация). Примеры использования.
15. Маркеры подстановки. Примеры использования.
16. Флаги форматирования строк. Примеры использования.
17. Вывод на консоль в одной строке значения сразу нескольких переменных. Интерполяция. Пример.
17. Оператор `sizeof`. Примеры использования.
18. Сравнение значений разных типов. Примеры.
19. `Var` - неявно типизированные локальные переменные. Правила. Примеры.