# **Методи C#**

**Метод** – це конструкція у вигляді блоку коду, яка виконує певні дії, та має закінчене смислове навантаження. Аналогом методів мови C#, є математичні функції. В мові C#, такого розділення немає, хоча умовно можна вважати функціями методи, які повертають значення, а процедурами – методи, що не повертають значень.

Синтаксично метод виглядає наступним чином:

<modifiers> type name(<parameters>)

{

//тіло методу

}

* **modifiers** – необов’язкові модифікатори доступу;
* **type** – тип даних, які повертаються:
  + якщо метод не повертає значень, то використовується тип **void**;
  + якщо повертає, то в тілі методу, повинно бути присутнє, ключове слово **return**, після якого вказують значення(того ж типу, що і метод), яке повертається;
* **name** – ім’я методу;
* **parameters** – необов’язкові параметри(аргументи).

Розглянемо приклади методів:

using System;

class Program

{

static void PrintHelloWorld()

{

Console.WriteLine("Hello World!");

}

static void PrintHello(string name)

{

var text = "Hello " + name + "!";

Console.WriteLine(text);

}

static int Cube(int x)

{

return x \* x \* x;

}

static void Main(string[] args)

{

PrintHelloWorld();

PrintHello("Andrew");

var b1 = Cube(2); //8

var b2 = Cube(3); //27

Console.ReadLine();

}

}

Всі три методи викликаються в основному методі програми – **Main**. Вони мають модифікатори доступу *static*. Перші два методи типу **void**, тобто нічого не повертають.

* **PrintHelloWorld** – викликає Console.WriteLine з текстовим параметром.
* **PrintHello** – приймає в якості аргументу текстовий рядок і після модифікації тексту, передає його в інший метод Console.WriteLine, який виводить текст в консоль.
* **Cube** – приймає на вхід ціле число, підносить його до кубу, та повертає результат.

## **Коли використовувати методи**

Якщо частина коду повторюється два і більше рази, то є сенс винести інструкції в окремий метод. Розглянемо приклад вводу масиву з клавіатури:

var a = new int[10];

var b = new int[7];

for (int i = 0; i < a.Length; i++)

{

Console.Write($"a[{i}] = ");

a[i] = Convert.ToInt32(Console.ReadLine());

}

for (int i = 0; i < b.Length; i++)

{

Console.Write($"b[{i}] = ");

b[i] = Convert.ToInt32(Console.ReadLine());

}

Як аргументи, методу передаються: назва масиву та кількість елементів(довжина).

Після цього, метод можна багаторазово використовувати, для створення нових масивів цілих чисел, які вводить користувач:

static void Main(string[] args)

{

int[] t = GetArrayFromConsole("t", 5);

var g = GetArrayFromConsole("g", 4);

}

## **Оператор return**

**Оператор return** завершує виконання методу, та може повертати значення. Інструкції розміщені після return ігноруються. В методі дозволяється багатократне використання оператора return, однак інколи це ускладнює читання коду програми. Якщо метод має тип void – оператор return може використовуватись для дострокового виходу:

static void Test(int i)

{

if (i <= 0 || i > 5)

{

return;

Console.Write("123"); //цей рядок завжди ігнорується

}

Console.WriteLine(i);

}

static string ZeroCompare(double number)

{

if (number < 0)

{

return "Число менше нуля";

}

else if (number > 0)

{

return "Число більше нуля";

}

return "Число рівне нулю";

}

### **Передача параметрів по посиланню ref**

Для передачі змінних **за посиланням**, використовується модифікатор **ref**. Якщо параметр позначений ключовим словом ref, то будь які зміни цього параметру в тілі методу, відображаються на передану змінну:

using System;

class Program

{

static int Mult(ref int x1, int x2)

{

x1 \*= x2; //зміна x1 впливає на змінну n1

return x1;

}

static void Main(string[] args)

{

var n1 = 24;

var n2 = 4;

var mult = Mult(ref n1, n2);

//змінна n1 містить нове значення

Console.WriteLine($"{n1}");

Console.ReadLine();

}

}

### **Вихідні параметри**

Вище розглянуті параметри мають назву **вхідні**, оскільки передаються на вхід методу. Крім цього, в мові C#, є **вихідні параметри**. Для того, щоб позначити параметр як вихідний, використовується **модифікатор out**.

using System;

class Program

{

static string Div(int a, int b, out int result)

{

if (b == 0)

{

result = int.MinValue;

return "На нуль ділити не можна!";

}

else

{

result = a / b;

return "";

}

}

static void PrintResult(string errorText, int res)

{

if (string.IsNullOrEmpty(errorText))

{

Console.WriteLine(res);

}

else

{

Console.WriteLine(errorText);

}

}

static void Main(string[] args)

{

int r1;

var err1 = Div(64, 8, out r1);

PrintResult(err1, r1);

//змінна r2 оголошується в списку аргументів

var err2 = Div(34, 0, out int r2);

PrintResult(err2, r2);

Console.ReadLine();

}

}

* змінна оголошується перед викликом методу;
* змінна оголошується в списку аргументів.
* Для всіх розглянутих методів використовується модифікатор **static**, оскільки передбачається їх безпосереднє використання в головному методі програми – **Main**.

# **Параметри масиви**

При використанні масиву в якості параметра, синтаксис не відрізняється від інших типів даних, які ми раніше використовували, розглянемо приклад програми для пошуку максимального значення в масиві:

using System;

class Program

{

static int ArrayMax(int[] array)

{

var maxValue = int.MinValue;

foreach (var element in array)

{

if (maxValue < element)

{

maxValue = element;

}

}

return maxValue;

}

static void Main(string[] args)

{

var a = new[] { 1, 2, 3, 4, 5, 6, 7, 8, 9 };

Console.WriteLine($"Максимальне значення масиву {ArrayMax(a)}");

Console.ReadLine();

}

}

Масиви є типом даних за посиланням і передаються по посиланню, це означає, що будь які зміни переданого в якості аргументу масиву всередині методу, відображаються на масиві в основній програмі. Використання масивів в якості параметрів не накладає ніяких обмежень, їх можна використовувати разом з іншими типами даних, в будь якій послідовності.

## **Змінна кількість аргументів**

Ключове слово **params** дозволяє відмітити параметр методу, який приймає змінну кількість аргументів.

При використанні модифікатора params на метод накладаються наступні обмеження:

* Параметр, відмічений ключовим словом params, повинен бути **одновимірним** масивом;
* Після ключового слова params не допускається оголошення інших параметрів, іншими словами він повинен бути останнім в списку параметрів;
* Можна використовувати тільки один параметр відмічений як params;
* Аргументи повинні мати той же тип, який вказаний при оголошенні параметру;
* Список аргументів розділяється комою;
* Якщо аргументи не передаються, то список params пустий – Length == 0.

Метод зі змінним числом аргументів може приймати на вхід масив того ж типу.

Приклад використання змінної кількості параметрів в методі для пошуку мінімального значення зі списку:

using System;

class Program

{

static float Minimal(params float[] values)

{

var min = float.MaxValue;

for (var i = 0; i < values.Length; i++)

{

if (min > values[i])

{

min = values[i];

}

}

return min;

}

static void Main(string[] args)

{

Console.WriteLine(Minimal()); //3.4E38

Console.WriteLine(Minimal(2, 5)); //2

Console.WriteLine(Minimal(6, 5, 4, 3, 1));//1

Console.WriteLine(Minimal(new[] { 5f, 9f })); //5

Console.ReadLine();

}

}

# **Перевантаження методів**

**Перевантаження методів** – створення однойменних методів, в межах одного класу, які відрізняються кількістю і/або типом параметрів. Перевантажені методи можуть повертати значення різних типів, однак відмінність тільки в типі методу не дозволяється.

## **Сигнатури**

**Сигнатура методу** – це частина оголошення методу, яка дозволяє компілятору ідентифікувати метод серед інших.

До сигнатури входять:

* Ім’я методу;
* Кількість параметрів;
* Порядок параметрів;
* Тип параметрів;
* Модифікатори параметрів.

*Назви параметрів та тип значення яке повертається до сигнатури не входять.*

Спираючись на сигнатуру, компілятор вибирає метод, який необхідно використовувати.

int Div(int a, int b)

{

return a / b;

}

uint Sum(uint x, uint y, uint z)

{

return x + y + z;

}

Метод Div має наступну сигнатуру **Div(int, int)**, а метод Sum – **Sum(uint, uint, uint)**.

## **Перевантаження**

Виходячи з поняття сигнатури, **перевантаженими** називають методи, які відрізняються сигнатурами, але при цьому мають однакові імена.

Приклад перевантаження:

public int Mult(int a, int b)

{

return a \* b;

}

public double Mult(double x, double y)

{

return x \* y;

}

public double Mult(double x, double y, double z)

{

//викликає попередній метод

return Mult(x, y) \* z;

}

public string Mult(string s, uint k)

{

var retVal = string.Empty;

for (var i = 0; i < k; i++)

{

retVal += s;

}

return retVal;

}

Як можна побачити, в кожному з розглянутих прикладів використовується унікальна сигнатура. Також C# підтримує скорочений запис перевантажених методів:

float F(float x) => x - 2 / x;

int F(int x) => x - 2 / x;

Перевантажені методи можуть відрізнятися тільки модифікаторами:

int PlusOne(int i)

{

return i + 1;

}

int PlusOne(ref int i)

{

i++;

return i;

}

## **Для чого використовувати перевантаження методів?**

Перевантаження використовується для створення універсальних методів, логіка поведінки яких однакова, але типи чи кількість аргументів різні. Це дає змогу писати красивий код, групуючи методи з одинаковою логікою по іменах.

Розглянемо приклад пошуку мінімального значення з двох цілих чисел.

static int Min(int n1, int n2)

{

return n1 < n2 ? n1 : n2;

}

Використовуючи перевантаження можна збільшити кількість аргументів, для знаходження мінімального з трьох чисел:

static int Min(int n1, int n2, int n3)

{

//викликаємо попередній метод

var m = Min(n1, n2);

return m < n3 ? m : n3;

}

### **Методи з не обов’язковими параметрами**

Розглянемо метод з параметрами за замовчуванням:

static void ShowSum(byte a, byte b, byte c = 5)

{

Console.WriteLine(a + b + c);

}

Здається, до цього методу підходять одразу дві сигнатури: **ShowSum(byte, byte, byte)** та **ShowSum(byte, byte)**, та це не так, підходить тільки перший варіант. Тому якщо його перевантажити методом з двома параметрами:

static void ShowSum(byte x, byte y)

{

Console.WriteLine(x + y);

}

він буде мати більший пріоритет, тому аргумент за замовчуванням не використовується. Для виходу з ситуації, можна використати іменовані параметри:

ShowSum(2, 3); //5, а не 10, як можна очікувати

ShowSum(2, 3, 1); //6

ShowSum(a: 2, b: 3); //10

# **Рекурсія**

**Рекурсія** – конструкція, в якій метод напряму(*пряма рекурсія*) або через інші методи(*складна рекурсія*) викликає себе. Кількість вкладених викликів методу називають **глибиною рекурсії**. Рекурсивні методи дають можливість створювати повторювані обчислення без використання циклічних структур.