Лекція 2

# ОСНОВИ МОВИ ПРОГРАМУВАННЯ C#

# Лекція 2. Основи мови програмування C#

## План

1. Огляд мови C#
2. Типи даних
3. Консольний ввід-вивід
4. Основні операції
5. Оператори мови C#

# 1. Огляд мови C#

**C#** – сучасна універсальна високорівнева об'єктно-орієнтована мова програмування, створена у **1998-2001** рр. компанією Microsoft як засіб розробки додатків для платформи Microsoft .NET Framework.

C# відноситься до сімейства мов з **C**-подібним синтаксисом. На його розробників значний вплив зробили мови **Java** та **C++** . Він підтримує такі можливості як:

- інкапсуляцію;
- успадкування;
- поліморфізм;
- перевантаження операторів;
- статичну типізацію;
- ітератори;
- коментарі у форматі XML;
- та багато іншого.

# 1. Огляд мови C#

**Основні переваги мови C#:**

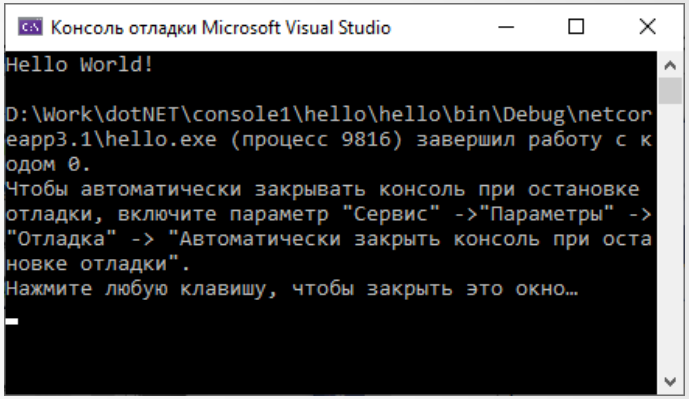
- підтримка більшості продуктів Microsoft;
- фіксований розмір базових типів даних (**int** – 32-біта; **long –** 64), що покращує можливості перенесення програм на цій мові;
- автоматичне «прибирання сміття» (звільнення пам'яті);
- низький «поріг входження» (синтаксис C# вважається найбільш зрозумілим та придатним для новачків);
- велика кількість вакансій на посаду C#-програміста.

**Недоліки C#:**

- пріоритетна орієнтованість на роботу в середовищі Windows (хоча зараз активно розвиваються проекти з портування .NET Framework на інші платформи);
- використання мови безкоштовно тільки у певних випадках (для невеликих фірм, індивідуальних програмістів, стартапів та учнів).

# 1. Огляд мови C#

**Простий консольний додаток на C# буде мати такий вигляд:**




```
using System;

namespace hello
{
    // Клас запуску програми
    class Program
    {
        // метод Main є точкою входу до програми
        static void Main(string[] args)
        {
            // Виведення текстового повідомлення на екран
            Console.WriteLine("Hello World!");
        }
    }
}
```

**Примітка: як і C++, мова C# є реєстрозалежною.**

# 1. Огляд мови C#

**У C# є три типи коментарів:**

- однорядковий;
- багаторядковий;
- документуючий XML-коментар.

**Однорядковий коментар** може починатися у будь-якій позиції рядка програмного коду. Він починається з двох символів "//" і триває до кінця рядка.
Наприклад:

```
// Це однорядковий коментар
Console.WriteLine("Hello World!"); // Це теж однорядковий коментар
```

**Багаторядковий коментар** може складатися із довільної кількості рядків. Він починається із двох символів «/*» і закінчується – «*/».
Наприклад:

```
/*
   Це багаторядковий
   коментар.
   Він складається з п'яти рядків.
*/
```

**Примітка: Багатострокові коментарі не можуть бути вкладені!**

# 1. Огляд мови C#

**Документуючий XML-коментар** починається з "///". Він використовується для автоматичне створення XML-документації. В ньому використовуються спеціальні теги, наприклад, <para> (структурування тексту), <summary> (описи якогось елемента) та ін. Наприклад:

```
using System;
/// <summary>
/// Програма "Hello World!"
/// </summary>
namespace hello
{
   /// <summary>
   /// Клас запуску програми
   /// </summary>
   class Program
   {
        /// <summary>
        /// Метод Main є точкою входу до програми
        /// <para>Він має аргумент - параметри командного рядка</para>
        /// </summary>
        static void Main(string[] args)
        {
              Console.WriteLine("Hello World!");
        }
   }
}
```

## 2. Типи даних

C# є **строго типізованою мовою**. Тобто компілятор і середовище часу виконання CLR суворо стежать за відповідністю типів даних у виразах. **Кожна змінна перед використанням має бути оголошена** . Синтаксис опису змінної має такий формат:

<тип> <ім'я_змінної> [<ініціалізація>][, …];

Тобто спочатку вказується тип змінної, потім її ім'я. При необхідності змінній при її оголошенні може бути присвоєне деяке початкове значення.

Всі типи даних C# можна умовно розділити на:

- **прості**;
- **структурні**;
- **посилальні**;
- **перерахування**.

До простих типів даних відносяться **логічні** та **числові** типи.
**Логічний** (**бульовий**) тип позначається ключовим словом **bool**. Він може приймати одне з двох можливих значень **true** або **false** .
Наприклад:

bool isFree = false;

## 2. Типи даних

Числові типи даних C# діляться на **цілочисленні** і типи з **рухомою комою** (**дійсні**). До цілих відносяться **sbyte**, **byte**, **short**, **ushort**, **int**, **uint**, **long**, **ulong** і **char**. У табл. 1 наведено їх характеристики.

**Таблиця 1 – Характеристики цілих типів даних C#**

| Тип даних | Розмір (у розрядах) | Діапазон допустимих значень |
|---|---|---|
| sbyte | 8- бітне ціле зі знаком | -128 – +127 |
| byte | 8- бітне ціле без знака | 0 – 255 |
| char | 16-бітний символ Unicode | 0000 – FFFF |
| short | 16- бітне ціле зі знаком | -32768 – +32767 |
| ushort | 16- бітне ціле без знака | 0 – 65535 |
| int | 32- бітне ціле зі знаком | -2147483648 – +2147483647 |
| uint | 32- бітне ціле без знака | 0 – 4294967295 |
| long | 64- бітне ціле зі знаком | -9223372036854775808 – +9223372036854775807 |
| ulong | 64- бітне ціле без знака | 0 – 18446744073709551615 |

# 2. Типи даних

Тип даних **char** призначений для збереження одного символу у форматі Unicode. Наприклад:

```
char b = 'H';
char c = 'C';
char d = 'x000D';
```

**Таблиця 2 – Допустимі керуючі послідовності C#**

| Керуюча послідовність | Опис |
|---|---|
| \' | Одинарна лапка |
| \" | Подвійна лапка |
| \\ | Зворотний слеш |
| \0 | Нуль (Null) |
| \ a | Звуковий сигнал |
| \ b | Стирання символу зліва (Back Space) |
| \f | Подання сторінки (Form Feed) |
| \ n | Переклад каретки або пропуск рядка |
| \r | Повернення каретки |
| \ t | Горизонтальна табуляція |
| \ v | Вертикальна табуляція |

## 2. Типи даних

Дійсні типи даних у C# це **float**, **double** і **decimal**.

Тип **float** може набувати значення в діапазоні від $\pm 1.5 \cdot 10^{-45}$ до $\pm 3.4 \cdot 10^{38}$ (**точність типу float становить до 9 знаків**). Значення типу float може бути написане як у звичайній формі, так і в експоненційній. Наприклад:

```
float a = 10;
float b = 10.5;
float c = 1.0E-10;
```

Тип **double** може приймати значення в діапазоні від $\pm 5.0 \cdot 10^{-324}$ до $\pm 1.7 \cdot 10^{308}$ (**точність типу double становить до 17 знаків**). Наприклад:

```
double a = 0.000000001234567;
```

Тип даних **decimal** може набувати значення в діапазоні від $\pm 1.0 \cdot 10^{-28}$ до $\pm 7.9 \cdot 10^{28}$ (**точність типу decimal становить до 29 знаків**). Наприклад:

```
decimal a = 0.00000000000000000001234567;
```

# 2. Типи даних

**Структурні типи** даних C# призначені для зберігання наборів різнотипних елементів. Оголошення структури здійснюється за допомогою ключового слова **struct**.

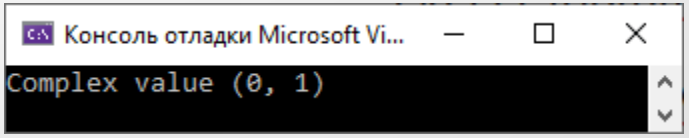
Приклад програми, що реалізує оголошення та роботу зі структурою даних, що описує комплексні числа, може мати такий вигляд:

```
using System;

namespace hello
{
   public struct Complex
   {
      public float re, im;
      public Complex(float r, float i)
      {
         re = r;
         im = i;
      }
}
```

```
class Program
{
   static void Main(string[] args)
   {
      Complex i = new Complex(0, 1);

      Console.WriteLine("Complex value ({0}, {1})",
            i.re, i.im);
   }
}
}
```



## 2. Типи даних

**Посилальні типи** в C# – це типи даних, які розміщуються в "**купі**" (heap), тобто вони використовують додаткові ресурси системи. Такі типи керуються вбудованим в CLR "прибиральником сміття" (**garbage collection**).

У мові C# є чотири типи посилальних типів:

- **класи** – типи даних, створені програмістами;
- **інтерфейси** – засоби мови, які забезпечують взаємодію з відкритими атрибутами і поведінкою класів;
- **делегати** – дозволяють безпечно динамічно звертатися до методів класів;
- **масиви** – призначені для зберігання множини однотипних елементів.

**Рядковий тип** даних C# представляється у вигляді послідовностей символів Unicode, взятих у подвійні лапки. Вони можуть включати керуючі послідовності. Наприклад:

```
string a = "Hello \n World!"; // Рядковий літерал з керуючою послідовністю
string b = @"Літерний
             рядковий
             літерал" / / Буквальний рядковий літерал (упр. посл. не обробляються)
```

**Примітка: на відміну від C++ C# рядковий тип є вбудованим.**

## 2. Типи даних

**Перелічувані типи** в C# призначені для завдання списків константних значень, які можуть належати до типів **byte**, **int**, **long** та **short**. Наприклад:

```
enum Colors {Red, Green, Blue}; // Red = 0, Green = 1, Blue = 2
enum Sex: short {Male = 1, Female}; // Константи відносяться до типу short, значення
                                    // починаються з 1
```

На практиці часто виникає потреба у перетворенні одного типу даних на інший. Існують два види **перетворень типів даних**:

- **неявні перетворення** – відбуваються автоматично;
- **явні перетворення** – виконуються з допомогою операції приведення типу (**cast**), що використовується у випадках, коли може статися втрата точності чи помилка.

Наприклад:

```
float a = 1.5f;
short b = 1;
int c = b; // неявне перетворення типу
int d = (int) a; // явне перетворення дійсного типу до цілого із втратою точності
```

## 2. Типи даних

**Масив** в C# – це об'єкт для зберігання множини однотипних даних з вбудованими функціями різних операцій. Усі масиви можна розділити на **одномірні**, **багатовимірні** та **вільні**.

**Одновимірні масиви** (вектори) оголошуються, наприклад, таким чином:

```
int[] arr; // Оголошення порожнього цілісного масиву без ініціалізації
float[] arr1 = new float[5] {1, 2, 3, 4, 5.5f}; // Оголошення масиву з ініціалізацією
double[] arr2 = {1.0, 2.0, 3.0}; // Ще один варіант оголошення з ініціалізацією
```

**Багатовимірні масиви** оголошуються, наприклад, таким чином:

```
int[,,] arr = new int[2, 2, 3]; // Оголошення цілого масиву 2 x 2 x 3 без ініціалізації
float[,] arr1 = {{1.5f, 2}, {2, 3}};; // Оголошення речовинного масиву 2 x 2 з ініціалізацією
```

**Вільні масиви** – це багатовимірні масиви, що мають неоднакову довжину різних вимірів:

```
Int [] [] Arr = new int [] [];

Arr[0] = new int[4] {1, 2, 3, 4};
Arr[1] = new int[2] {5, 6};
```

# 2. Типи даних

**Операції над масивами**

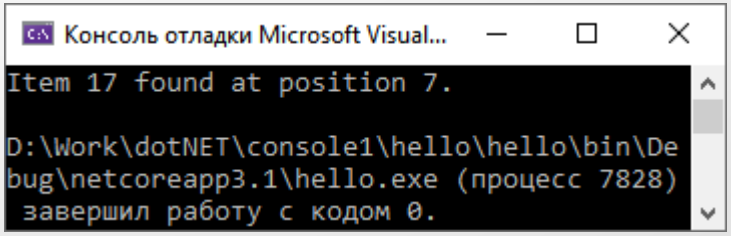
У для роботи з масивами передбачені різні властивості і методи, що дозволяють виконувати різні операції над його елементами. Розглянемо деякі з них.

Для **пошуку заданого значення** одномірному масиві використовується метод **BinarySearch()** .

Наприклад:

```
int[] intArray = { 1, 3, 5, 7, 11, 13, 17, 19, 23, 31};
int target = 17 // Value to search for
pos;

if ((pos = Array.BinarySearch(intArray, target)) >= 0)
Console.WriteLine("Item {0} found at position {1}.", intArray[pos].ToString(), pos + 1);
else
Console.WriteLine("Item not found");
```



# 2. Типи даних

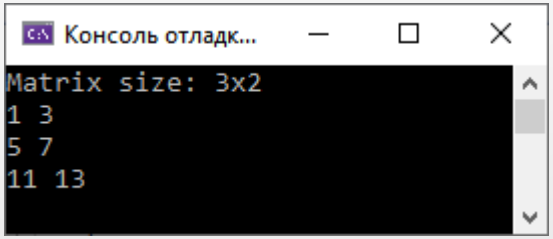
**Операції над масивами (продовження)**

Для отримання **кількості елементів** у масиві у заданому напрямку використовується метод **GetLengh()**.

Наприклад:

```
int[,] matrix = {{ 1, 3 }, { 5, 7 }, { 11, 13 }};

Console.WriteLine("Matrix size: {0}x{1}", matrix.GetLength(0), matrix.GetLength(1));
for (int i = 0; i < matrix.GetLength(0); i++)
{
    for (int j = 0; j < matrix.GetLength(1); j++)
        Console.Write("{0}", matrix[i, j]);
    Console.WriteLine();
}
```



# 2. Типи даних

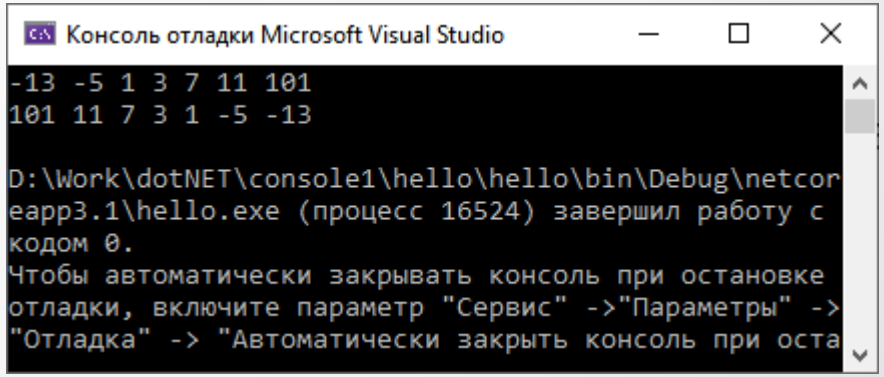
**Операції над масивами (продовження)**

Для сортування елементів одновимірного масиву використовуються методи **Sort()** та **Reverse()** .

Наприклад:

```
int[] vector = { 1, 3, -5, 101, 7, 11, -13};

Array.Sort(vector);
for (int i = 0; i < vector.Length; i++)
    Console.Write("{0} ", vector[i]);
Console.WriteLine();
Array.Reverse(vector);
for (int i = 0; i < vector.Length; i++)
    Console.Write("{0} ", vector[i]);
Console.WriteLine();
```



# 3. Консольний ввід-вивід

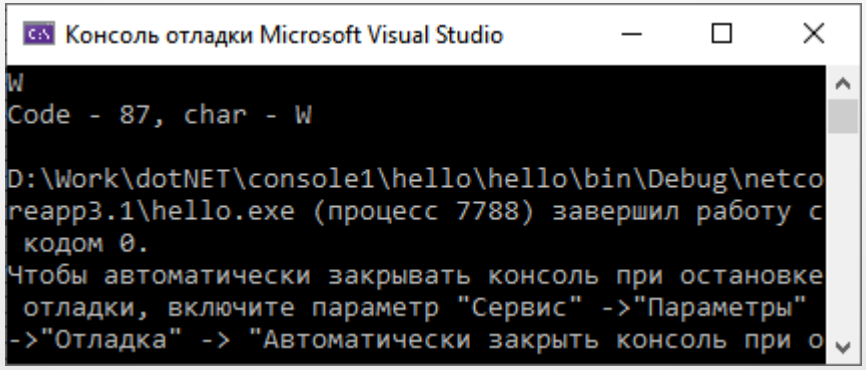
Для консольного вводу-виводу в C# найчастіше використовуються методи **Read()**, **ReadLine()**, **Write()** і **WriteLine()** .

Зчитування чергового символу з вхідного потоку здійснюється за допомогою функції **Read()**, що повертає ціле значення (**int**) – код символу.

Наприклад:

```
int ch;

ch = Console.Read();
Console.WriteLine("Code – {0}, char – {1}", ch, (char)ch);
```

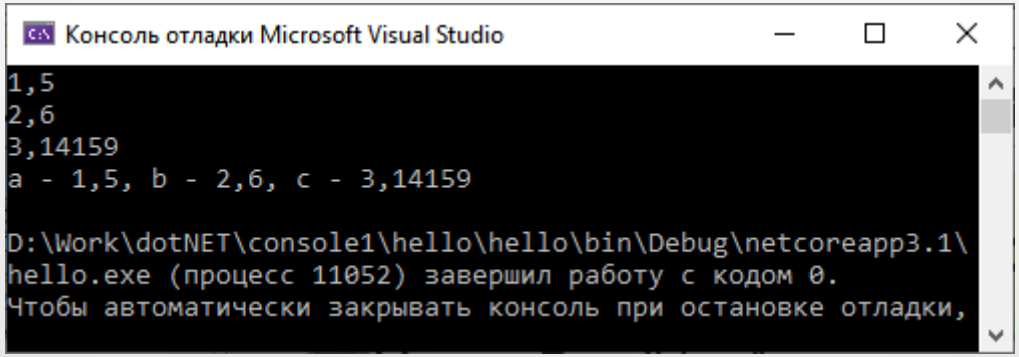


# 3. Консольне введення-виведення

Метод **ReadLine()** здійснює зчитування чергового рядка символів із вхідного потоку. Він повертає тип **string**. Наприклад:

```
double a, b, c;

a = Convert.ToDouble(Console.ReadLine());
b = Convert.ToDouble(Console.ReadLine());
c = Convert.ToDouble(Console.ReadLine());
Console.WriteLine("a – {0}, b – {1}, c – {2}", a, b, c);
```

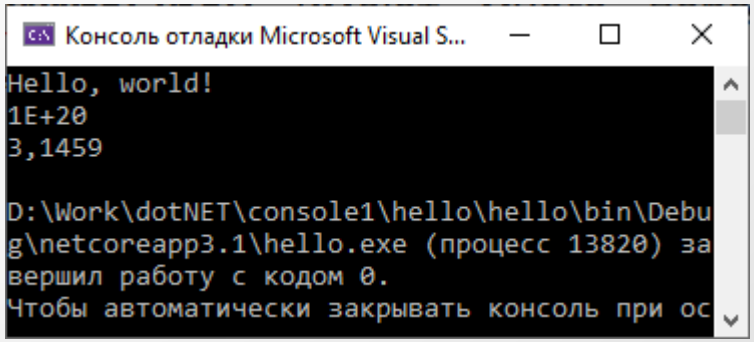


# 3. Консольний ввід-вивід

Методи **Write()** та **WriteLine()** реалізують запис даних у потік. Виклик першого методу не призводить до переведення каретки на новий рядок.

Методи **Write()** і **WriteLine()** мають багато навантажень для виведення цілих і речових чисел, рядків тощо. Наприклад:

```
Console.Write('H');
Console.Write('e');
Console.Write('l');
Console.Write('l');
Console.Write('o');
Console.WriteLine(", world!");
Console.WriteLine(1.0E+20);
Console.WriteLine(3.1459);
```



# 4. Основні операції

Під **операцією** зазвичай розуміється деяка дія над даними. Дані, що використовуються для виконання операції, називаються **операндами**. Операндами можуть бути константи, змінні чи виклики методів (функцій). Результатом виконання операцій завжди є якесь значення.

У мові C# існує велика кількість різноманітних операцій, які за кількістю операндів поділяються на **унарні**, **бінарні**, **тернарні** та інші (що не належать до жодного з перших трьох типів).

Наприклад:

**-**a; // Унарна операція зміна знака операнда
a **+** b; // Бінарна операція додавання
int a = 1, b = 2, **c = a > b? 3: -3** ; // Тернарна операція (повертає 3, якщо
// умова a > b істинна, або -3 – якщо навпаки)

# 4. Основні операції

На практиці найчастіше застосовуються **арифметичні операції**

**Таблиця 3 – Арифметичні операції**

| Знак операції | Призначення | Приклад використання | Результат |
|---|---|---|---|
| + | додавання | 2.5 + 5 | 7.5 |
| - | віднімання | 4 – 1 | 3 |
| * | множення | 3*5 | 15 |
| / | ділення | 2.4/2 | 1.2 |
| % | обчислення залишку при діленні цілих чисел | 5% 2 | 1 |
| ++ | інкремент | a++ | a + 1 |
| -- | декремент | --a | a - 1 |

# 4. Основні операції

**Операції порівняння** та **логічні операції** застосовуються в логічних виразах

**Таблиця 4 – Операції порівняння**

| Знак операції | Призначення |
|---|---|
| < | менше |
| <= | менше або дорівнює |
| == | дорівнює |
| != | не дорівнює |
| > | більше |
| >= | більше або дорівнює |

**Таблиця 5 – Логічні операції**

| Знак операції | Назва | приклад |
|---|---|---|
| ! | Логічне заперечення (**not**) | !a |
| && | Логічне множення (**and**) | a && b |
| \|\| | Логічне додавання (**or**) | a \|\| b |

# 5. Оператори мови C#

**Оператори** – це команди процесору на виконання певних дій. Їх можна умовно поділити на дві групи:

- **прості** (не змінюють ходу виконання програми);
- **структуровані** (змінюють порядок виконання програми).

Прикладом простого оператора є присвоювання. До структурованих можна віднести такі оператори:

- **переход**;
- **розгалуження** (умовний оператор);
- **цикл**.

**Оператор переходу** (безумовного переходу) здійснює перехід від поточного місця виконання інше місце. Наприклад:

```
goto label1;
Console.WriteLine("Hello");
label1: Console.WriteLine("world!");
```

**Примітка: використання оператора безумовного переходу вважається поганим стилем програмування!**

# 5. Оператори мови C#

У C# (як і в мовах C/C++) є поняття **блочного оператора** (або **блоку**), під яким розуміється довільний набір операторів, укладених у фігурні дужки:

```
{
  <оператор_1>
  <оператор_2>
  …
  <оператор_N>
}
```

На логічному рівні блоковий оператор сприймається як один оператор.

Спеціальним типом оператора є **порожній оператор** ";". Його зазвичай застосовують там, де згідно із синтаксисом потрібен оператор, але жодних дій не потрібно виконувати. Наприклад:

```
// Довічний цикл :)
while (1)
    ;
```

# 5. Оператори мови C#

**Оператор розгалуження if** в залежності від умови дозволяє вибрати один із кількох можливих варіантів виконання програми. Він має кілька форм:

1) if (<логічний_вираз>)
    <оператор>;

Наприклад:

```
if (a > b and b > c)
{
  a = 1;
  Console.WriteLine("{0}, {1}, {2}", a, b, c);
}
```

2) if (<логічний_вираз>)
    <оператор_1>;
  else
    <оператор_2>;

Наприклад:

```
if (a > b)
  Console.WriteLine("{0} > {1}", a, b);
else
  Console.WriteLine("{0} <= {1}", a, b);
```

3) if (<логічний_вираз_1>)
    <оператор_1>;
  else if (<логічний_вираз_2>)
    <оператор_2>;
  else if (<логічний_вираз_3>)
    …
  else
    <оператор_N>;

Наприклад:

```
if (a < 0)
   Console.WriteLine("{0} < 0", a);
else if (a == 0)
   Console.WriteLine("{0} == 0", a);
else
   Console.WriteLine("{0} > 0", a);
```

# 5. Оператори мови C#

**Оператор вибору** є варіантом умовного оператора. Він має таку форму:

```
switch (<вираз>)
{
   case <константа_1>:
      <оператор_1>;
      break;
   case <константа_2>:
     <оператор_2>;
     break;
   ...
   case <константа_N>:
     <оператор_N>;
     break;
   default:
     <оператор_за замовчуванням>;
     break;
}
```

Наприклад:

```
switch (n)
{
   case 1:
      Console.WriteLine("Понеділок");
      break;
   case 2:
      Console.WriteLine("Вівторок");
      break;
   ...
   default:
      Console.WriteLine("Помилка");
      break;
}
```

# 5. Оператори мови C#

**Оператори циклу** реалізують повторне виконання деякого оператора чи групи операторів. Вони бувають чотирьох видів:

- **while** (цикл із передумовою);
- **do...while** (цикл із постумовою);
- **for** (параметричний цикл);
- **foreach** (ітерування за колекцією)

Цикл **while** застосовується тоді, коли заздалегідь невідомо кількість ітерацій. Його синтаксис наступний:

```
while (<умова>)
   <оператор>;
```

Наприклад:

```
while(i <= n)
{
   ...
   i++;
}
```

**Примітка. Залежно від умови тіло циклу while може не виконатися жодного разу.**

# 5. Оператори мови C#

Цикл з постумовою **do...while** застосовується тоді, коли бажано, щоб тіло циклу виконалося хоча б один раз. Його синтаксис наступний:

```
do
   <оператор>;
while (<умова>);
```

Наприклад:

```
do
{
   ...
   i--;
}
while (i >= 0);
```

# 5. Оператори мови C#

Параметричний цикл **for** застосовується у тому випадку, коли наперед відома кількість ітерацій. Його синтаксис наступний:

```
for (<ініціалізація>; <умова>; <приріст>)
    <оператор>;
```

Наприклад:

```
for (int i = 0; i < 10; i + = 2)
    Console.WriteLine(i);
```

Цикл **foreach** застосовується для ітерування за колекціями:

```
foreach (<тип> <ідентифікатор> in <колекція>)
    <оператор>;
```

Наприклад:

```
double[] arr = { 1.0, 2.0, 3.0, 4.0, 5.6};
foreach (double it_var in arr)
    Console.WriteLine(it_var);
```