Как пояснялось выше, аргументы простых типов, например ***int*** или ***char***, передаются

методу по значению. Это означает, что изменения, вносимые в параметр, принимающий

значение, не будут оказывать никакого влияния на аргумент, используемый

для вызова. Но такое поведение можно изменить, используя ключевые слова ***ref*** и ***out***

для передачи значений обычных типов по ссылке. Это позволяет изменить в самом

методе аргумент, указываемый при его вызове.

Нередко требуется, чтобы метод оперировал теми аргументами, которые ему передаются.

Характерным тому примером служит метод Swap(), осуществляющий перестановку

значений своих аргументов. Но поскольку аргументы простых типов передаются по значению, то, используя выбираемый в C# по умолчанию механизм вызова

по значению для передачи аргумента параметру, невозможно написать метод, меняющий

местами значения двух его аргументов, например типа int. Это затруднение разрешает

модификатор ***ref***.

Как вам должно быть уже известно, значение возвращается из метода вызывающей

части программы с помощью оператора return. Но метод может одновременно возвратить

лишь *одно* значение. А что, если из метода требуется возвратить два или более

фрагментов информации, например, целую и дробную части числового значения с

плавающей точкой? Такой метод можно написать, используя модификатор ***out***.

**Использование модификатора параметра ref**

Модификатор параметра ***ref*** принудительно организует вызов по ***ссылке***, а не по

значению. Этот модификатор указывается как при объявлении, так и при вызове метода.

В приведенной ниже программе создается метод Sqr(), возвращающий вместо своего аргумента квадрат его целочисленного значения. Обратите особое внимание на применение и местоположение модификатора ***ref***. (***glava8\_5***)

class RefTest

{

//method will change argumet

//look closer how to use mod ref

public void Sqr(ref int i)

{

i = i \* i;

}

}

class RefDemo

{

static void Main()

{

RefTest ob = new RefTest();

int a = 10;

Console.Write("a before method: " + a);

ob.Sqr(ref a);

Console.WriteLine();

Console.Write("a after method: " + a);

Console.WriteLine();

}

Как видите, модификатор ***ref*** указывается ***перед*** объявлением параметра в самом

методе и ***перед*** аргументом при вызове метода.

В отношении модификатора ref необходимо иметь в виду следующее. Аргументу,

передаваемому по ссылке с помощью этого модификатора, должно быть присвоено

значение *до* вызова метода. Дело в том, что в методе, получающем такой аргумент в

качестве параметра, предполагается, что параметр ссылается на действительное значение.

Следовательно, при использовании модификатора ref в методе нельзя задать

первоначальное значение аргумента.

**Использование модификатора параметра out**

Иногда ссылочный параметр требуется использовать для получения значения из

метода, а не для передачи ему значения. Допустим, что имеется метод, выполняющий

некоторую функцию, например, открытие сетевого сокета и возврат кода успешного

или неудачного завершения данной операции в качестве ссылочного параметра.

В этом случае методу не передается никакой информации, но в то же время он должен

возвратить определенную информацию. Главная трудность при этом состоит в том,

что параметр типа ***ref*** должен быть инициализирован определенным значением до

вызова метода. Следовательно, чтобы воспользоваться параметром типа ***ref***, придется

задать для аргумента фиктивное значение и тем самым преодолеть данное ограничение.

Правда, в C# имеется более подходящий вариант выхода из подобного затруднения

— воспользоваться модификатором параметра ***out***.

Модификатор параметра ***out*** подобен модификатору ***ref***, за одним исключением:

он служит ***только*** для передачи значения за пределы метода. Поэтому переменной,

используемой в качестве параметра ***out***, не нужно (да и бесполезно) присваивать

какое-то значение. Более того, в методе параметр ***out*** считается ***неинициализированным****,*

т.е. предполагается, что у него отсутствует первоначальное значение. Это

означает, что значение должно быть присвоено данному параметру в методе ***до***его

завершения. Следовательно, после вызова метода параметр out будет содержать некоторое

значение. Своими словами, ***out*** позволяет изменить фрагмент внутри какого – нибудь метода. Без возвращения. То есть можно было бы создать отдельный метод для изменения переменной, но ***out*** позволяет это сделать без него. Так же важно, что переменная должна быть ***пустой*** (неинициализированной). (***glava8\_6***)

class Decompose

{

//divide float number on int and fraction

public int GetParts(double n, out double frac)

{

int whole;

whole = (int)n;

frac = n - whole; //get fraction part through parameter frac

return whole; //return integer part

}

}

class UseOut

{

static void Main()

{

Decompose ob = new Decompose();

int i;

double f;

i = ob.GetParts(10.125, out f);

Console.WriteLine("Integer part: " + i);

Console.WriteLine("Fraction part: " + f);

}

}

Метод GetParts() возвращает два фрагмента информации. Во-первых, целую

часть исходного числового значения переменной ***n*** обычным образом с помощью оператора

return. И во-вторых, дробную часть этого значения посредством параметра

***frас*** типа ***out***. Как показывает данный пример, используя модификатор параметра

out, можно организовать возврат двух значений из одного и того же метода.

Разумеется, никаких ограничений на применение параметров ***out*** в одном методе

не существует. С их помощью из метода можно возвратить сколько угодно фрагментов

информации. Рассмотрим пример применения двух параметров ***out***. (***glava8\_6.1***)

**Использование модификаторов ref и out для ссылок на объекты**

Применение модификаторов ref и out не ограничивается только передачей значений

обычных типов. С их помощью можно также передавать ссылки на объекты. Если

модификатор ref или out указывает на ссылку, то сама ссылка передается по ссылке.

Это позволяет изменить в методе объект, на который указывает ссылка. Рассмотрим в

качестве примера следующую программу, в которой ссылочные параметры типа ref

служат для смены объектов, на которые указывают ссылки.

class RefSwap

{

int a, b;

public RefSwap(int i, int j)

{

a = i;

b = j;

}

public void Show()

{

Console.WriteLine("a: {0}, b: {1}", a, b);

}

//change args

public void Swap(ref RefSwap ob1, ref RefSwap ob2)

{

RefSwap t;

t = ob1;

ob1 = ob2;

ob2 = t;

}

}

class RefSwapDemo

{

static void Main()

{

RefSwap x = new RefSwap(1, 2);

RefSwap y = new RefSwap(3, 4);

Console.Write("x before method: ");

x.Show();

Console.Write("y before method: ");

y.Show();

Console.WriteLine();

//swap obj

x.Swap(ref x, ref y);

Console.Write("x after method: ");

x.Show();

Console.Write("y after method: ");

y.Show();

}

}

В данном примере в методе Swap() выполняется смена объектов, на которые ссылаются

два его аргумента. До вызова метода Swap() аргумент х ссылается на объект,

содержащий значения 1 и 2, тогда как аргумент у ссылается на объект, содержащий

значения 3 и 4. А после вызова метода Swap() аргумент х ссылается на объект, содержащий

значения 3 и 4, тогда как аргумент у ссылается на объект, содержащий значения

1 и 2. *Если бы не параметры типа ref, то перестановка в методе Swap() не имела*

*бы никаких последствий* за пределами этого метода. Для того чтобы убедиться в этом,

исключите параметры типа ref из метода Swap().

**Использование переменного числа аргументов params**

Допустим, что требуется метод, обнаруживающий наименьшее среди ряда значений. Такому методу можно было бы передать не менее двух, трех, четырех или еще больше значений. Но

в любом случае метод должен возвратить наименьшее из этих значений. Такой метод

нельзя создать, используя обычные параметры. Вместо этого придется воспользоваться

специальным типом параметра, обозначающим произвольное число параметров.

И это делается с помощью создаваемого параметра типа ***params***.

Для объявления массива параметров, способного принимать от нуля до нескольких

аргументов, служит модификатор ***params***. Число элементов массива параметров будет

равно числу аргументов, передаваемых методу. А для получения аргументов в программе

организуется доступ к данному массиву. (***glava8\_7***)

class Min

{

public int MinVal(params int[] nums)

{

int m;

if (nums.Length == 0)

{

Console.WriteLine("No arguments!.");

return 0;

}

m = nums[0];

for (int i = 1; i < nums.Length; i++)

if (nums[i] < m) m = nums[i];

return m;

}

}

//call method 2 args

min = ob.MinVal(a, b);

Console.WriteLine("The least = " + min);

//call method with 3 args

min = ob.MinVal(a, b, -1);

Console.WriteLine("The least = " + min);

//call method with 5 args

min = ob.MinVal(18, 23, 3, 14, 25);

Console.WriteLine("The least = " + min);

//call method with array

int[] args = { 45, 67, 34, 9, 112, 8 };

min = ob.MinVal(args);

Console.WriteLine("The least = " + min);

Console.WriteLine();

Всякий раз, когда вызывается метод MinVal(), ему передаются аргументы в массиве

nums. Длина этого массива равна числу передаваемых аргументов. Поэтому с помощью

метода MinVal() можно обнаружить наименьшее среди любого числа значений.

Обратите внимание на последний вызов метода MinVal(). Вместо отдельных значений

в данном случае передается массив, содержащий ряд значений. И такая передача

аргументов вполне допустима. Когда создается параметр типа ***params***, он воспринимает

список аргументов переменной длины или же массив, содержащий аргументы.

Несмотря на то что параметру типа ***params*** может быть передано любое число

аргументов, все они должны иметь тип массива, указываемый этим параметром.

(Работает отлично, в С++ пришлось бы создавать кучу шаблонов для разного размера параметров, либо же использовать троеточие).

Пользоваться модификатором params следует осторожно, соблюдая граничные

условия, так как параметр типа params может принимать любое число аргументов —

даже ***нулевое***! Например, вызов метода MinVal() в приведенном ниже фрагменте кода

считается правильным с точки зрения синтаксиса С#.

Именно поэтому в методе MinVal() организована проверка на наличие в массиве

nums хотя бы одного элемента перед тем, как пытаться получить доступ к этому

элементу. Если бы такой проверки не было, то при вызове метода MinVal() без аргументов

возникла бы исключительная ситуация во время выполнения.

У метода могут быть как обычные параметры, так и параметр переменной длины.

В качестве примера ниже приведена программа, в которой метод ShowArgs()

принимает один параметр типа string, а также целочисленный массив в качестве

параметра типа ***params***. (***glava8\_7.1***)

class MyClass

{

public void ShowArgs(string msg, params int[] nums)

{

Console.Write(msg + " ");

foreach (int i in nums)

Console.Write(i + " ");

Console.WriteLine();

}

}

class ParamsDemo2

{

static void Main()

{

MyClass ob = new MyClass();

ob.ShowArgs("This is line of int nums: "

, 1, 2, 3, 4, 5);

ob.ShowArgs("Two more int nums: ", 17, 20);

}

}

В тех случаях, когда у метода имеются обычные параметры, а также параметр переменной

длины типа ***params***, он должен быть указан последним в списке параметров

данного метода. Но в любом случае параметр типа ***params*** должен быть единственным.