**ОБЪЯВЛЕНИЕ процедур И ФУНКЦИЙ**

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

Сигнатура функции состоит из:

**Модификатор доступа** — откуда этот функция (процедура) может быть вызван. Некоторые из них:

* *private* — наиболее ограничительный, допускающий доступ к функии 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 функция является статическим членом класса, а не членом экземпляра конкретного объекта
* **Тип возвращаемого значения** — используется для указания типа возвращаемого значения. Используйте *void*, если функция не возвращает значение (процедура)
* **Идентификатор (имя) функции** — все функции должны иметь идентификатор (имя), чтобы была возможность вызывать функцию в коде. Правила идентификаторов применяются и к именам функций
* **Список аргументов (параметров)** — разделенный запятыми список аргументов, передаваемых в функцию

Пример функции:

|  |
| --- |
| **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 функции в си # должны быть статическими, что означает, что их можно вызвать без создания класса объекта.
* Чтобы функции, описываемые в одном классе, сделать видимыми в другом, они должны быть описаны с помощью модификатора доступа public
* Если функция вызывается в том же классе, модификатор доступа public не требуется.

**ВЫЗОВ ФУНКЦИИ**

 Чтобы вызвать функцию в c#, необходимо указать имя функции и указать в скобках все аргументы, соответствующие параметрам функции.

|  |
| --- |
| **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**

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 refю

 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 функции. 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"; } |

 Создайте функцию Sum(), который принимает два целочисленных аргумента и суммирует их. Функция не возвращает никакого значения (именно поэтому вы должны использовать ключевое слово void).

**замечание:** Необходимо использовать ключевое слово static в сигнатуре функции, потому что основная функция статична, и мы не можем вызвать нестатическая функцияиз статической фунции.

**Пример выполнения:**

Введите два числа

20 40

Сумма 20 + 40 = 60

* Запустите Visual Studio.
* Создайте консольное приложение, назовите проект Lesson\_XLab1.
* В окне Solution Explorer (Обозреватель решений) найдите файл Program.cs и переименуйте его в LXLab1.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*..
* Объявите новe. Aeyrwb. 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}");

}

...

Функция не возвращает никакого значения основной программе, поэтому мы можем использовать ключевое слово void.

* Теперь мы можем вызвать эту функцию 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 ту функцию так, чтобы она возвращала результат вызывающей функции (основной функцииу, откуда она была вызвана).
* 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;

 }

...

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

* Затем нам нужно изменить способ вызова функции. 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 функции:

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

* Давайте теперь перегрузим нашу функцию Sum(). Для этого мы создадим две дополнительных функции с одинаковыми названиями.
* Поместите курсор после функции Sum(). Сначала создайте функцию, которая принимает три целых числа:

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

{

 int sum = first + second + third;

 return sum;

}

* Эта функция использует то же имя, что и предыдущая функция Sum(), которая принимает два целых числа, но параметры здесь указывают, что функция ожидает три целых числа в качестве аргументов. Компилятор «узнает», какую функцию следует вызвать, основываясь на количестве переданных аргументов.
* Затем введите следующий код для создания функции Sum(), принимающего два вещественных аргумента (типа double):

...

static double Sum(double first, double second)

{

 double result = first + second;

 return result;

}

...

* Наконец, измените код в Main (), который вызывает данные функции:

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 правильную функцию для вызова на основе сигнатуры функции. Именно так работает перегрузка функции.

**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 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

**Задача 2.** Напишите функцию, которая будет менять в массиве целых чисел все элементы, которые равны указанному значению (аргумент) на противоположное значение по знаку. Например, все элементы массива которые равны 5, будут меняться на -5.

**Задача 3.** Напишите функцию, которая будет находить минимальное число из трех.

**Задача 4.** Напишите функцию, которая будет возвращать указанный элемент ряда Фибоначчи. Ряд Фибоначчи – это ряд, в котором каждый следующий элемент равен сумме двух предыдущих. 1 1 2 3 5 8 13 21… Функция принимает порядковый номер элемента, и возвращает соответствующий элемент.