ЖИТОМИРСЬКА ПОЛІТЕХНІКА
ДЕРЖАВНИЙ УНІВЕРСИТЕТ

C#. Колекції

Лекція 04

План

1. Що таке колекції?
2. Колекції в .NET
3. List<T>
4. Створення та ініціалізація
5. Основні операції
6. Практичні приклади

Що таке колекції?

Для зберігання невеликих наборів однотипних даних у C# використовуються масиви. Однак, оскільки вони зберігають фіксовану кількість елементів, то на практиці їх застосування не завжди є зручним.

Для зберігання не обов'язково однорідних даних і можливо заздалегідь невизначеної кількості зручніше застосовувати колекції.

Колекція - дозволяє зберігати та організовувати набір елементів одного або різних типів. Це як контейнер, в якому можна зібрати різні предмети і потім легко до них отримати доступ, додавати нові або видаляти старі.

Що таке колекції?

У C# класи колекцій організовані за просторами імен, кожне з яких має свою специфікацію:

System.Collections – містить прості, неузагальнені класи колекцій;
System.Collections.Generic – колекції з підтримкою узагальнення типів;
System.Collections.Specialized – спеціалізовані колекції для конкретних завдань;
System.Collections.Concurrent – класи колекцій для ефективної роботи з багатопотоковими додатками.

Колекції в .NET

У C# колекції поділяються на дві основні групи:

1. Неузагальнені колекції (non-generic collections)
   - Походять з простору імен System.Collections.
   - Можуть зберігати об'єкти будь-якого типу, оскільки вони базуються на типі object.
   - Вважаються застарілими та менш ефективними порівняно з узагальненими колекціями.

Представники:
ArrayList: динамічний масив, що може змінювати свій розмір.
Hashtable: колекція пар "ключ-значення".
Queue: черга.
Stack: стек.

Колекції в .NET

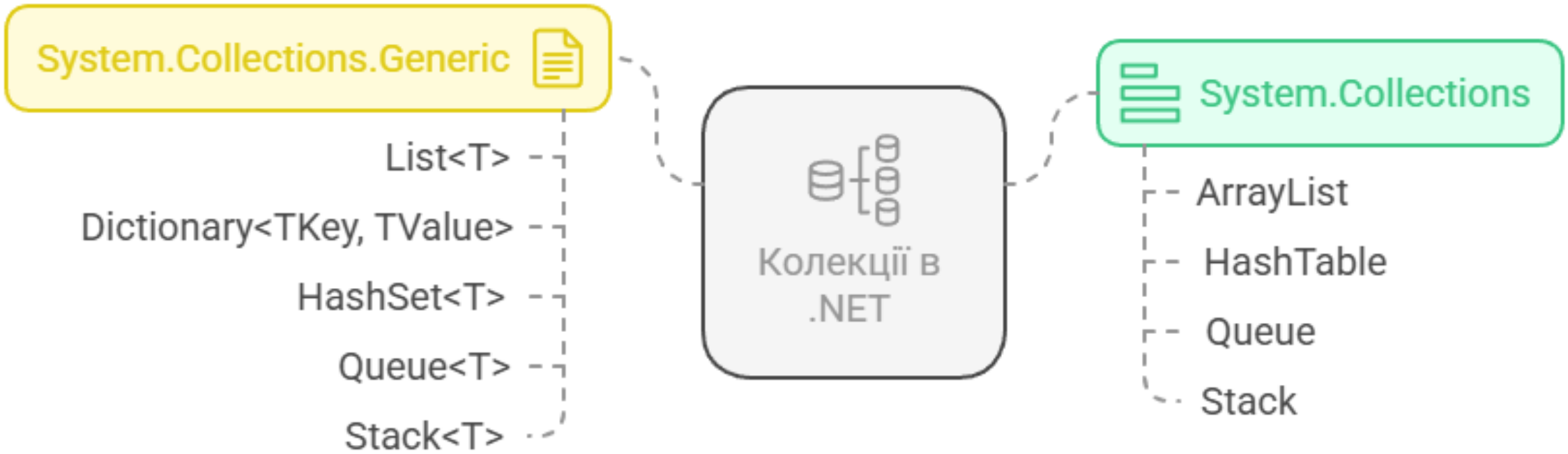
2. Узагальнені колекції (generic collections)

- Знаходяться в просторі імен System.Collections.Generic.
- Дозволяють створювати колекції, що зберігають елементи конкретного типу.
- Базуються на узагальненнях (generics), що забезпечує безпеку типу на етапі компіляції.
- Рекомендовані для використання в сучасних розробках на C#.

Представники:

List<T> - список елементів типу T.
Dictionary<TKey, TValue> - словник пар "ключ-значення".
Queue<T> - черга елементів типу T.
Stack<T> - стек елементів типу T.

Колекції в .NET



Приклад

```
// Неузагальнена колекція ArrayList
ArrayList objList = new ArrayList() { 1, 2, "Hello world!", 'c', 3.14 };

// Отримуємо елементи та явно перетворюємо їх до потрібного типу
  int num = (int)objList[0];
  string text = (string)objList[2];
  double p = (double)objList[4];
  Console.Write($": {num} : {text} : {p}");

  // Узагальнена колекція List<int>
  List<int> intList = new List<int>();

  // Отримуємо елементи
  intList.Add(1);
  intList.Add(2);
  int number2 = intList[0];
 Console.WriteLine($": {number2}");
```

List<T>

List<T> - це колекція строго типізованих об'єктів, до яких можна отримати доступ за індексом. Має методи для сортування, пошуку та модифікації списку.

Це узагальнена версія ArrayList, що належить до простору імен System.Collections.Generic.

List<T>

List<T> - це один з найпоширеніших і універсальних класів колекцій в C#.

Представляє собою динамічний масив, тобто його розмір може змінюватися під час виконання програми.

Дозволяє зберігати будь-яку кількість елементів певного типу.

Параметр T вказує, які саме елементи можна зберігати в цьому конкретному списку.

Створення та ініціалізація

Синтаксис:

List<T> <list_name> = new List<T>(<size>);

<T> - тип об’єкта, який потрібно створити (int, float, doublei тощо)
<list_name> - назва списку.
< size > - це розмір списку, який є необов’язковим.

Приклад:

```
List<string> myList = new List<string>();
```

Створення та ініціалізація

```
// Створення списку
List<int> numbers = new List<int>();

// Список заданого розміру
List<double> values = new List<double>(10);

// Список з початковими елементами
List<string> names = new List<string> { "<Ana>", "Oleh",
"Felipe" };

foreach (var name in names)
    Console.WriteLine($"Hello {name}!");
```

Основні операції

Додавання об'єкта до List<T>

Метод Add(T obj) додасть елемент типу <T> до списку по одному:

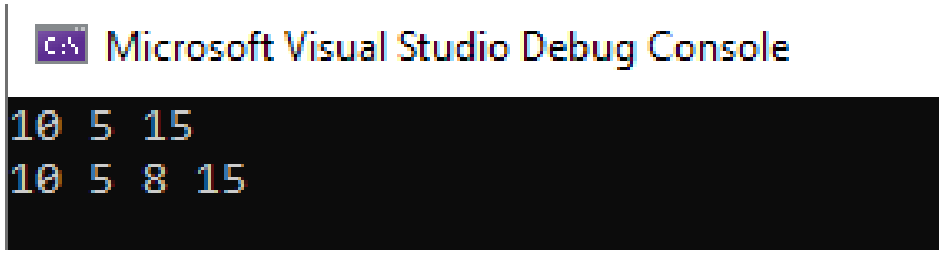
myList.Add(“Element1”);

Метод Insert(int index, T obj) додасть елемент за вказаним індексом.

myList.Insert(1,”Element1”);

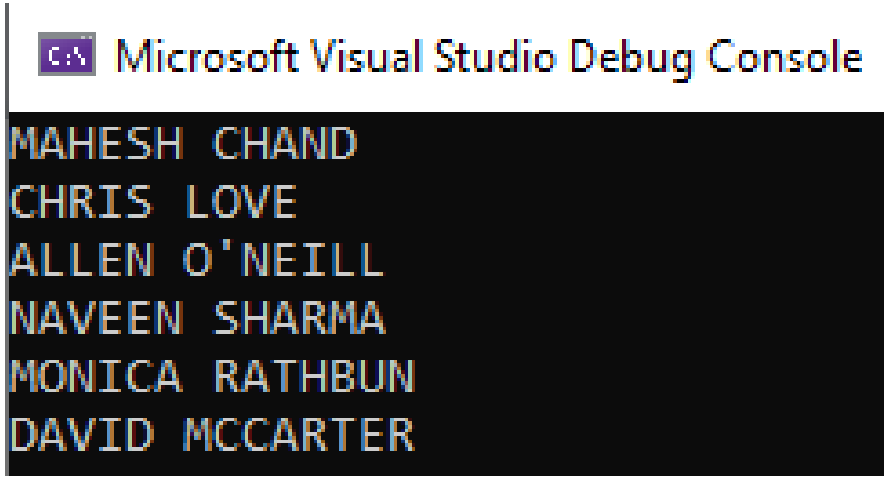
Приклад

```
 List<int> numbers = new List<int>();

// Додаємо елементи
numbers.Add(10);
numbers.Add(5);
numbers.Add(15);

// Виводимо елементи
foreach (int number in numbers)
    Console.Write(number+" ");
Console.WriteLine();

// Додаємо елемент у вказану позицію
numbers.Insert(2,8);
foreach (int number in numbers)
    Console.Write(number + " ");
```



Приклад

```
List<string> authors = new List<string>(5);
authors.Add("Mahesh Chand");
authors.Add("Chris Love");
authors.Add("Allen O'neill");
authors.Add("Naveen Sharma");
authors.Add("Monica Rathbun");
authors.Add("David McCarter");

foreach (var name in authors)
    Console.WriteLine($"{name.ToUpper()}");
```



Основні операції

Видалення об’єкта з List<T>
Метод Remove(T obj) видаляє елемент зі списку (перше входження):

myList.Remove(“Element1”);

Метод RemoveAt(int index) видаляє елемент за вказаним індексом:

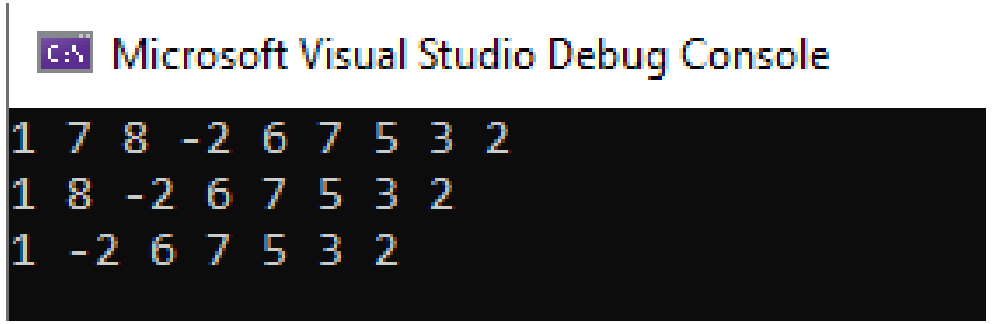
RemoveAt(int index)

Метод RemoveAt(int index) видаляє список елементів (початковий індекс, кількість елементів):

RemoveRange(int index, int index)

Приклад

```
List<int> numbers = new List<int>() {1, 7, 8, -2, 6, 7, 5, 3, 2 };

foreach (int number in numbers)
    Console.Write(number+" ");
Console.WriteLine();

// Видаляє перше входження значення числа
numbers.Remove(7);

foreach (int number in numbers)
    Console.Write(number + " ");

// Видаляє елемент за індексом
numbers.RemoveAt(1);

foreach (int number in numbers)
    Console.Write(number + " ");
```



Основні операції

Отримати розмір List<T>:
Метод Count використовується для отримання розміру списку.

myList.Count();

Метод Sort() використовується для сортування List<T> в порядку зростання.

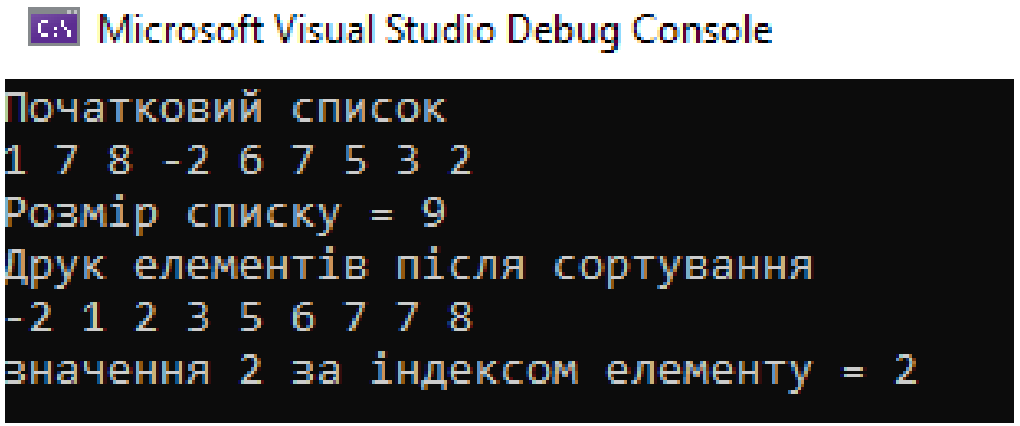
myList.Sort();

Номер індексу можна використовувати для доступу до елементів List<T>, як до масиву.
наприклад:

myList[0]

Приклад

```
List<int> numbers = new List<int>() {1, 7, 8, -2, 6, 7, 5, 3, 2 };
Console.WriteLine("Початковий список");
foreach (int number in numbers)
    Console.Write(number+" ");

Console.WriteLine("\nРозмір списку = " + numbers.Count());

numbers.Sort();
Console.WriteLine("Друк елементів після сортування");
foreach (int number in numbers)
    Console.Write(number + " ");

Console.WriteLine("\nзначення 2
за індексом елементу ="+numbers[2]);
```



Основні операції

Пошук елементів List<T>:
Метод Contains() використовується для перевірки: чи містить список задане значення.

myList.Contains();

Метод IndexOf() повертає індекс першого входження заданого значення у List<T> :

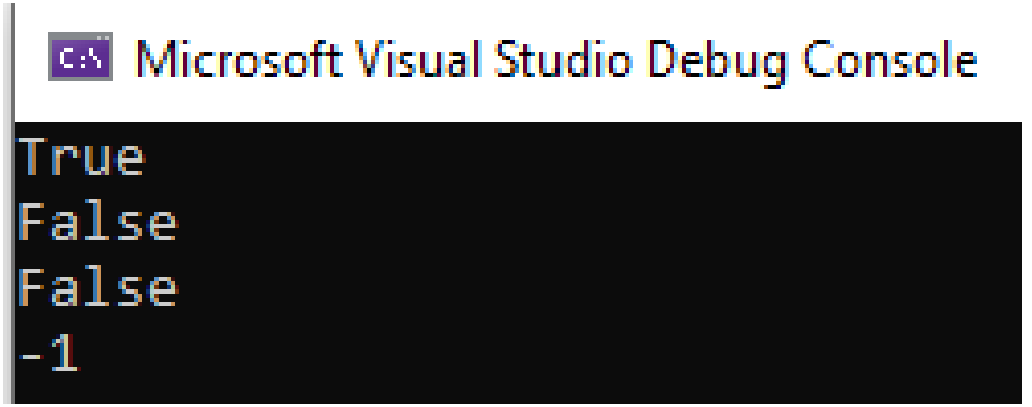
myList.IndexOf();

Очистити всі елементи зі списку

myList. Clear();

Приклад

```
List<int> numbers = new List<int>() { 1, 2, 5, 7, 8, 10 };

Console.WriteLine(numbers.Contains(10));
Console.WriteLine(numbers.Contains(11));
Console.WriteLine(numbers.Contains(20));

Console.WriteLine(numbers.IndexOf(20));
```



Основні операції

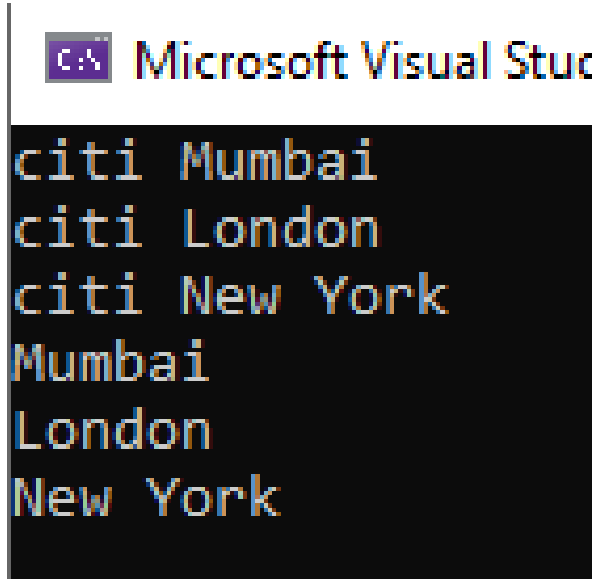
Доступ до списку можна отримати за допомогою індексу, циклу for/foreach і за допомогою запитів LINQ.

```
List<int> numbers = new List<int>() { 1, 2, 5, 7, 8, 10 };
Console.WriteLine(numbers[0]);
Console.WriteLine(numbers[1]);
Console.WriteLine(numbers[2]);
Console.WriteLine(numbers[3]);

for (int i = 0; i < numbers.Count; i++)
    Console.Write(numbers[i]+" ");
```

Основні операції

Використовуйте метод AddRange(), щоб додати всі елементи з масиву або іншої колекції до списку.

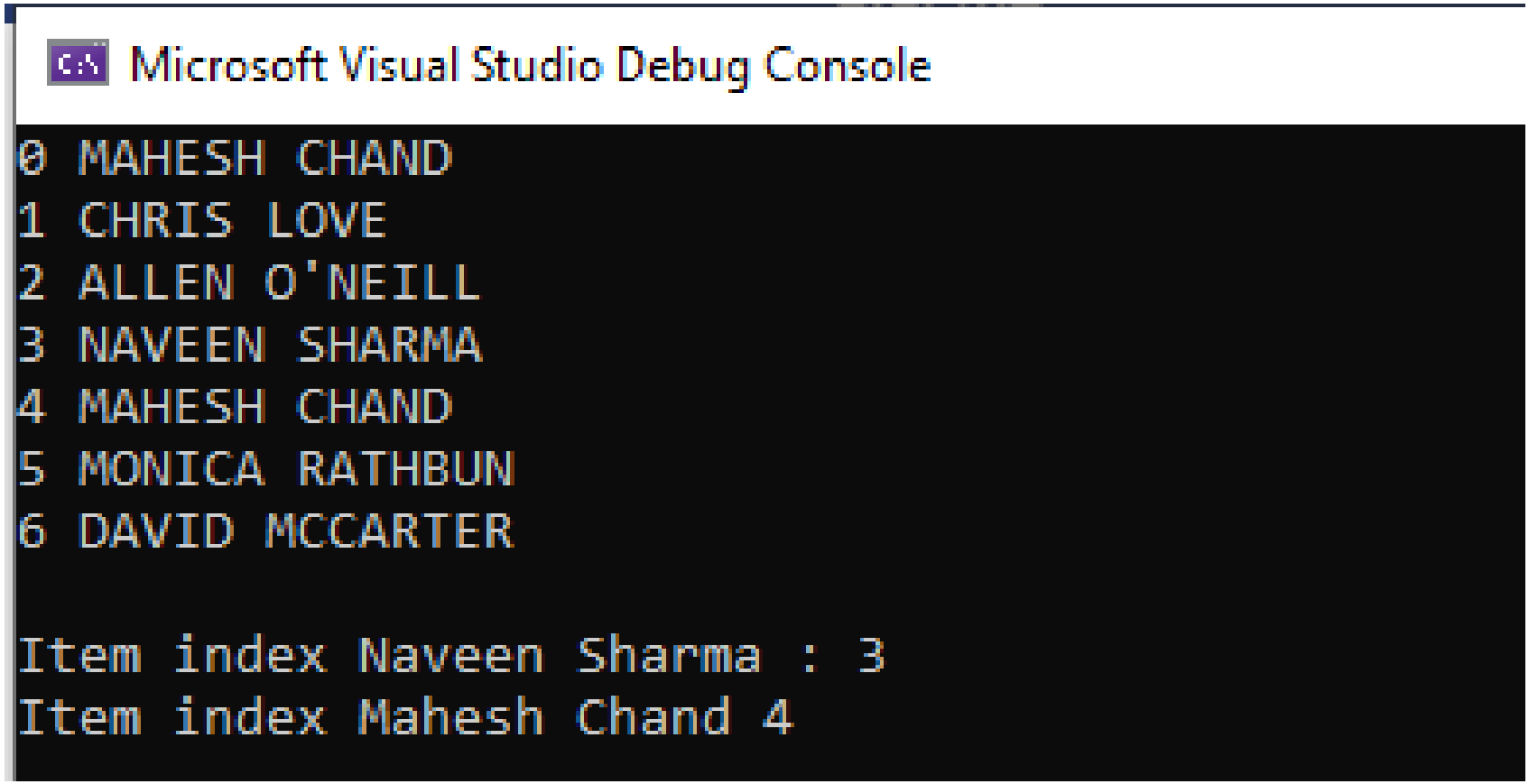
```
string[] citi = new string[3] { "Mumbai", "London", "New York" };
var popularCities = new List<string>();

popularCities.AddRange(citi);
foreach (var p_citi in popularCities)
    Console.WriteLine($"citi {p_citi}");

var favouriteCities = new List<string>();
favouriteCities.AddRange(popularCities);

foreach (var p_citi in favouriteCities)
    Console.WriteLine($"{p_citi}");
```



Приклад

```
 List<string> authors = new List<string>(5);
authors.Add("Mahesh Chand"); authors.Add("Chris Love");
authors.Add("Allen O'neill");authors.Add("Naveen Sharma");
authors.Add("Mahesh Chand"); authors.Add("Monica Rathbun");
authors.Add("David McCarter");
int i = 0;
foreach (var name in authors)
    Console.WriteLine($"{i++} {name.ToUpper()}");

string str = "Naveen Sharma";
int idx = authors.IndexOf(str);
if (idx > 0)
    Console.WriteLine($"\nItem index {str} : {idx}");
else
    Console.WriteLine("Item not found");

Console.WriteLine("Item index Mahesh Chand "+authors.LastIndexOf("Mahesh
Chand"));
```

Приклад


Microsoft Visual Studio Debug Console
0 MAHESH CHAND
1 CHRIS LOVE
2 ALLEN O'NEILL
3 NAVEEN SHARMA
4 MAHESH CHAND
5 MONICA RATHBUN
6 DAVID MCCARTER
Item index Naveen Sharma : 3
Item index Mahesh Chand 4

Лямбда-функції

У C# лямбда-функції надають спосіб для створення анонімних функцій, які можуть бути передані як аргументи іншим функціям або використані для створення делегатів. Вони особливо корисні при роботі з LINQ та іншими методами, що потребують функціональних аргументів.

Лямбда-функція - це анонімна функція, яка може бути визначена без використання ключового слова delegate. Зазвичай вони використовуються для коротких операцій, які зручно визначити безпосередньо в місці їх використання.

Лямбда-функції

Синтаксис лямбда-функцій:

(параметри) => вираз або блок коду

де

(параметри) - список параметрів, які приймає функція;

$\Rightarrow$ - оператор лямбда-виразу, що розділяє параметри та тіло функції;

вираз або блок коду - код, що виконується функцією.

Приклад

Використайте лямбда-вираз для збільшення кожного елемента масиву в 2 рази

```
int[] numbers = { 1, 12, -3, 4, -5 };

numbers = Array.ConvertAll(numbers, x => x * 2);

foreach (int nam in numbers)
    Console.WriteLine(nam);
```

Array.ConvertAll() - метод, який створює новий масив, перетворюючи кожен елемент вихідного масиву за допомогою заданої функції.

x => x * 2 - лямбда-вираз, який приймає один аргумент x (елемент масиву) та повертає його значення, збільшене у 2 рази.

Приклад

Відсортувати масив цілих чисел за спаданням

```
int[] numbers = { 1, 12, -3, 4, -5 };

Array.Sort(numbers, (a, b) => b.CompareTo(a));

foreach (int nam in numbers)
    Console.WriteLine(nam);
```

CompareTo - це метод, який використовується для порівняння двох об'єктів.
Він повертає ціле число, яке вказує на відношення між цими об'єктами:
-1 - перший об'єкт менший за другий;
 1 - перший об'єкт більший за другий;
 0 - об'єкти рівні.

Array.Exists<T>(T[] array, Predicate<T> match)

Перевіряє, чи існує в масиві хоча б один елемент, який задовольняє задану умову.

```
int[] numbers5 = { 1, 2, 3, 4, 5 };

// Перевіряємо, чи є в масиві число більше 3
bool exists = Array.Exists(numbers5, x => x > 3);

Console.WriteLine(exists); // Виведе: True, якщо є числа >  3
```

де
x => x > 3 - лямбда-функція, яка визначає умову пошуку. Це лямбда-функція приймає один аргумент x (елемент масиву) та повертає true, якщо x більше за 3, і false в іншому випадку.

Array.Find<T>(T[] array, Predicate<T> match)

Знаходить перший елемент в масиві, який задовольняє заданій умові.

```
string[] names = { "Іван", "Петро", "Марія", "Ольга", "Михайло" };

// Знаходимо перше ім'я, що починається з літери "М"
string name = Array.Find(names, s => s.StartsWith("М"));

Console.WriteLine(name);
```

де
s => s.StartsWith("M") - лямбда-функція приймає рядок s та перевіряє, чи починається він з літери "М".

match - делегат Predicate<T>, який представляє метод, що визначає умову перевірки для кожного елемента.

=> оператор лямбда-виразу, що розділяє параметри та тіло функції.

Приклад

```
(int x, int y) => x + y;
s => s.Length;
() => Console.WriteLine("Hello, World!");
```

Самостійна робота

1. https://learn.microsoft.com/en-us/dotnet/api/system.collections.generic.list-1?view=netframework-4.7.2
2. https://learn.microsoft.com/en-us/dotnet/api/system.collections.generic.list-1?view=net-5.0#remarks