

Методи

Якщо змінні зберігають деякі значення, то методи містять собою набір операторів, які виконують певні дії. По суті метод - це іменований блок коду, який виконує деякі дії.

Загальне визначення методів виглядає наступним чином:

```
1 [модификаторы] тип_возвращаемого_значения название_метода ([параметры])
2 {
3     // тело метода
4 }
```

Модифікатори і параметри необов'язкові.

Наприклад, за замовчуванням консольна програма на мові C# має містити як мінімум один метод - метод Main, який є точкою входу в додаток:

```
1 static void Main(string[] args)
2 {
3
4 }
```

Ключове слово **static** є модифікатором. Далі йде тип значення. В даному випадку ключове слово **void** вказує на те, що метод нічого не повертає. Далі йде назва методу - **Main** і в дужках параметри - **string[] args**. І в фігурні дужки укладено тіло методу - всі дії, які він виконує. В даному випадку метод Main порожній, він не містить ніяких операторів і по суті нічого не виконує.

Визначимо ще пару методів:

```
1 using System;
2
3 namespace HelloApp
4 {
5     class Program
6     {
7         static void Main(string[] args)
8         {
9
10        }
11
12        static void SayHello()
13        {
14            Console.WriteLine("Hello");
15        }
16        static void SayGoodbye()
17        {
18            Console.WriteLine("GoodBye");
19        }
20    }
21 }
```

В даному випадку визначені ще два методи: **SayHello** і **SayGoodbye**.

Обидва методи визначені в рамках класу **Program**, вони мають модифікатор **static**, а в якості повертається типу для них визначено тип **void**. Тобто дані методи нічого не повертають, просто роблять деякі дії. І також обидва методи не мають ніяких параметрів, тому після назви методу вказані порожні дужки.

Обидва методи виводять на консоль деяку рядок. Причому для виведення на консоль методи використовують інший метод, який визначений в .NET за замовчуванням - **Console.WriteLine()**.

Але якщо ми запустимо дану програму, то ми не побачимо ніяких повідомлень, які повинні виводити методи SayHello і SayGoodbye. Тому що стартовою точкою є метод Main. При запуску програми виконується тільки метод Main і всі оператори, які складають тіло цього методу. Всі інші методи не виконуються.

Загальна форма	Приклад.
<div data-bbox="367 801 761 869" style="border: 1px solid black; padding: 2px; margin: 10px auto; width: fit-content;">заголовок функції</div> <pre data-bbox="244 913 798 1052">static <тип результату> <ім'я функції>(<список форм. парам.>){ //тіло функції }</pre>	<p data-bbox="938 757 1449 786">Знайти максимальне з трьох цілих чисел</p> <pre data-bbox="885 790 1412 1052">static int Max(int c1,int c2,int c3) { int m = c1; if (c2 > m) m = c2; if (c3 > m) m = c3; return m; }</pre>

Виклик методів

Щоб використовувати методи **SayHello** і **SayGoodbye** в програмі, нам треба викликати їх в методі **Main**.

Для виклику методу вказується його ім'я, після якого в дужках йдуть значення для його параметрів (якщо метод приймає параметри).

```
1  название_метода (значения_для_параметров_метода);
```

Наприклад, викличемо методи SayHello і SayGoodbye:

```

class Program
{
    static void Main(string[] args)
    {
        SayHello();
        SayGoodbye();

        Console.ReadKey();
    }

    static void SayHello()
    {
        Console.WriteLine("Hello");
    }
    static void SayGoodbye()
    {
        Console.WriteLine("GoodBye");
    }
}

```

```

Hello
GoodBye

```

Повернення значення

Метод може повертати значення, будь-який результат. В наведеному вище прикладі були визначені два методу, які мали тип **void**. Методи з таким типом не повертають жодного значення. Вони просто виконують деякі дії.

Якщо метод має будь-який інший тип, відмінний від **void**, то такий метод зобов'язаний повернути значення цього типу. Для цього застосовується оператор **return**, після якого йде повертається значення:

```

1 | return возвращаемое значение;

```

Наприклад, визначимо ще пару методів:

```

1 | static string GetHello()
2 | {
3 |     return "Hello";
4 | }
5 | static int GetSum()
6 | {
7 |     int x = 2;
8 |     int y = 3;
9 |     int z = x + y;
10 |    return z;
11 | }

```

Метод **GetHello** має тип **string**, отже, він повинен повернути рядок. Тому в тілі методу використовується оператор **return**, після якого вказана рядок, що повертається.

Метод **GetSum** має тип **int**, отже, він повинен повернути значення типу **int** - ціле число. Тому в тілі методу використовується оператор **return**, після якого зазначено, що повертається число (в даному випадку результат суми змінних *x* і *y*).

Після оператора **return** також можна вказувати складні вирази, які повертають певний результат. наприклад:

```
1 static int GetSum()  
2 {  
3     int x = 2;  
4     int y = 3;  
5     return x + y;  
6 }
```

При цьому методи, які повертають будь-який тип мають, відмінний від **void**, обов'язково повинні використовувати оператор **return** для повернення значення. Наприклад, таке визначення методу некоректно:

```
1 static string GetHello()  
2 {  
3     Console.WriteLine("Hello");  
4 }
```

Також між типом методу, що повертається, і значенням після оператора **return** має бути відповідність. Наприклад, в наступному випадку повертається тип - **int**, але метод повертає рядок (тип **string**), тому таке визначення методу некоректно:

```
1 static int GetSum()  
2 {  
3     int x = 2;  
4     int y = 3;  
5     return "5"; // помилка - надо возвращать число  
6 }
```

Результат методів, який повертають значення, ми можемо присвоїти змінним чи іншим чином користуватися в програмі:

```

static void Main(string[] args)
{
    string message = GetHello();
    int sum = GetSum();

    Console.WriteLine(message); // Hello
    Console.WriteLine(sum);     // 5

    Console.ReadKey();
}

static string GetHello()
{
    return "Hello";
}
static int GetSum()
{
    int x = 2;
    int y = 3;
    return x + y;
}

```

Метод **GetHello** повертає значення типу **string**. Тому ми можемо присвоїти це значення якої-небудь змінної типу **string**: **string message = GetHello();**

Другий метод - **GetSum** - повертає значення типу **int**, тому його можна привласнити змінної, яка приймає значення цього типу: **int sum = GetSum();**.

Вихід з методу

Оператор **return** не тільки повертає значення, а й виробляє вихід з методу. Тому він повинен визначатися після усіх інструкцій. наприклад:

```

1 static string GetHello()
2 {
3     return "Hello";
4     Console.WriteLine("After return");
5 }

```

З точки зору синтаксису даний метод коректний, проте його інструкція **Console.WriteLine("After return")** не має сенсу - вона ніколи не виконається, так як до її виконання оператор **return** поверне значення і зробить вихід з методу.

Однак ми можемо використовувати оператор **return** і в методах з типом **void**. У цьому випадку після оператора **return** не ставиться ніякого значення, що повертається (адже метод нічого не повертає). Типова ситуація - в залежності від умови призвести вихід з методу:

```

1 static void SayHello()
2 {
3     int hour = 23;
4     if(hour > 22)
5     {
6         return;
7     }
8     else
9     {
10        Console.WriteLine("Hello");
11    }
12 }

```

Скорочений запис методів

Якщо метод як тіла визначає тільки одну інструкцію, то ми можемо скоротити визначення методу. Наприклад, припустимо у нас є метод:

```

1 static void SayHello()
2 {
3     Console.WriteLine("Hello");
4 }

```

Ми можемо скоротити наступним чином:

```

1 static void SayHello() => Console.WriteLine("Hello");

```

Тобто після списку параметрів ставиться знак рівності і більше ніж, після якого йде виконувана інструкція.

Подібним чином ми можемо скорочувати методи, які повертають значення:

```

1 static string GetHello()
2 {
3     return "hello";
4 }

```

Аналогічне наступним алгоритмом:

```

1 static string GetHello() => "hello";

```

Параметри методів

Параметри дозволяють передати в метод деякі вхідні дані. Наприклад, задамо метод, який складає два числа:

```

1 static int Sum(int x, int y)
2 {
3     return x + y;
4 }

```

Метод Sum має два параметри: x і y. Обидва параметри представляють тип int. Тому при виклику даного методу нам обов'язково треба передати на місце цих параметрів два числа.

```

1 class Program
2 {
3     static void Main(string[] args)
4     {
5         int result = Sum(10, 15);
6         Console.WriteLine(result); // 25
7
8         Console.ReadKey();
9     }
10    static int Sum(int x, int y)
11    {
12        return x + y;
13    }
14 }

```

При виклику методу **Sum** значення передаються параметрам по позиції. Наприклад, у виклику **Sum(10, 15)** число 10 передається параметру **x**, а число **15** - параметру **y**. Значення, які передаються параметрам, ще називаються аргументами. Тобто передані числа 10 і 15 в даному випадку є аргументами.

Іноді можна зустріти такі визначення як формальні параметри і фактичні параметри. Формальні параметри - це власне параметри методу (в даному випадку x і y), а фактичні параметри - значення, які передаються формальним параметрам. Тобто фактичні параметри - це і є аргументи методу.

Передані параметру значення можуть представляти значення змінних або результат роботи складних виразів, які повертають деяке значення:

```

{
    static void Main(string[] args)
    {
        int a = 25;
        int b = 35;
        int result = Sum(a, b);
        Console.WriteLine(result); // 60

        result = Sum(b, 45);
        Console.WriteLine(result); // 80

        result = Sum(a + b + 12, 18); // "a + b + 12" представляет значение параметра x
        Console.WriteLine(result); // 90

        Console.ReadKey();
    }
    static int Sum(int x, int y)
    {
        return x + y;
    }
}

```

Якщо параметрами методу передаються значення змінних, то таким змінним має бути присвоєно значення. Наприклад, наступна програма НЕ скомпілюється:

```

class Program
{
    static void Main(string[] args)
    {
        int a;
        int b = 9;
        Sum(a, b); // Ошибка - переменной a не присвоено значение

        Console.ReadKey();
    }
    static int Sum(int x, int y)
    {
        return x + y;
    }
}

```

При передачі значень параметрам важливо враховувати тип параметрів: між аргументами і параметрами має бути відповідність за типом. наприклад:


```

class Program
{
    static void Main(string[] args)
    {
        Display("Tom", 24); // Name: Tom Age: 24

        Console.ReadKey();
    }
    static void Display(string name, int age)
    {
        Console.WriteLine($"Name: {name} Age: {age}");
    }
}

```

В даному випадку перший параметр методу **Display** представляє тип **string**, тому ми повинні передати цим параметром значення типу **string**, тобто рядок. Другий параметр представляє тип **int**, тому повинні передати йому ціле число, яке відповідає типу **int**.

Інші дані параметрам передати не можемо. Наприклад, наступний виклик методу **Display** буде помилковим:

```

1 Display(45, "Bob"); // Ошибка! несоответствие значений типам параметров

```

Необов'язкові параметри

За замовчуванням при виклику методу необхідно надати значення для всіх його параметрів. Але C # також дозволяє використовувати необов'язкові параметри. Для таких параметрів нам необхідно оголосити значення за замовчуванням. Також слід враховувати, що після необов'язкових параметрів всі наступні параметри також мають бути необов'язковими:

```

1 static int OptionalParam(int x, int y, int z=5, int s=4)
2 {
3     return x + y + z + s;
4 }

```

Так як останні два параметри оголошені як необов'язкові, то ми можемо один з них або обидва опустити:

```

1 static void Main(string[] args)
2 {
3     OptionalParam(2, 3);
4
5     OptionalParam(2,3,10);
6
7     Console.ReadKey();
8 }

```

Іменовані параметри

У попередніх прикладах при виклику методів значення для параметрів

передавалися в порядку оголошення цих параметрів в методі. Але ми можемо порушити подібний порядок, використовуючи іменовані параметри:

```

1 static int OptionalParam(int x, int y, int z=5, int s=4)
2 {
3     return x + y + z + s;
4 }
5 static void Main(string[] args)
6 {
7     OptionalParam(x:2, y:3);
8
9     //Необязательный параметр z использует значение по умолчанию
10    OptionalParam(y:2, x:3, s:10);
11
12    Console.ReadKey();
13 }

```

Проілюструємо застосування структурного підходу на прикладі. Обчислити значення виразу

$$S = \underbrace{\max\{a, -b, 3\}}_{\max 1} * \underbrace{\max\{a, 2b, c\}}_{\max 2} - \underbrace{\max\{-a, b, 7\}}_{\max 3} = \max 1 * \max 2 - \max 3$$

Без функцій	З функціями
<pre> class Program { static void Main(string[] args) { int a,b,c; Console.Write("a="); a = int.Parse(Console.ReadLine()); Console.Write("b="); b = int.Parse(Console.ReadLine()); Console.Write("c="); c = int.Parse(Console.ReadLine()); } } </pre>	<pre> class Program { static int Max(int c1, int c2, int c3) { int m = c1; if (c2 > m) m = c2; if (c3 > m) m = c3; return m; } static void Main(string[] args) { int a,b,c; Console.Write("a="); a = int.Parse(Console.ReadLine()); Console.Write("b="); b = int.Parse(Console.ReadLine()); Console.Write("c="); c = int.Parse(Console.ReadLine()); } } </pre>

Без функцій	З функціями
<pre>int max1 = a; if (-b > max1) max1 = -b; if (3 > max1) max1 = 3;</pre>	<pre>int max1 = Max(a, -b, 3);</pre>
<pre>int max2 = a; if (2*b > max2) max2 = 2*b; if (c > max1) max2 = c;</pre>	<pre>int max2 = Max(a, 2*b, c);</pre>
<pre>int max3 = -a; if (b > max3) max3 = b; if (7 > max3) max3 = 7;</pre>	<pre>int max3 = Max(-a, b, 7);</pre>
<pre>int S = max1 * max2 - max3; Console.WriteLine("S={0}", S); Console.ReadKey(); } }</pre>	<pre>int S = max1 * max2 - max3; Console.WriteLine("S={0}", S); Console.ReadKey(); } }</pre>

Передача параметрів по посиланню і значенням. Вихідні параметри

Існує два способи передачі параметрів в метод в мові C #: за значенням і за посиланням.

Передача параметрів за значенням

Найбільш простий спосіб передачі параметрів представляє передача за значенням, по суті це звичайний спосіб передачі параметрів:

```

1  class Program
2  {
3      static void Main(string[] args)
4      {
5          Sum(10, 15);          // параметри передаються по значенню
6          Console.ReadKey();
7      }
8      static int Sum(int x, int y)
9      {
10         return x + y;
11     }
12 }
```

Приклад. Створити функцію, що повертає середнє арифметичне значення трьох дійсних чисел.

```
class Program
{
    static double Average(double c1, double c2, double c3)
    {
        return (c1 + c2 + c3) / 3;
    }
    static void Main(string[] args)
    {
        double n, m, k;
        Console.WriteLine("n=");
        n = double.Parse(Console.ReadLine());
        Console.WriteLine("m=");
        m = double.Parse(Console.ReadLine());
        Console.WriteLine("k=");
        k = double.Parse(Console.ReadLine());
        double ser= Average(n, m, k);
        Console.WriteLine("Average = {0} ",ser);
        Console.ReadKey();
    }
}
```

Передача параметрів по посиланню і модифікатор ref

При передачі параметрів по посиланню перед параметрами використовується модифікатор ref :

```
1 static void Main(string[] args)
2 {
3     int x = 10;
4     int y = 15;
5     Addition(ref x, y); // вызов метода
6     Console.WriteLine(x); // 25
7
8     Console.ReadLine();
9 }
10 // параметр x передается по ссылке
11 static void Addition(ref int x, int y)
12 {
13     x += y;
14 }
```

Зверніть увагу, що модифікатор **ref** вказується, як при оголошенні методу, так і при його виклику в методі **Main**.

У чому відмінність двох способів передачі параметрів? При передачі по значенню метод отримує не саму змінну, а її копію. А при передачі параметра за посиланням метод отримує адресу змінної в пам'яті. І, таким чином, якщо в методі змінюється значення параметра, переданого за посиланням, то також змінюється і значення змінної, яка передається на його місце.

Розглянемо два аналогічних прикладу. Перший приклад - передача параметра за значенням:

```
1 class Program
2 {
3     static void Main(string[] args)
4     {
5         int a = 5;
6         Console.WriteLine($"Начальное значение переменной a = {a}");
7
8         //Передача переменных по значению
9         //После выполнения этого кода по-прежнему a = 5, так как мы передали лишь ее копию
10        IncrementVal(a);
11        Console.WriteLine($"Переменная a после передачи по значению равна = {a}");
12        Console.ReadKey();
13    }
14    // передача по значению
15    static void IncrementVal(int x)
16    {
17        x++;
18        Console.WriteLine($"IncrementVal: {x}");
19    }
20 }
```

Начальное значение переменной a = 5

IncrementVal: 6

Переменная a после передачи по значению равна = 5

При виклику метод **IncrementVal** отримує копію змінної **a** і збільшує значення цієї копії. Тому в самому методі **IncrementVal** ми бачимо, що значення параметра **x** збільшилася на 1, але після виконання методу змінна **a** має колишнє значення - 5. Тобто змінюється копія, а сама змінна не змінюється.

Другий приклад - аналогічний метод з передачею параметра за посиланням:

```

1 class Program
2 {
3     static void Main(string[] args)
4     {
5         int a = 5;
6         Console.WriteLine($"Начальное значение переменной a = {a}");
7         //Передача переменных по ссылке
8         //После выполнения этого кода a = 6, так как мы передали саму переменную
9         IncrementRef(ref a);
10        Console.WriteLine($"Переменная a после передачи ссылке равна = {a}");
11
12        Console.ReadKey();
13    }
14    // передача по ссылке
15    static void IncrementRef(ref int x)
16    {
17        x++;
18        Console.WriteLine($"IncrementRef: {x}");
19    }
20 }

```

```

Начальное значение переменной a = 5
IncrementRef: 6
Переменная a после передачи по ссылке равна = 6

```

У метод **IncrementRef** передається посилання на саму змінну **a** в пам'яті. І якщо значення параметра в **IncrementRef** змінюється, то це призводить і до зміни змінної **a**, так як і параметр і змінна вказують на один і той же адресу в пам'яті.

***Приклад.** Створити функцію, яка більше значення зменшує на 2, а менше збільшує на 3.*

```

class Program
{
    static void Fun1(ref int a, ref int b)
    {
        if (a > b)
        {
            a -= 2;
            b += 3;
        }
        else
        {
            b -= 2;
            a += 3;
        }
    }
    static void Main(string[] args)
    {
        int n,m;
        Console.WriteLine("n=");
        n=int.Parse(Console.ReadLine());
        Console.WriteLine("m=");
        m = int.Parse(Console.ReadLine());
        Fun1(ref n,ref m);
        Console.WriteLine("n= {0} m={1}",n,m);
        Console.ReadKey();
    }
}

```

Вихідні параметри. Модифікатор out

Вище ми використовували вхідні параметри. Але параметри можуть бути також вихідними. Щоб зробити параметр вихідним, перед ним ставиться модифікатор out:

```

1 static void Sum(int x, int y, out int a)
2 {
3     a = x + y;
4 }

```

Тут результат повертається не через оператор **return**, а через вихідний параметр. Використання в програмі:

```

1 static void Main(string[] args)
2 {
3     int x = 10;
4
5     int z;
6
7     Sum(x, 15, out z);
8
9     Console.WriteLine(z);
10
11    Console.ReadKey();
12 }

```

Причому, як і у випадку з **ref** ключове слово **out** використовується як при визначенні методу, так і при його виклику.

Також зверніть увагу, що методи, які використовують такі параметри, обов'язково повинні привласнювати їм певне значення. Тобто наступний код буде неприпустимий, оскільки в ньому для out-параметра не вказано ніякого значення:

```
1 static void Sum(int x, int y, out int a)
2 {
3     Console.WriteLine(x+y);
4 }
```

Особливість використання подібних параметрів полягає в тому, що по суті ми можемо повернути з методу не один варіант, а декілька. наприклад:

```
1 static void Main(string[] args)
2 {
3     int x = 10;
4     int area;
5     int perimetr;
6     GetData(x, 15, out area, out perimetr);
7     Console.WriteLine("Площадь : " + area);
8     Console.WriteLine("Периметр : " + perimetr);
9
10    Console.ReadKey();
11 }
12 static void GetData(int x, int y, out int area, out int perim)
13 {
14     area= x * y;
15     perim= (x + y)*2;
16 }
```

Тут у нас є метод **GetData**, який, припустимо, приймає сторони прямокутника. А два вихідних параметра використовуємо для підрахунку площі і периметра прямокутника.

По суті, як і в випадку з ключовим словом **ref**, ключове слово **out** застосовується для передачі аргументів за посиланням. Однак на відміну від **ref** для змінних, які передаються з ключовими словами **out**, не потрібно ініціалізація. І крім того, що викликається метод повинен обов'язково надати їм значення.

Варто зазначити, що починаючи з версії C # 7.0 можна визначати змінні в безпосередньо при виклику методу. Тобто:


```
int x = 10;
int area;
int perimetr;
GetData(x, 15, out area, out perimetr);
Console.WriteLine($"Площадь : {area}");
Console.WriteLine($"Периметр : {perimetr}");
```

```
int x = 10;
GetData(x, 15, out int area, out int perimetr);
Console.WriteLine($"Площадь : {area}");
Console.WriteLine($"Периметр : {perimetr}");
```

Приклад. Створити функцію, яка повертає максимальне та мінімальне значення із двох дійсних чисел.

```
class Program
{
    static void MinMax(int c1, int c2, out int max, out int min)
    {
        if (c1 > c2)
        {
            max = c1;
            min = c2;
        }
        else
        {
            max = c2;
            min = c1;
        }
    }
    static void Main(string[] args)
    {
        int n,m;
        Console.WriteLine("n=");
        n=int.Parse(Console.ReadLine());
        Console.WriteLine("m=");
        m = int.Parse(Console.ReadLine());
        int max, min;
        MinMax(n, m, out max, out min);
        Console.WriteLine("max= {0} min={1}",max,min);
        Console.ReadKey();
    }
}
```

Вхідні параметри. Модифікатор **in**

Крім вихідних параметрів з модифікатором **out** метод може використовувати вхідні параметри з модифікатором **in**. Модифікатор **in** вказує, що через цей параметр буде передаватися в метод по посиланню, однак всередині методу його значення параметра не можна буде змінити. Наприклад, візьмемо наступний метод:

```

1 static void GetData(in int x, int y, out int area, out int perim)
2 {
3     // x = x + 10; нельзя изменить значение параметра x
4     y = y + 10;
5     area = x * y;
6     perim = (x + y) * 2;
7 }

```

В даному випадку через параметри **x** і **y** в метод передаються значення, але в самому методі можна змінити тільки значення параметра **y**, так як параметр **x** вказано з модифікатором **in**.

Передача за значенням (немає ніяких специфікаторів) (ініціалізація змінної n обов'язкова)	Передача за покажчиком (специфікатор ref) (ініціалізація змінної n обов'язкова)	Аргумент як вихідне значення (специфікатор out) (ініціалізація змінної n не обов'язкова)
<pre> class Program { static void Fun1(int m) { m = 1; } static void Main(string[] args) { int n = 25; Fun1(n); Console.WriteLine("n= {0}",n); Console.ReadKey(); } } </pre>	<pre> class Program { static void Fun1(ref int m) { m = 1; } static void Main(string[] args) { int n = 25; Fun1(ref n); Console.WriteLine("n= {0}",n); Console.ReadKey(); } } </pre>	<pre> class Program { static void Fun1(out int m) { m = 1; } static void Main(string[] args) { int n; Fun1(out n); Console.WriteLine("n= {0}",n); Console.ReadKey(); } } </pre>
Вивід програми: n=25	Вивід програми: n=1	Вивід програми: n=1
Змінні в пам'яті ЕОМ: До виклику: <div style="border: 1px solid black; padding: 5px; margin: 10px auto; width: 100px; text-align: center;"> Програма n 25 </div> Під час роботи функції: (для змінної m виділяється нова пам'ять) <div style="display: flex; justify-content: space-around; border: 1px solid black; padding: 5px; margin: 10px auto; width: 200px;"> <div style="border-right: 1px solid black; padding: 5px; text-align: center;">Програма n 25</div> <div style="padding: 5px; text-align: center;">Функція m 1</div> </div> Після роботи функції: <div style="border: 1px solid black; padding: 5px; margin: 10px auto; width: 100px; text-align: center;"> Програма n 25 </div>	Змінні в пам'яті ЕОМ: До виклику: <div style="border: 1px solid black; padding: 5px; margin: 10px auto; width: 100px; text-align: center;"> Програма n 25 </div> Під час роботи функції: (для змінної m нова пам'ять не виділяється а використовується пам'ять змінної n) <div style="display: flex; justify-content: space-around; border: 1px solid black; padding: 5px; margin: 10px auto; width: 200px;"> <div style="border-right: 1px solid black; padding: 5px; text-align: center;">Програма n 25</div> <div style="padding: 5px; text-align: center;">Функція m ←→ n</div> </div> Після роботи функції: <div style="border: 1px solid black; padding: 5px; margin: 10px auto; width: 100px; text-align: center;"> Програма n 1 </div>	Змінні в пам'яті ЕОМ: До виклику: <div style="border: 1px solid black; padding: 5px; margin: 10px auto; width: 100px; text-align: center;"> Програма n 25 </div> Під час роботи функції: (для змінної m нова пам'ять не виділяється а використовується пам'ять змінної n) <div style="display: flex; justify-content: space-around; border: 1px solid black; padding: 5px; margin: 10px auto; width: 200px;"> <div style="border-right: 1px solid black; padding: 5px; text-align: center;">Програма n 25</div> <div style="padding: 5px; text-align: center;">Функція m ←→ n</div> </div> Після роботи функції: <div style="border: 1px solid black; padding: 5px; margin: 10px auto; width: 100px; text-align: center;"> Програма n 1 </div>