

ЗНАЙОМСТВО З БАЗОВИМ СИНТАКСИСОМ МОВИ ПРОГРАМУВАННЯ C#

Тема 02
Об'єктно-орієнтоване програмування
ЧДБК, 2020

Питання лекції

- Управляючі інструкції мови програмування C#.
- Посилальні числові типи даних.
- Робота з масивами.
- Програмні засоби для роботи з текстом. Регулярні вирази.



УПРАВЛЯЮЧІ ІНСТРУКЦІЇ МОВИ ПРОГРАМУВАННЯ C#

Питання 2.1.

Управляючі інструкції (оператори мови програмування) в C#

- Конструкції прийняття рішень:
 - оператор if-else
 - оператор switch
- Ітераційні оператори C#:
 - цикл for;
 - цикл foreach/in;
 - цикл while;
 - цикл do/while.
- Оператори переходу:
 - break
 - continue
 - goto
 - return
 - throw

```

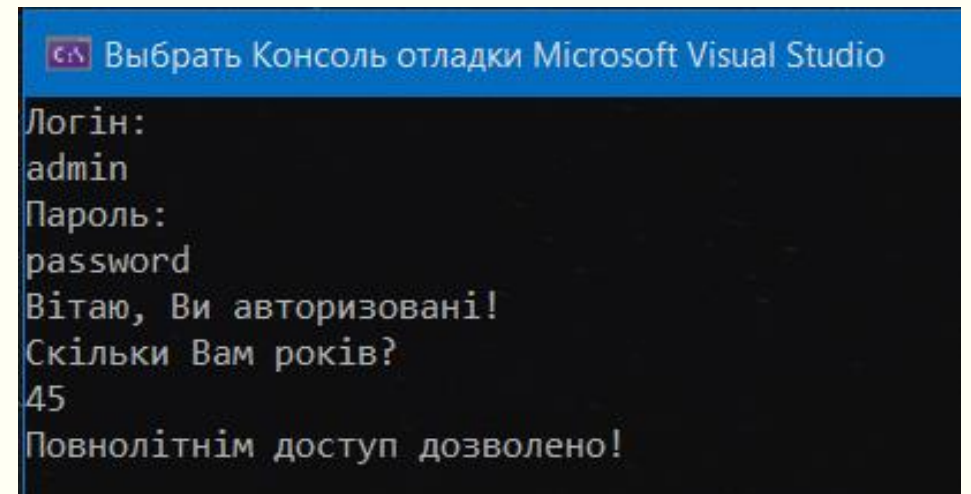
Console.OutputEncoding = Encoding.UTF8;
Console.WriteLine("Логін: ");
string login = Console.ReadLine();
Console.WriteLine("Пароль: ");
string password = Console.ReadLine();

if (login == "admin" && password == "password")
{
    Console.WriteLine("Вітаю, Ви авторизовані!");
    Console.WriteLine("Скільки Вам років?");
    int age;
    if(int.TryParse(Console.ReadLine(), out age)) {
        if (age >= 18)
        {
            Console.WriteLine("Повнолітнім доступ дозволено!");
        }
        else if(age > 0)
        {
            Console.WriteLine("Неповнолітнім доступ заборонено!");
        }
        else
        {
            Console.WriteLine("Спочатку народіться на цей світ!");
        }
    } else
    {
        Console.WriteLine("Некоректно введений вік!");
    }
} else
{
    Console.WriteLine("Неправильний логін чи пароль!");
}

```

Умови та оператор if-else

- На відміну від C та C++, у мові C# може працювати лише з булевими виразами.
 - Використовуються оператори порівняння (==, !=, <, >, <=, >=) та логічні оператори (&&, ||, !).
 - Операції && та || за необхідності підтримують скорочений шлях виконання (до першого false або true).
 - Для перевірки всіх виразів умови можна використовувати побітові операції & та |.



```

Выбрать Консоль отладки Microsoft Visual Studio
Логін:
admin
Пароль:
password
Вітаю, Ви авторизовані!
Скільки Вам років?
45
Повнолітнім доступ дозволено!

```

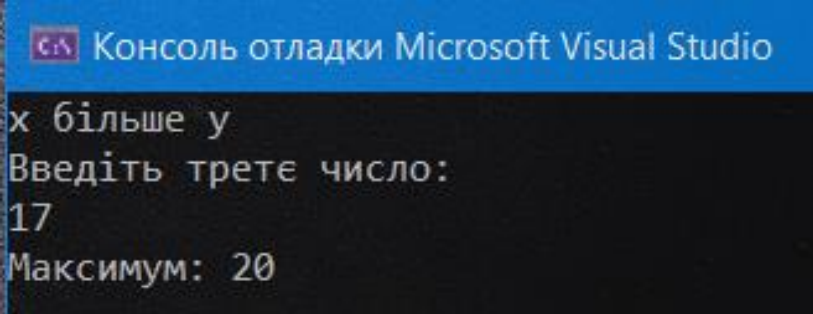
Тернарний оператор ?:

- Правоасоціативний оператор з формою запису
 - Умова ? Вираз1 : Вираз2;

```
int x = 20, y = 10, z;
```

```
var result = x > y ? "x більше y" : "x менше або дорівнює y";  
Console.WriteLine(result);
```

```
Console.WriteLine("Введіть третє число: ");  
// обчислення максимального значення з трьох чисел  
int.TryParse(Console.ReadLine(), out z);  
// вкладений тернарний оператор  
var max = x > y ? x > z ? x : z : y > z ? y : z;  
Console.WriteLine($"Максимум: {max} ");
```



```
Консоль отладки Microsoft Visual Studio  
x більше y  
Введіть третє число:  
17  
Максимум: 20
```

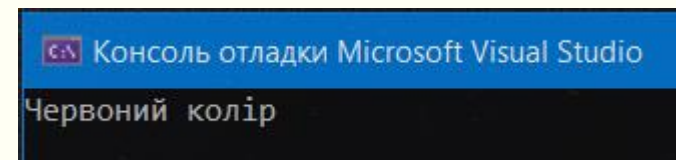
Оператор switch

```
public enum Color { Red, Green, Blue }

class Program
{
    static void Main(string[] args)
    {
        Console.OutputEncoding = Encoding.UTF8;
        Color c = (Color)(new Random()).Next(0, 3);
        switch (c)
        {
            case Color.Red:
                Console.WriteLine("Червоний колір");
                break;
            case Color.Green:
                Console.WriteLine("Зелений колір");
                break;
            case Color.Blue:
                Console.WriteLine("Синій колір");
                break;
            default:
                Console.WriteLine("Даний колір невідомий!");
                break;
        }
    }
}
```

01.09.2020

- Часто використовується як альтернатива конструкції if-else, якщо один вираз перевіряється на відповідність трьом та більше умовам.



```

public enum Directions {
    Up,
    Down,
    Right,
    Left
}

public enum Orientation {
    North,
    South,
    East,
    West
}

static void Main(string[] args)
{
    Console.OutputEncoding = Encoding.UTF8;
    // switch expression
    var direction = Directions.Right;
    Console.WriteLine($"Map view direction is {direction}");

    var orientation = direction switch
    {
        Directions.Up => Orientation.North,
        Directions.Right => Orientation.East,
        Directions.Down => Orientation.South,
        Directions.Left => Orientation.West,
    };
    Console.WriteLine($"Cardinal orientation is {orientation}");
}

```

01.09.2020

@Марченко С.В., ЧДБК, 2020

Вираз switch (представлено в C# 8.0)

- Базові елементи виразу switch:
 - *вираз діапазону (range expression)*: у прикладі – змінна direction.
 - *вітки виразу switch (switch expression arms)*: кожна містить *шаблон (pattern)*, необов'язкову *умову регістра (optional case guard)*, токен => та вираз.
- Результат виразу switch – значення виразу першої з віток, шаблон якої відповідає виразу діапазону, а умова регістру за наявності, буде мати значення true.
 - Вираз справа від токена => не може бути інструкцією.

Ключове слово when

- Починаючи з C# 7.0, знімається вимога, щоб case-мітки (case labels) були взаємовиключними.
 - Виконання блоку switch регламентується порядком появи case-міток.
 - Ключове слово when дозволяє задавати умову фільтрування, пов'язану з case-міткою.
 - Корисна при поліморфній природі case-мітки (уточнення породженого типу).
- Для оператора switch синтаксис такий:
 - `case Shape shape when sh.Area == 0`
 - `case Rectangle r when r.Length == r.Width && r.Area > 0`
- Для виразу switch (алмазна нотація розглядатиметься в темі 5):
 - `IEnumerable<T> list when !list.Any() => default(T),`
 - `IEnumerable<T> list when list.Count() < 3 => list.Last(),`

Цикл for

- Дозволяє вказати, скільки разів повинен повторюватись блок коду, а також задати умову завершення.
 - Синтаксис аналогічний до мов C, C++ та Java.
 - Можна створювати складні умови завершення, будувати нескінченні цикли, цикли в зворотному напрямку й використовувати ключові слова для переходів goto, continue та break.

```
for ([ініціалізація лічильника]; [умова]; [зміна лічильника])  
{  
    // дії  
}
```

{

class Program

{

static void Main(string[] args)

{

Console.OutputEncoding = Encoding.UTF8;

for (int i = 0; i < 10; i++) {

Console.WriteLine(\$"Квадрат числа {i} дорівнює {i * i}");

}

Console.WriteLine();

int j = 0;

for (; ;) {

if(j == 10) {

break;

}

Console.WriteLine(\$"Квадрат числа {++j} дорівнює {j * j}");

}

Console.WriteLine();

j = 0;

for (; j < 10;) {

Console.WriteLine(\$"Квадрат числа {++j} дорівнює {j * j}");

}

Console.WriteLine();

int number;

for (; int.TryParse(Console.ReadLine(), out number);) {

Console.WriteLine(\$"Квадрат числа {number} дорівнює {number * number}");

}

}

}

{

Приклади використання циклу for

01.09.2020

@Марченко С.В., ЧДБК, 2020

Квадрат числа 0 дорівнює 0
 Квадрат числа 1 дорівнює 1
 Квадрат числа 2 дорівнює 4
 Квадрат числа 3 дорівнює 9
 Квадрат числа 4 дорівнює 16
 Квадрат числа 5 дорівнює 25
 Квадрат числа 6 дорівнює 36
 Квадрат числа 7 дорівнює 49
 Квадрат числа 8 дорівнює 64
 Квадрат числа 9 дорівнює 81

Квадрат числа 10 дорівнює 100
 Квадрат числа 9 дорівнює 81
 Квадрат числа 8 дорівнює 64
 Квадрат числа 7 дорівнює 49
 Квадрат числа 6 дорівнює 36
 Квадрат числа 5 дорівнює 25
 Квадрат числа 4 дорівнює 16
 Квадрат числа 3 дорівнює 9
 Квадрат числа 2 дорівнює 4
 Квадрат числа 1 дорівнює 1

Квадрат числа 1 дорівнює 1
 Квадрат числа 2 дорівнює 4
 Квадрат числа 3 дорівнює 9
 Квадрат числа 4 дорівнює 16
 Квадрат числа 5 дорівнює 25
 Квадрат числа 6 дорівнює 36
 Квадрат числа 7 дорівнює 49
 Квадрат числа 8 дорівнює 64
 Квадрат числа 9 дорівнює 81
 Квадрат числа 10 дорівнює 100

Квадрат числа 1 дорівнює 1
 Квадрат числа 2 дорівнює 4
 Квадрат числа 3 дорівнює 9
 Квадрат числа 4 дорівнює 16
 Квадрат числа 5 дорівнює 25
 Квадрат числа 6 дорівнює 36
 Квадрат числа 7 дорівнює 49
 Квадрат числа 8 дорівнює 64
 Квадрат числа 9 дорівнює 81
 Квадрат числа 10 дорівнює 100

7
 Квадрат числа 7 дорівнює 49
 9
 Квадрат числа 9 дорівнює 81
 9

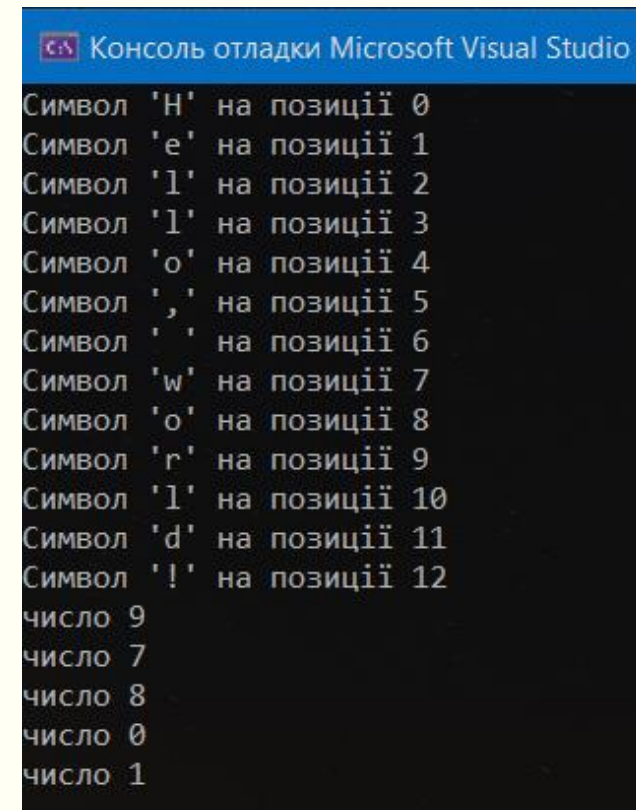
Цикл foreach

- Дозволяє проходити в циклі по всіх елементах контейнера, не вимагаючи перевірки його меж.
 - На відміну від циклу for, цикл foreach буде виконувати прохід по контейнеру тільки лінійно (не можна обходити елементи контейнера в зворотному напрямку, пропускати елементи тощо).

```
var str = "Hello, world!";
var ints = new int[] { 9, -6, 7, 8, 0, 1 };

int index = 0;
foreach (var c in str)
{
    Console.WriteLine($"Символ '{c}' на позиції {index}");
    index++;
}

foreach (int i in ints)
{
    if (i < 0)
        continue;
    Console.WriteLine("число {0}", i);
}
```



```
Консоль отладки Microsoft Visual Studio
Символ 'H' на позиції 0
Символ 'e' на позиції 1
Символ 'l' на позиції 2
Символ 'l' на позиції 3
Символ 'o' на позиції 4
Символ ',' на позиції 5
Символ ' ' на позиції 6
Символ 'w' на позиції 7
Символ 'o' на позиції 8
Символ 'r' на позиції 9
Символ 'l' на позиції 10
Символ 'd' на позиції 11
Символ '!' на позиції 12
число 9
число 7
число 8
число 0
число 1
```

Цикли while та do-while

```
int k = 0;

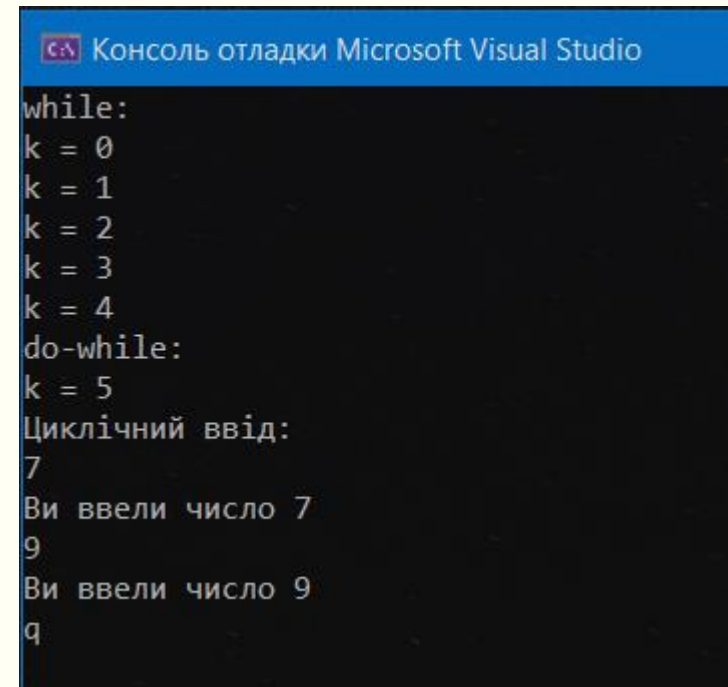
Console.WriteLine("while: ");
while (k < 5) {
    Console.WriteLine($"k = {k}");
    k++;
}

Console.WriteLine("do-while: ");
do {
    Console.WriteLine($"k = {k}");
    k++;
} while (k > 10);

Console.WriteLine("Циклічний ввід:");
while(true) {
    if (!int.TryParse(Console.ReadLine(), out k))
        break;

    Console.WriteLine($"Ви ввели число {k}");
}
```

- Ітераційну конструкцію while зручно застосовувати, коли потрібно, щоб блок операторів виконувався доти, доки не буде задоволено умову завершення.
 - Цикл do-while гарантує виконання відповідного блоку коду принаймні один раз.
 - Цикл while може його взагалі не виконувати, якщо умова буде хибною відразу.



```
Консоль отладки Microsoft Visual Studio
while:
k = 0
k = 1
k = 2
k = 3
k = 4
do-while:
k = 5
Циклічний ввід:
7
Ви ввели число 7
9
Ви ввели число 9
q
```

```

Console.OutputEncoding = Encoding.UTF8;
Console.WriteLine("break та вкладені цикли:");
for (int i = 1; i <= 7; i++) {
    for (int j = 1; j <= 7; j++) {
        int product = i * j;

        if (product >= 20) {
            Console.WriteLine("\nОбчислення закінчені.");
            break;
        }
        Console.Write("{0}\t", product);
    }
    Console.Write("\n");
}

```

```

Console.WriteLine("goto та вкладені цикли:");
for (int i = 1; i <= 7; i++) {
    for (int j = 1; j <= 7; j++) {
        int product = i * j;

        if (product >= 20) {
            goto LoopEnd;
        }
        Console.Write("{0}\t", product);
    }
    Console.Write("\n");
}

```

```

LoopEnd:
    Console.WriteLine("\nОбчислення закінчені.");

```

01.09.2020

Оператори переходу (jump statements) та вкладені цикли

```

Консоль отладки Microsoft Visual Studio
break та вкладені цикли:
1      2      3      4      5      6      7
2      4      6      8      10     12     14
3      6      9      12     15     18
Обчислення закінчені.

4      8      12     16
Обчислення закінчені.

5      10     15
Обчислення закінчені.

6      12     18
Обчислення закінчені.

7      14
Обчислення закінчені.

goto та вкладені цикли:
1      2      3      4      5      6      7
2      4      6      8      10     12     14
3      6      9      12     15     18
Обчислення закінчені.

```

Винятки та оператор throw

- **Виняток** – це будь-який стан помилки або непередбачена поведінка, що виникає при виконанні програми.
 - Можливі ситуації: спроба підключення до вже не існуючої БД, ділення на 0, відкриття пошкодженого файлу тощо.
 - У таких випадках система перехоплює (catches) помилку та генерує виняток (raises an exception).
- Процес генерування помилки та сигналювання щодо неї називають **викиданням винятку** (*throwing exception*).
 - Здійснюється за допомогою ключового слова throw, після якого йде екземпляр винятку (класу, породженого від System.Exception):

throw e;

Обробка (handling) винятків. Оператор try-catch

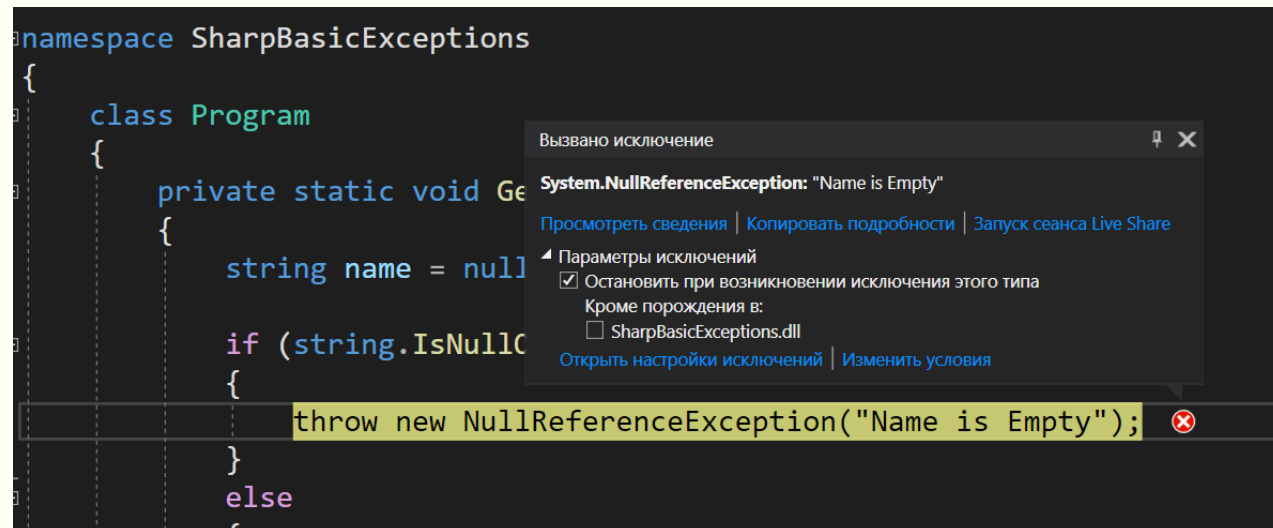
```
static void GetDetails()
{
    string name = null;

    if (string.IsNullOrEmpty(name))
    {
        throw new NullReferenceException("Name is Empty");
    }
    else
    {
        Console.WriteLine("Name: " + name);
    }
}
```

```
static void Main(string[] args)
{
    try
    {
        GetDetails();
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
    }

    Console.ReadLine();
}
```

- Код, який в перспективі може викинути винятки, зазвичай огортають оператором try-catch.
- При виникненні винятку середовище CLR шукає оператор catch, який обробить цей виняток.
- Якщо метод, що в цей момент виконується, не містить такий блок catch, середовище CLR виконує пошук у методі, який поточний метод викликав, і так далі вгору по стеку викликів.
- Якщо блок catch не вдається знайти, CLR відображає користувачу повідомлення про необроблений виняток та зупиняє виконання програми.



Фільтри винятків та блок finally

```
WebClient wc = null;
try
{
    wc = new WebClient(); // завантаження веб-сторінки
    var resultData = wc.DownloadString("http://google.com");
    Console.WriteLine(resultData);
}
catch (WebException ex) when (ex.Status == WebExceptionStatus.ProtocolError)
{
    // код спеціально для WebException ProtocolError
}
catch (WebException ex) when ((ex.Response as HttpWebResponse)?.StatusCode ==
                                HttpStatusCode.NotFound)
{
    // код окремо для WebException NotFound
}
catch (WebException ex) when ((ex.Response as HttpWebResponse)?.StatusCode ==
                                HttpStatusCode.InternalServerError)
{
    // код конкретно для WebException InternalServerError
}
finally
{
    // викликається незалежно від викидання винятку
    wc?.Dispose();
}
```

- Починаючи з C# 6.0, за допомогою ключового слова `when` можна визначити умову виконання обробника конкретного винятку – **фільтр винятку (*exception filter*)**.
 - У прикладі – обробка HTTP-відповіді залежно від її коду.
 - Перелічення [WebException.Status](#).
- Блок `finally` використовується для виконання набору інструкцій незалежно від того, чи викидався виняток.
 - Наприклад, закриття відкритого файлу, з'єднання з БД тощо.

Потреба у блоці `finally`. Оператор `try-finally`

- Блок `finally` потрібний лише тоді, коли після завершення блоку `try` необхідно негайно виконати певні дії незалежно від викидання винятку.
 - Єдина причина – вивільнення ресурсів (resource disposal).
 - У мовах без збирача сміття (C++) без `finally` вчасно не здійсниться деалокація, що призведе до витоків пам'яті.
 - У мовах із збирачем сміття він НЕ дає гарантії, що об'єкт буде знищено після настання його недоступності, тому `finally` все ще корисний.
- Вивільнення ресурсів у .NET забезпечують 2 можливості платформи:
 - Фіналізатори (Finalizers) – методи на зразок деструкторів у C++, автоматично викликаються збирачем сміття в проміжку часу між настанням неможливості об'єкта та моментом його знищення.
 - Інтерфейс `IDisposable` – дозволяє розробнику описати процес вивільнення всіх ресурсів об'єкта.
- Обидві можливості прив'язані одна до одної, тому при реалізації фіналізатора завжди очікується реалізація інтерфейсу `IDisposable`.

Методи та оператор return

```
static double CalculateArea(int r)
{
    return r * r * Math.PI;
}

static void Main()
{
    int radius = 5;
    double result = CalculateArea(radius);
    Console.WriteLine("Площа дорівнює {0:0.00}",
                      result);

    Console.ReadKey();
}

public ref Person GetContactInformation(
    string fname, string lname) {
    // ...реалізація методу...
    return ref p;
}
```

- Оператор return перериває виконання методу, в якому він знаходиться, та повертає управління викликаючому методу.
 - Опційно може повертати значення.
 - Якщо тип повернення – void, оператор return можна пропустити.
 - Якщо оператор return знаходиться всередині блоку try, то блок finally, за умови його існування, буде виконано перед поверненням управління викликаючому методу.
- Починаючи з C# 7.0 підтримується **повернення значень за посиланням (reference return values, ref returns)**.
 - Повернення посилання (або псевдоніму) на деяку змінну означає, що викликаючий метод може бачити зміну значення цієї змінної у викликаному методі.
 - Детальніше в наступній темі.



ДЯКУЮ ЗА УВАГУ!

Наступне питання: Математичні обчислення та клас Math