**Урок № 5. C# Теория**

**СОЧЕТАНИЯ КЛАВИШ ДЛЯ ЗАПУСКА И ОТЛАДКИ ПРИЛОЖЕНИЙ**

**Ctrl + F5** запуск приложения без отладки  
**F5** запуск приложения с отладкой  
**Shift + F5** остановка отладки  
**F9** вставка / удаление точки останова  
**F10** шаг без захода в метод (во время отладки)  
**F11** шаг с заходом в метод (во время отладки)  
**Ctrl + Shift + F5** перезапуск отладки  
**Shift + F11** шаг с выходом из метода (во время отладки)  
**Ctrl + m + m** свернуть код метода, когда курсор находится в каком-то месте этого кода; чтобы развернуть его, используйте ту же комбинацию клавиш

**ОБЪЯВЛЕНИЕ МЕТОДОВ И ФУНКЦИЙ**

* Методы позволяют нам инкапсулировать поведение и функциональность в объекты, которые мы создаем в нашем коде.
* Метод объявляется с использованием сигнатуры.

Сигнатура метода состоит из:  
    
**Модификатор доступа** — откуда этот метод может быть вызван. Некоторые из них:

* *private* — наиболее ограничительный, допускающий доступ к методу только из содержащего его класса или структуры
* *public*— доступ из любого фрагмента кода в рамках приложения
* *protected*— позволяет получить доступ из содержащего класса или из производных классов
* *internal*— доступ из файлов внутри одной сборки
* *static*— указывает, что метод является статическим членом класса, а не членом экземпляра конкретного объекта
* **Тип возвращаемого значения** — используется для указания типа возвращаемого значения. Используйте *void*, если метод не возвращает значение
* **Идентификатор (имя) метода** — все методы должны иметь идентификатор (имя), чтобы была возможность вызывать метод в коде. Правила идентификаторов применяются и к именам методов
* **Список аргументов (параметров)** — разделенный запятыми список аргументов, передаваемых в метод

Пример функции (метода):

|  |
| --- |
| **public** Boolean StartService(**string** serviceName)  {  *// код функции*  } |

В примере:  
public модификатор доступа,  
Boolean тип возвращаемого значения,  
StartService идентификатор (имя),  
string serviceName параметр (аргумент).

* Чтобы вернуть значение из функции, используйте оператор:

|  |
| --- |
| **return** выражение; |

* Если вместо возвращаемого типа у нас присутствует ключевое слово void, то это означает, что функция не вернет никакого значения.

|  |
| --- |
| **static** **void** SayHello()  {  Console.WriteLine("Hello");  } |

* Можно определить функцию в том же классе, что и функцию Main(), или определить ее в другом классе.
* Все методы или функции в си шарп должны быть статическими, что означает, что их можно вызвать без создания класса объекта.
* Чтобы методы (функции), описываемые в одном классе, сделать видимыми в другом, они должны быть описаны с помощью модификатора доступа public
* Если метод (функция) вызывается в том же классе, модификатор доступа public не требуется.

**ВЫЗОВ МЕТОДА / ФУНКЦИИ**

 Чтобы вызвать метод в c sharp, необходимо указать имя метода и указать в скобках все аргументы, соответствующие параметрам метода.

|  |
| --- |
| **static** **void** Main(**string**[] args)  {  **int** res;  Square(5, **out** res);  Console.WriteLine(res);  }  **static** **void** Square(**int** a, **out** **int** res)  {  res = a \* a;  } |

В этом примере:  
a — *ref* аргумент (входно-выходной);  
res — *out* аргумент (выходной).

**ВОЗВРАТ ЗНАЧЕНИЯ ИЗ ФУНКЦИИ / МЕТОДА**

Три подхода для возврата значений в си шарп:

 Возврат массива или коллекции

 Использование ключевого слова ref (входно-выходной параметр)

 Использование ключевого слова out (выходной параметр)

 Итак, параметры, передаваемые по ссылке, делятся на out и ref — параметры. Out – выходные, ref – входно-выходные.

 ref-параметры перед вызовом функции обязательно должны быть инициализированы, иначе – ошибка компилятора. При вызове функции также указываются ключевые слова out и ref

 Посмотрите на следующие примеры: первый пример — без ref, второй — с ref:

|  |
| --- |
| *// 1. без ref*  **static** **void** Main(**string**[] args)  {  **int** a = 1;  Plas2(a);  Console.WriteLine(a); *// результат 1*  }  **static** **void** Plas2(**int** a)  {  a += 2;  } |
| *// 2. с ref*  **static** **void** Main(**string**[] args)  {  **int** a = 1;  Plas2(**ref** a);  Console.WriteLine(a); *// результат 3*  }  **static** **void** Plas2(**ref** **int** a)  {  a += 2;  } |

 Для параметров используется ключевое слово out, чтобы указать, что значения тих параметров будут возвращены. При вызове также указывается ключевое слово out:

|  |
| --- |
| **static** **void** Main(**string**[] args)  {  **int** a; *// здесь не нужно инициализировать будущий параметр значением*  Plas2(**out** a);  Console.WriteLine(a); *// 3*  }  **static** **void** Plas2(**out** **int** a)  {  a = 1; *// здесь необходимо присвоить какое-то значение*  a += 2;  } |

 Еще один пример использования модификатора out:

|  |
| --- |
| **static** **void** Main(**string**[] args)  {  **int** first;  **String** sValue;  ReturnMultiOut(**out** first, **out** sValue);  Console.WriteLine($"{first.ToString()}, {sValue}");  }  **static** **void** ReturnMultiOut(**out** **int** i, **out** **string** s)  {  i = 25;  s = "использование out";  } |

Результат будет:

25 использование out

 В следующем примере ключевое слово ref используется для возврата значений из метода. При использовании ref модификатор необходимо, чтобы сначала данные аргументы были инициализированы значениями

|  |
| --- |
| **static** **void** Main(**string**[] args)  {  *// сначала инициализируем значением, так как ref*  **int** first =0;  **String** sValue="";  ReturnMultiOut(**ref** first, **ref** sValue);  Console.WriteLine($"{first.ToString()}, {sValue}");    }  **static** **void** ReturnMultiOut(**ref** **int** i, **ref** **string** s)  {  i = 25;  s = "использование ref";  } |

**ПЕРЕГРУЗКА МЕТОДОВ / ФУНКЦИЙ**

 Перегрузка обозначает наличие метода с тем же именем, но разными сигнатурами.

**Задания и лабораторные си шарп**

**Лабораторная работа 1. Функция, суммирующая два аргумента**

**Выполнить:** Создайте метод Sum(), который принимает два целочисленных аргумента и суммирует их. Метод не возвращает никакого значения (именно поэтому вы должны использовать ключевое слово void).

**Указание:** Необходимо использовать ключевое слово static в сигнатуре метода, потому что основная функция статична, и мы не можем вызвать нестатический метод из статического метода.

**Пример выполнения:**

Введите два числа

20 40

Сумма 20 + 40 = 60

[Название проекта: Lesson\_5Lab1, название файла L5Lab1.cs]

 Алгоритм:

* Запустите Visual Studio.
* Создайте консольное приложение, назовите проект Lesson\_5Lab1.
* В окне Solution Explorer (Обозреватель решений) найдите файл Program.cs и переименуйте его в L5Lab1.cs.
* В главной функции запросите пользователя ввести два числа:

...

Console.WriteLine("Введите два числа");

int a = int.Parse(Console.ReadLine());

int b = int.Parse(Console.ReadLine());

...

* Поместите курсор после закрывающей фигурной скобки функции Main() и нажмите клавишу *enter*. Мы делаем это потому, что нельзя помещать методы/функции внутри другой функции.
* Объявите новый метод Sum(), который будет использоваться для суммы переданных в него значений:

...

static void Sum(int first, int second)

{

int sum = first + second;

Console.WriteLine($"Сумма {first} + {second} = {sum}");

}

...

Метод не возвращает никакого значения основной программе, поэтому мы должны использовать ключевое слово void.

* Теперь мы можем вызвать этот метод из основной функции. Введите следующий код в фигурные скобки Main():

|  |
| --- |
| **static** **void** Main(**string**[] args)  {  ...  Sum(a, b);  } |

* Щелкните CTRL+F5, чтобы запустить приложение без отладки.
* Созданная нами функция Sum() не возвращает значения. Теперь мы изменим этот метод так, чтобы он возвращал результат вызывающему методу (основному методу, откуда он был вызван).
* Закоментируйте код функции Sum(), используя горячие клавиши [CTRL]+k+c:

|  |
| --- |
| *//static void Sum(int first, int second)*  *//{*  *// int sum = first + second;*  *// Console.WriteLine($"The sum of {first} and {second} is: {sum}");*  *//}* |

* Установите курсор после комментариев и введите следующий код:

...

static int Sum(int first, int second)

{

int sum = first + second;

return sum;

}

...

Метод возвращает целочисленное значение, именно поэтому мы используем int в сигнатуре (static **int** Sum(...)).  
Обратите внимание, что имена параметров, которые мы указали здесь в сигнатуре метода, могут не совпадать с именами аргументов, которые мы передали. Эти параметры становятся локальными переменными в рамках этого метода.

* Затем нам нужно изменить способ вызова метода. Объявите целочисленную переменную (в функции Main()), чтобы получить возвращаемое значение. Выведите результат в окно консоли

...

static void Main(string[] args)

{

int result = Sum(a, b);

Console.WriteLine($"Сумма {a} + {b} = {result}");

}

...

* Запустите приложение. Результаты должны быть одинаковыми.

Перегрузка метода / функции:

Перегрузка метода означает наличие метода с тем же именем, но разными сигнатурами (обычно с разным количеством параметров).

* Давайте теперь перегрузим наш метод Sum(). Для этого мы создадим два дополнительных метода с одинаковыми названиями.
* Поместите курсор после метода Sum(). Сначала создайте метод, который принимает три целых числа:

static int Sum(int first, int second, int third)

{

int sum = first + second + third;

return sum;

}

* Этот метод использует то же имя, что и предыдущий метод Sum(), который принимает два целых числа, но параметры здесь указывают, что метод ожидает три целых числа в качестве аргументов. Компилятор «узнает», какой метод следует вызвать, основываясь на количестве переданных аргументов.
* Затем введите следующий код для создания метода Sum(), принимающего два вещественных аргумента (типа double):

...

static double Sum(double first, double second)

{

double result = first + second;

return result;

}

...

* Наконец, измените код в Mai (), который вызывает данные методы:

static void Main(string[] args)

{

...

int result = Sum(a, b);

Console.WriteLine($"Вызов Sum() с двумя аргументами: {result}");

int result3 = Sum(10, 50, 80);

Console.WriteLine($"Вызов Sum() с тремя аргументами: {result3}");

double dblResult = Sum(20.5, 30.6);

Console.WriteLine($"Вызовg Sum() с вещественными аргументами: {dblResult}");

}

* Запустите приложение еще раз и проверьте выходные данные. Вы должны увидеть правильные результаты суммы для всех трех различных вызовов. Несмотря на то, что у всех них идентификатор Sum, компилятор выбирает правильный метод для вызова на основе сигнатуры метода. Именно так работает перегрузка метода.

**Задание 1:**

**Выполнить:** Вводятся три числа — длины трех сторон треугольника. Создайте функцию Perimeter(), которая вычисляет периметр треугольника по длинам трех его сторон.  
    
**Указание 1:** Метод Perimeter() должен принимать в качестве аргументов три целых числа.  
**Указание 2:** Метод не должен возвращать никакого значения, поэтому вы должны использовать ключевое слово void в сигнатуре:

|  |
| --- |
| **static** **void** Perimeter(...); |

**Указание 3:** Не забудьте преобразовать введенные значения в целые числа. Например:

|  |
| --- |
| **int** a = **int**.Parse(Console.ReadLine()); |

**Пример выполнения:**

Введите значения для трех сторон треугольника:

3 5 6

Периметр: 14

[Название проекта: Lesson\_5Task1, название файла L5Task1.cs]

**Задание 2:**

**Выполнить:** Необходимо переделать предыдущую задачу, теперь функция периметр должна возвращать целое значение. Напомним задание: Вводятся три числа — длины трех сторон треугольника. Создайте функцию Perimeter(), которая вычисляет периметр треугольника по длинам трех его сторон.  
    
**Указание 1:** Метод Perimeter() должен принимать в качестве аргументов три целых числа.  
**Указание 2:** Метод должен возвращать значение типа **integer**, поэтому вы должны// использовать ключевое слово int в сигнатуре:

|  |
| --- |
| **static** **int** Perimeter(...); |

**Указание 3:** Не забудьте преобразовать введенные значения в целые числа. Например:

|  |
| --- |
| **int** a = **int**.Parse(Console.ReadLine()); |

    
**Пример выполнения:**

Введите значения для трех сторон треугольника:

3 5 6

Периметр: 14

[Название проекта: Lesson\_5Task2, название файла L5Task2.cs]

**Лабораторная работа 2. Построение экспоненциальной функции**

**Выполнить:** Создайте метод GetPow(), который принимает два целочисленных аргумента — число и степень. Метод возвращает результат возведения числа в степень.

**Указание 1:** Необходимо использовать служебное слово static, потому что основная функция static, а мы не можем вызвать нестатический метод из статического метода.

**Указание 2:** Функция возвращает целочисленное значение, поэтому необходимо использовать int в сигнатуре функции:

|  |
| --- |
| **static** **int** GetPow(**int** baseNum, **int** powNum) {…} |

**Пример выполнения:**

Введите два числа – число и степень:

2 4

число 2 возведенное в степь 4 = 16

[Название проекта: Lesson\_5Lab2, название файла L5Lab2.cs]

 Алгоритм:

* Запустите Visual Studio.
* Создайте консольное приложение, назовите проект Lesson\_5Lab2.
* После закрывающей скобки функции Main создайте функцию GetPow:

static **int** GetPow(**int** baseNum, **int** powNum) {…}

Функция принимает два целых числа — аргумент baseNum и аргумент powNum. Внутри функции необходимо возвести baseNum в степень powNum.

* Объявите переменную result для хранения результата и возврата этого значения из функции.

**int** result = 1;

…

return result;

* Внутри функции создайте цикл, в котором накапливается произведение числа. Цикл должен выполнится powNum раз:

for (**int** i=0; i<powNum; i++)

{

result = result \* baseNum;

}

В первую итерацию цикла всегда будет выполняться 1 \* baseNum, результат сохраняется в переменной result. За две итерации имеем baseNum \* baseNum. За три – baseNum \* baseNum \* baseNum и т. д. Мы повторяем это действие каждую итерацию.

* Вызовите созданный метод внутри функции Main:

Console.WriteLine("Число {baseNum} возведенное в степень {powNumNum} = {GetPow(2,4)} ");

* Запустите приложение и посмотрите на результат.
* Теперь вместо конкретных чисел (*2* и *4*) запросите эти числа у пользователя. Затем измените код вызова функции:

Console.WriteLine ("Введите два числа – число и степень числа: ");

**int** a = Int32.Parse(Console.ReadLine());

**int** b = Int32.Parse(Console.ReadLine());

Console.WriteLine(GetPow(a,b));

* Запустите приложение и протестируйте результат с разными значениями.
* Сохраните проект.

**Задание 3:**

**Выполнить:** Создайте метод Distance который вычисляет расстояние между двумя точками на плоскости. Координаты точек вводятся (переменные x1,y1 — для первой точки, и x2,y2 — для второй).  
    
**Указание 1:** Метод Distance() принимает 4 аргумента (координаты точек).  
**Указание 2:** Метод не должен возвращать значение, поэтому следует использовать ключевой слово void в сигнатуре:

|  |
| --- |
| **static** **void** Distance(...); |

**Указание 3:** Для подсчета расстояния между двумя точками на плоскости используйте формулу:

https://labs-org.ru/wp-content/uploads/765.png

|  |
| --- |
| квадратный корень: Math.Sqrt(...);  степень числа: Math.Pow(number, power); |

**Пример выполнения:**

Введите координаты двух точек (четыре целых числа: x1, y1, x2, y2):

1 -2 4 2

Расстояние: 5

[Название проекта: Lesson\_5Task3, название файла L5Task3.cs]

**Задание 4:**

**Выполнить:** Необходимо переделать предыдущее задание. Создайте метод Distance который вычисляет расстояние между двумя точками на плоскости. Координаты точек вводятся (переменные x1,y1 — для первой точки, и x2,y2 — для второй).  
    
**Указание:** Метод должен возвращать вещественное значение (**double**):

|  |
| --- |
| **static** **double** Distance(...); |

**Пример выполнения:**

Введите координаты двух точек (четыре целых числа: x1, y1, x2, y2):

3.2 3.4 8 7.1

Расстояние: 6.0605

[Название проекта: Lesson\_5Task4, название файла L5Task4.cs]

**Лабораторная работа 3. *Ref* аргументы (аргументы ввода-вывода)**

**Выполнить:** Создайте метод Minmax() который принимает два целочисленных аргумента по ссылке (ref) и меняет их значения таким образом, что первый становится максимальным, а второй — минимальным (меняет значения аргументов, если требуется). Создайте перегруженный метод Minmax для трех параметров.

**Указание 1:** Необходимо использовать ключевое слово static в сигнатуре создаваемого метода, т.к. функция Main тоже static, невозможно вызвать не static метод из static метода.

**Указание 2:** Метод не возвращает значения значит, используется ключевое слово void в сигнатуре).

**Пример выполнения:**

Введите два числа

2 4

Результат после метода Minmax : 4 2

Введите три числа

2 4 3

Результат после перегрузки метода Minmax : 4 3 2

[Название проекта: Lesson\_5Lab3, название файла L5Lab3.cs]

 Алгоритм:

* Создайте консольное приложение, назовите проект Lesson\_5Lab3.
* В окне Solution Explorer (Обозреватель решений) найдите файл Program.cs и переименуйте его в L5Lab3.cs.
* В теле функции Main попросите пользователя ввести два числа. Присвойте введенные значения переменным:

...

Console.WriteLine("Введите два числа");

int a = int.Parse(Console.ReadLine());

int b = int.Parse(Console.ReadLine());

...

* Затем, то же самое сделайте для перегруженной функции с тремя аргументами. Попросите пользователя ввести три числа и присвойте введенные значения переменным:

...

Console.WriteLine("Введите три числа");

a = int.Parse(Console.ReadLine());

b = int.Parse(Console.ReadLine());

int c = int.Parse(Console.ReadLine());

...

* Для добавления метода установите курсор после тела функции Main().
* Объявите методMinmax(), который будет менять значения переменных, если это потребуется. Используйте модификатор ref для аргументов:

...

static void Minmax(ref int a, ref int b)

{

...

}

...

Модификатор ref используется для того, чтобы значение аргумента передалось из основной программы в создаваемую функцию и, наоборот, из функции измененное значение должно возвратиться в основную программу. *Ref* аргументы должны быть инициализированы до их вызова, иначе возникнет ошибка. При вызове модификатор ref также указывается.

* Найдите максимальное и минимальное значения, используя стандартные функции Min() и Max(). Присвойте найденные значения соответствующим переменным (первый аргумент должен быть больше второго):

static void Minmax(ref int a, ref int b)

{

int max = Math.Max(a, b);

int min = Math.Min(a, b);

a = max;

b = min;

...

* Поскольку созданный метод Minmax не возвращает никакого значения, то новые значения переменных будут передаваться в основную программу, благодаря модификатору ref.
* Теперь можно вызвать созданный метод из функции Main:

...

Minmax(ref a,ref b);

Console.WriteLine($"Результат после метода Minmax: {a}, {b}");

...

* Запустите приложение и проверьте результат.
* Чтобы свернуть или развернуть код какого-либо метода, можно использовать горячие клавиши Ctrl+M (дважды).

Перегруженный метод (функция):

* Установите курсор после метода Minmax() и создайте перегруженный метод (с таким же именем) для трех аргументов:

...

static void Minmax(ref int a, ref int b, ref int c)

{

Minmax(ref a, ref b);

Minmax(ref b, ref c);

Minmax(ref a, ref b);

}

...

В коде метода найдем сначала минимальное и максимальное из переменных a и b, затем из b и c, и после — из a и b еще раз, так как значения переменных могли уже поменяться к этому моменту.

* Наконец добавьте вызов метода из функции Main():

...

Minmax(ref a,ref b, ref c);

Console.WriteLine($"After overloaded Minmax the result is: {a}, {b}, {c}");

...

* Запустите приложение и проверьте результат. Несмотря на то, что имена у созданных методов одинаковы, компилятор вызовет нужный метод, исходя из количества аргументов.

**Задание 5:**

**Выполнить:** Вводится двузначное целое число. Создайте метод ChangeDigits(), который принимает введенный аргумент по ссылке и изменяет его значение так, чтобы первая цифра нового числа была второй цифрой введенного числа и, наоборот, вторая цифра нового числа была первой цифрой введенного числа. Например, если ввести **45**, то результирующее число будет равно **54**.  
    
**Указание 1:** Метод ChangeDigits() принимает аргумент с модификатором *ref*.  
**Указание 2:** Метод не возвращает значения:

|  |
| --- |
| **static** **void** ChangeDigits(...); |

**Указание 3:** Сначала необходимо получить цифры из числа. Затем, поменяв местами цифры, сделать новое число. Пример создания нового числа из двух цифр:

если даны цифры 2, 3

2\*10 + 3 = 23

**Пример выполнения:**

Введите двузначное число:

58

Результат: 85

[Название проекта: Lesson\_5Task5, название файла L5Task5.cs]

**Задание 6:**

**Выполнить:** Вводятся два двузначных целых числа. Создайте функцию BitwiseSum(), которая вычисляет их побитовую сумму по модулю 10. Например, побитовая сумма чисел 34 и 59 является числом 83 (3 + 5 = 8; 4 + 9 = 13, 13%10 = 3).  
    
**Указание 1:** Функция BitwiseSum() имеет два аргумента.  
**Указание 2:**Функция возвращает целочисленное значение int:

|  |
| --- |
| **static** **int** BitwiseSum(...); |

**Указание 3:** Сначала необходимо получить цифры из числа. Затем, поменяв местами цифры, сделать новое число. Пример создания нового числа из двух цифр:

если даны цифры 2, 3

2\*10 + 3 = 23

<="" strong="" style="box-sizing: border-box;">

Введите двузначное число:

34 59

Побитовая сумма чисел 34 и 59 = 83

[Название проекта: Lesson\_5Task6, название файла L5Task6.cs]

**Lab 4. Параметр *Ref* (параметры ввода-вывода)**

**Выполнить:** Создайте функцию MinmaxSeq(), которая принимает два аргумента по ссылке (ref аргументы) — минимальное и максимальное значение вводимой последовательности чисел. Запросите у пользователя ввести последовательность, которая должна закончиться 0 (если вводится 0, то ввод завершается). Функция должна выполнять поиск максимального и минимального значения последовательности и возвращать результат.

**Указание 1:** Функция не должна возвращать значениен (поэтому в объявлении функции используется void). Параметры ref — это параметры *ввода-вывода*, это говорит о том, что значения этих параметров меняются внутри функции и возвращаются в основную программу с вызовом фнукции.  
**Указание 2:** Необходимо использовать ключевое слово static в объявлении метиода, так как функция Main тоже статическая, а нестатическая фнукция не может быть вызвана внутри статической.

[Название проекта: Lesson\_5Lab4, название файла L5Lab4.cs]

**Пример вывода:**

Введите полседовательность и закончите ввод нулем:

2 4 8 3 0

минимальное: 2, максимальное: 8

 Алгоритм:

* Создайте *Консольное приложение* и назовите его Lesson\_5Lab4.
* В окне *Обозреватель решений* найдите файл Program.cs и переименуйте его в L5Lab4.cs.
* В функции Main объявите две переменные для хранения минимального и максимального значения последовательности. Присвойте им начальные значения: для переменной min — наибольшее возможное целое число, для переменной max — наименьшее возможное целое число:

...

int max = int.MinValue;

int min = int.MaxValue;

...

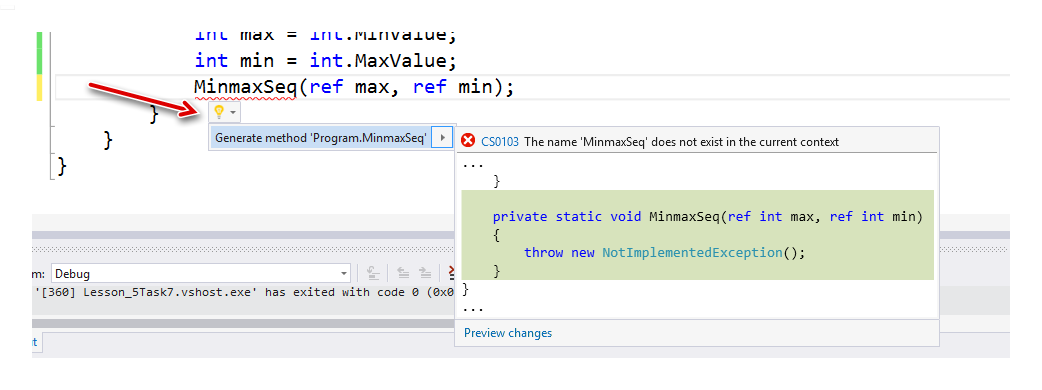
* Функцию, которую мыходит создать, будет называться MinmaxSeq(), и она будет принимать два аргумента по ссылке — переменные min и max. Сначала давайте вызовем пока еще несозданный метод из функции Main, сразу после инициализации переменных min и max:

int max = int.MinValue;

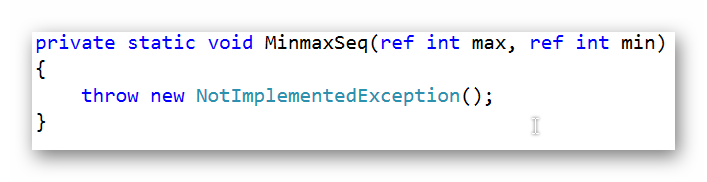
int min = int.MaxValue;

MinmaxSeq(ref max, ref min); // новый код

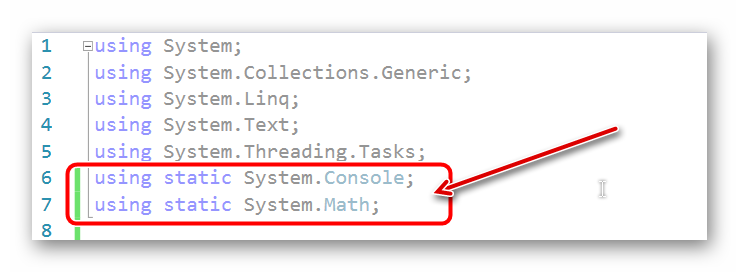
* C# может «самостоятельно» добавлять объявления фнукций. Давайте попросим сделать это. Наведите курсор мыши на вызов функции и щелкните стрелку рядом с появившейся лампочкой:



* Щелкните по фразе *«Сгенерировать метод…»*. Псоле этого у нас готово объявление функции:



* Удалите строку внутри функции.
* Теперь необходимо добавить код функции. Но сначала попросим пользователя ввести числа последовательности.
* Чтобы постоянно не печатать класс Console и Math, подключите их в виде директив в самом начале кода:



* Теперь нет необходимости полностью печатать Console.ReadLine() или Console.WriteLine(...), достаточно напечатать ReadLine() или WriteLine(...).
* Попросителя пользователя ввести последовательность:

...

private static void MinmaxSeq(ref int max, ref int min)

{

WriteLine("Введите последовательность и закончите ввод нулем");

}

...

* Чтобы ввести числа последовательности необходимо использовать цикл [do..while](https://labs-org.ru/c-sharp4-eng/" \l "Dowhile_Loop" \t "_blank):

int a;

do

{

a = Int32.Parse(ReadLine());

}

while (a != 0);

* Внутри цикла необходимо проверять элементы последовательности на максимум и минимум.

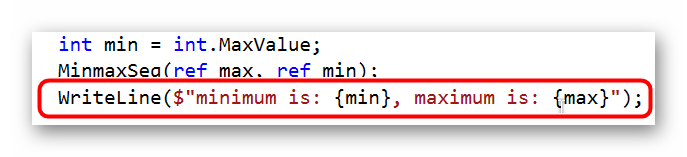
...

if (a > max && a != 0) { max = a; }

if (a < min && a != 0) { min = a; }

...

* Теперь осталось вывести min и max значения в функции Main:



* Запустите приложение и проверьте результат.

**Task 7:**

**Выполнить:** Создайте функцию PosNegSeq(), которая принимает два аргумента *по ссылке* — счетчики положительных чисел последовательности и отрицательных чисел. Пользователь вводит последовательность целых чисел, завершая ввод 0. Функция должна посчитать количество отрицательных и положительных чисел и вернуть значения счетчиков в главную программу.

**Указание 1:** Функция не должна возвращать значение (поэтому используется ключевое слово void в объявлении функции). Параметр ref — это параметр *ввода-вывода*, что означает, что значения параметров будут изменены внутри функции и возвращены в основную программу.  
**Указание 2:** Параметры *ref* должны быть инициализированы до вызова функции. Поэтому сначала вам необхзодимо инициализировать такие параметры значениями. Установите начальные значения для этих параметров, равные 0:

|  |
| --- |
| ...  **int** counterPositive = 0;  **int** counterNegative = 0;  ... |

    
[Название проекта: Lesson\_5Task7, название файла L5Task7.cs]  
    
**Пример вывода:**

Введите последовательность, завершите ввод 0

2 -4 8 -3 5 0

кол-во отрицательных: 2, количество положительных: 3