Лабораторна робота 4. РОБОТА З МАСИВАМИ

Мета: набути умінь і навичок роботи з масивами мовою C#.
Призначення: засвоєння технології ініціалізації, створення та оброки масивів на мові C#.

4.1. Масиви в C#

Масив задає спосіб організації даних. ***Масивом*** називають упорядковану сукупність елементів одного типу. Кожен елемент масиву має індекси, що визначають порядок елементів. Число індексів характеризує розмірність масиву. Кожен індекс змінюється в деякому діапазоні **[a, b].** У мові *C#,* як і у багатьох інших мовах, індекси задаються цілочисельним типом. Діапазон **[a, b]** називається граничною парою, **a** - нижньою, ***b*** - верхньою границею індексу. При оголошенні масиву границі задаються виразами. Якщо всі границі задані константними виразами, то число елементів масиву відомо в момент його оголошення і йому може бути виділена пам'ять ще на етапі трансляції. Такі масиви називаються статичними. Якщо ж вирази, що задають границі, залежать від змінних, то такі масиви називаються динамічними, оскільки пам'ять їм може бути відведена тільки динамічно в процесі виконання програми, коли стають відомими значення відповідних змінних. Масиву виділяється неперервна область пам'яті. У C# масиви динамічні. При написанні програм, можна створювати одновимірні. багатовимірні масиви й масиви масивів.

4.1.1. Одновимірні масиви

Оголошення одновимірного масиву виглядає в такни спосіб:

<тип>[] <ім'я масиву>;

Квадратні дужки приписані не до імені змінної, а до типу. Вони є невід'ємною частиною визначення класу, так що, наприклад, запис int [] варто розуміти як клас "одновимірний масив з елементами типу Int".

Що ж стосується границь зміни індексів, то ця характеристика до класу не відноситься, вона є характеристикою змінних.

Як і у випадку оголошення простих змінних, при оголошенні масиву одночасно може бути проведена й ініціалізація Потрібно розуміти, що при оголошенні з відкладеною ініціалізацією сам масив не формується, а створюється тільки посилання на масив, що мас невизначене значення null. Тому поки масив не буде реально створений і його елементи ініціалізовані, використати його в обчисленнях не можна. Приклад оголошення трьох масивів з відкладеною ініціалізацією:

int[ ] a, b, c;

Найчастіше при оголошенні масиву використовується ім'я з ініціалізацією. Як і у випадку простих змінних, можуть бути два варіанти ініціалізації. У першому випадку ініціал ізація є явною і задається константним масивом:

double[ ] x- {5.5, 6.6, 7.7};

За синтаксисом елементи константного масиву варто брати у фігурні дужки.

У другому випадку створення й ініціалізація масиву виконується в об'єктному стилі з викликом конструктора масиву. І це найпоширеніша практика оголошення масивів:

int[ ] d= new int[5];

Якщо масив оголошується без ініціалізації, то створюється тільки посилання на нього. Якщо ініціалізація виконується конструктором, то в динамічній пам'яті створюється сам масив, елементи якого ініціалізуються константами відповідного типу (нуль для арифметики, порожній рядок для

строкових масивів), і посилання зв'язується із цим масивом. Якщо масив ініціалізується константним масивом, то в пам'яті створюється константний масив, з яким і зв'язується посилання.

Розглянемо ще кілька прикладів оголошення масивів:

```
…
int[]k;                          //int- масив (ще без виделення пам’яті)
k=new int [3];                   //Визначаємо масив з трьох цілих
k[0]=-5; k[1]=1; k[2]=55;        // Задаємо елементи масиву
Console.WriteLine(k[2].ToString()); // Виводимо третій елемент масиву
…
```

Зміст наведеного фрагмента ясний з коментарів. Зверніть увагу на деякі особливості. По-перше, масив визначається саме як **int[]k**; а не як один з наступних варіантів:

```
int k[];      // Невірно!
int k[3];     // Невірно!
int[3] k;     // Невірно!
```

По-друге, оскільки масив являє собою посилальний об'єкт, то для створення масиву необхідний рядок **k=new int [3]**;. Саме в ньому ми й визначаємо розмір масиву. Хоча, можливі конструкції виду **int[ ] k = new int [3]**;.

Елементи масиву можна задавати відразу при оголошенні. Наприклад, **int[] k = {-5, 4, 55};**

У C# нумерація елементів масиву починається з нуля. Таким чином, у прикладі початковий елемент масиву - це **k[0].** а останній – **k[2].** Елемента **k[3]** немає.

4.1.2. Динамічні масиви

У C# всі масиви, незалежно від того, яким виразом описується границя, розглядаються як динамічні. У дійсності реальні потреби в розмірі масиву, швидше за все, визначаються в процесі виконання програми.

Вирази, що задають границю зміни індексів, у динамічному випадку містять змінні. Єдина вимога - значення змінних повинні бути визначені в момент оголошення.

Приклад, у якому описана робота з масивом:

```
public void TestDynAr()
{
    // оголошення масиву A1
    Console.WriteLine("Введіть кількість елементів масиву A1");
    int size = int.Parse(Console.ReadLine());
    int[] A1 = new int[size];
}
```

У даній процедурі верхня границя масиву визначається користувачем.

4.1.3. Багатовимірні масиви

Подій масивів на одновимірні і багатовимірні носить історичний характер. Ніякої принципової різниці між ними немає. Одновимірні масиви - це окремий випадок багатовимірних. Або: багатовимірні масиви є природним узагальненням одновимірних. Одновимірні масиви дозволяють задавати такі математичні структури як вектори, двовимірні - матриці, тривимірні - куби даних, масиви більшої розмірності - багатовимірні куби даних.

Оголошення багатовимірного масиву в загальному випадку:

<тип>[, … ,] <ім'я_масиву>;

Число ком, збільшене на одиницю, і задає розмірність масиву. Слід зазначити, що хоча явна ініціалізація з використанням багатовимірних константних масивів можлива, але застосовується рідко через громіздкість такої структури. Простіше ініціалізацію реалізувати програмно.

Приклади

Двовимірний масив:

int[,] k = new int [2,3];

Тут пара квадратних дужок тільки одна. У нашому прикладі в масиві 6 (=2x3) елементів (k[0,0] - перший, k[1,2] - останній). Аналогічно можна задавати багатовимірні масиви. Тривимірний масив:

int[, ,]k = new int [10,10,10];

Варіант ініціалізації багатовимірного масиву:

int[ ] k = {{2,-2},{3,-22},{0,4}};

4.1.4. Масиви масивів

Ще одним видом масивів C# є масиви масивів, називані також порізаними масивами (jagged arrays). Такий масив масивів можна розглядати як одновимірний масив, елементи якого є масивами, елементи яких, у свою чергу, знову можуть бути масивами, і так може тривати до деякого рівня вкладеності.

Ці масиви можуть застосовуватися для подання дерев, у яких вузли можуть мати довільне число нащадків. Таким може бути, наприклад, генеалогічне дерево. Вершини першого рівня - **Fathers**, що представляють батьків, можуть задаватися одновимірним масивом, так що **Fathers[i]** - це і-й батько. Вершини другого рівня представляються масивом масивів - **Children**, так що **Children[i]** - це масив дітей і-го батька, а **Children[i][j]** – це j-та дитина і-го батька. Для подання онуків знадобиться третій рівень, так що **Grandchildren [i][j][k]** буде представляти k-го онука j-ї дитини і-го батька.

Є деякі особливості в оголошенні й ініціалізації таких масивів. Якщо при оголошенні типу багатовимірних масивів для вказівки розмірності використовувалися коми, то для порізаних масивів застосовується більш ясна символіка - сукупності пар квадратних дужок; наприклад, **int [ ][ ]** задає масив, елементи якого - одновимірні масиви елементів типу **int**. Складніше зі створенням самих масивів й їх ініціалізацією. Тут не можна викликати конструктор new **int[3][5]**, оскільки він не задає порізаний масив. Фактично потрібно викликати конструктор для кожного масиву на самому нижньому рівні. У цьому й полягає складність оголошення таких масивів.

Приклад

```
// масив масивів. Оголошення й ініціалізація
 int[ ][ ] jagger = new int[3][ ]
{
   new int[ ] = {5,7,9,11},
   new int[ ] = {2,8},
   new int[ ] = {6,12,4}
}
```

Масив **jagger** має всього два рівні. Можна вважати, що в нього три елементи, кожний з яких є масивом. Для кожного такого масиву необхідно викликати конструктор **new**, щоб створити внутрішній масив. У даному прикладі елементи внутрішніх масивів одержують значення, будучи явно ініціалізовані константними масивами. Звичайно, припустиме й таке оголошення:

```
int[ ][ ] jaggerl = new int[3][ ]
{
   new int[4],
   new int[2],
   new int[3]
}
```

У цьому випадку елементи масиву одержать при ініціалізації нульові значення. Реальну ініціалізацію потрібно буде виконувати програмним шляхом. У конструкторі верхнього рівня константу 3 можна опустити й писати просто **new int[][].** Виклик цього конструктора можна взагалі опустити – він буде матися на увазі:

```
int[][] jagger2 =
{
   new int[4],
   new int[2],
   new int[3]
}:
```

Конструктори нижнього рівня необхідні. Ще одне зауваження – динамічні масиви можливі й тут. У загальному випадку, фаті ці на будь-якому рівні можуть бути виразами, що залежать від змінних. Більше того, припустимо, щоб масиви на нижньому рівні були багатовимірними.

Приклад

```
//  Оголошуємо двовимірний східчастий масив
int[][] k = newint [2][];
// Оголошуємо 0-й елемент східчастого масиву.
// Це знову масив і у ньому 3 елементи
k[0]=new int[3];
// Оголошуємо 1-й елемент східчастого масиву.
// Це знову масив і у ньому 4 елементи
k[1]=new int[4];
k[1] [3]=22; // записуємо 22 в останній елемент масиву
```

…

Зверніть увагу, що у східчастих масивів задається декілька пар квадратних дужок (стільки, скільки становить розмірність у масиву).

4.2. Цикл foreach

Новим видом циклу, що часто використовується й досить зручний при роботі з масивами, є цикл **foreach.** Його синтаксис:

foreach (тип ідентифікатор in контейнер) оператор

Тіло циклу виконується для кожного елемента масиву й закінчується, коли повністю перебрані всі елементи. Тип ідентифікатора повинен бути узгоджений з типом елементів, що зберігаються в масиві даних. Передбачається також, що елементи масиву впорядковані. На кожному кроці циклу ідентифікатор, що задає поточний елемент масиву, одержує значення чергового елемента відповідно до порядку, установленого на елементах масиву. Із цим поточним елементом і виконується тіло циклу -виконується стільки разів, скільки елементів знаходиться в масиві.

Недоліком циклів **foreach** у мові **C#** є те, що цикл працює тільки на читання, але не на запис елементів. Тому наповнювати масив елементами доводиться за допомогою інших операторів циклу.

Приклад використання циклу **foreach:**

```
class Program
 {
  static void Main(string[] args)
  {
    int[,] array1 = {0, 2, 4, 6, 8, 10);
    foreach (int n in array1)
    {
       System.Console.WriteLine(n.ToString());
```

```
        }
        string[ ] array2 = {"hello", "world");
        foreach (string s in array2)
        {
           System.Console.WriteLine(s);
        }
     }
   }
```

У наведеному прикладі цикл перебирає всі елементи масиву **array1.** На це вказує рядок **foreach (int n in array1),** який інтерпретується так: для кожного цілого числа з масиву **array1** робимо щось. Якби елементами масиву були б не цілі, а дійсні числа, то запис виглядав би так:

foreach(float n in array1)

Тобто ми пишемо саме тип елементів масиву. Цикл **foreach** використовується не тільки для масивів, але й для інших об'єктів.

2.3. Метод Format

Щоразу, коли виконувався вивід результатів на консоль, неявно викликався й метод **Format.** Загальний синтаксис такий:

{N [,M [:<коди_форматування>]]}

Обов'язковий параметр N задає індекс об'єкта, що заміняє формат. Можна вважати, що методу завжди передається масив об'єктів, навіть якщо фактично переданий один об'єкт. Індексація об'єктів починається з нуля, як це прийнято в масивах. Другий параметр M, якщо він заданий, визначає мінімальну ширину поля, що виділяється рядку, який не вставляється замість формату. Третій необов'язковий параметр задає форматування, що вказують, як варто форматувати об'єкт. Наприклад, код *C* (Currency) говорить про те, що параметр повинен форматуватися як валюта з врахуванням національних особливостей подання. Код *P* (Percent) задає форматування у вигляді відсотків з точністю до сотої частки.

Дія виводу на консоль використовувалася наступна конструкція:

```
int x=23, y=-4;
…
Console.WriteLine("x={0}, y={1}", x, y);
```

Тут ми використовуємо усередині лапок підстановочні знаки 0, 1 і т.д. (нумерація в них йде з нуля). Змінні при цьому виводяться у форматі за замовчуванням. Для виводу в певному форматі треба використати підстановочні знаки з параметрами. От деякі із них:

d - десятковий формат. Дозволяє задати загальну кількість знаків (при необхідності число доповнюється ліворуч нулями);

f-формат з фіксованою точністю. Дозволяє задати кількість знаків після коми;

x - шістнадцятковий формат;

c - грошовий формат (додає знак долара й показує два знаки після коми);

e - вивід числа в експонентній формі. Приклад використання:

…

```
int a=38;
Console.WrtteLine(“a={0:d4}”, a);              Виведеться 0C38
double pi=3.1415926;
Console.WriteLine("pi={0:f2}", pi);            Виведеться 3.14
Int b=255;
```

```
Console.WritLine("b={0:X}", b);                Виведеться FF.
int c=255;
Console.WriteLine("C"{0:x}", c);               Виведеться ff.
double d=1003.214;
Console.WriteLine("d={0:c}", d);               Виведеться $1, 003.14  в
                                               англійській версії Windows double e=213.1;
Console.WriteLlne("e={0:e}", e);               Виведеться 2.131000e+002
```

...

Параметри підстановочних знаків можна використати як маленькі, так і великі - це однаково. Виключення - вивід числа в шістнадцятковому вигляді (при використанні x цифри *a*, ..., *f* будуть маленькими, при використанні X-великими).

2.4. Методи класу System.Array

Масиви в C# засновані на класі System.Array. У всіх класів, що є масивами, багато загального, оскільки вони є нащадками класу System.Array.

Клас Array має досить велике число власних методів і властивостей.

Приклад

```
using System;
namespace test
{
 class Test
 {
   static void Main(strlng[] args)
   {
    Int [ ] num => (4, -5, 2, 0, 23);   // Оголошення масиву
    foreach (int i in num)              // Виводимо масив
    {
      Console.WriteLine(i.ToString());
    }
    Console.WriteLine("Перевернений масив");
    // Перевертаємо масив
    Array.Reverse(num);
    foreach (int i in num)
    {
      Console.WriteLine (i.ToString());
    }
    Array.Sort(num);                // Сортуємо масив.
    Console.WriteLine ("Відсортований масив");
    foreach (int i in num)
    {
      Console.WriteLlne(i.ToStringO);
    }
    Array.Clear(num, 0, 5);   // Обнуляємо масив
    Console. Console.WriteLine ('Обнулений масив");
    foreach (int i in num)
    {
       Console.WriteLine(i.ToString());
    }
   }
 }
}
```

Тут використовуються статичні методи класу Array для сортування, перевершення і очищення масиву.

У класі **Array** є, наприклад, вбудований статичний метод **IndexOf**, призначений для пошуку елемента в масиві.

Приклад використання:

```
int k = 5;
Console.WriteLine("Число {0} перебуває на (1) місці.", k,
Array.IndexOff(num, k));
```

Цей метод повертає індекс шуканого елемента (нумерація з нуля). Якщо такого елемента немає, то виводиться -1.

Зверніть увагу, що ці методи діють для вбудованих типів (у прикладі масив був типу int). Для користувальницьких типів даних їх застосування теж можливо. Деякі властивості й методи класу **Array** наведені в таблицях 2.1-2.3

Таблиця 2.1 - Статичні методи класу Array

Метод	Опис
BinarySearch	Двійковий пошук. Визначає індекс першого входження зразка у відсортований масив, використовуючи алгоритм двійкового пошуку
Clear	Виконує початкову ініціалізацію елементів. Залежно від типу елементів встановлює значення 0 для арифметичного типу, false - для логічного типу, Null - для посилань, " " - для рядків
Copy	Копіювання частини або всього масиву в інший масив
indexOf	Індекс першого входження зразка в масив
LastlndexOf	Індекс останнього входження зразка в масив
Reverse	Обернення одновимірного масиву. Виконує обернення масиву, переставляючи елементи у зворотному порядку
Sort	Сортування масиву

Таблиця 2.2 - Динамічні методи класу Array

Метод	Опис
ItoStrine	Перетворення елемента масиву в рядкову змінну
Clone	Дозволяє створити пласку або глибоку копію масиву. У першому випадку створюються тільки елементи першого рівня, а посилання вказують на ті самі об'єкти. У другому випадку копіюються об'єкти на всіх рівнях. Для масивів створюється тільки пласка копія
CopyTo	Копіюються всі елементи одновимірного масиву в інший одновимірний масив, починаючи із заданого індексу: co11.CopyTo(co12,0);
GetLeneth	Повертає кількість елементів масиву у зазначеному вимірі
GetLowerBound, GetUpperBound	Повертає нижню й верхню границю У зазначеному вимірі. Для масивів нижня границя завжди дорівнює нулю
GetValue, SetValue	Повертає або встановлює значення елемента масиву із зазначеними індексами

Таблиця 2.3 - Властивості класу Array

Властивість	Опис
Length	Число елементів масиву
Rank	Розмірність масиву

Приклад використання наведених методів для роботи з масивами,

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace Arrayl
{
 class Program
 {
  static void Main(string[] args)
  {
    int[] Arr1 - {4, -5, 2, 0, 23 };// Оголошення масиву,
    foreach (int I In Arr1)        // Виводимо масив.
    {
      Console.WriteLine(I.ToString());
    }
// Число елементів масиву (Arr1.Length)
    Console.WriteLlne<"Кількість елементів масиву = " +
            Arr1.Length);
// Розмірність масиву (Rank)
    Console.WrlteLlne("Розмірність масиву = " + Arr1.Rank);
// Верхня границя масиву (GetUpperBound)
    Console.WrtteLinef'Верхня границя масиву = " +
            Arr1.GetUpperBound(0));
// Нижня граници масиву (GetLoworBound)
    Consoie.WriteUne("Нижня границя масиву = {0}",
            Arr1.GetLowerBound(O));
// Прочитати значення 2-го елемента
    Console.WriteLine("Значення 2-го елемента = {0}",
            Arr1.GetValue(1));
// Змінити значення 2-го елемента на -11
            Arr1.SetValue(-11,1);
    Console.WriteLiiie("Змінене значення 2-го елемента = {0}",
            Arr1.GetValue(1));
// Оголошення масиву Arr2.
// Кількість елементів масиву визначається як
// верхня границя масиву Arr1 + 1 (Arrl.GetUpperBound(O) + 1)
   int [] Arr2 - new int[Arr1.GetUpperBound(O) + 1];
    Console.WriteLine("Скопійовании масив");
// Копіювання масиву Arr1 в Arr2 починаючи з 0 елемента
          Arr1.CopyTo(Arr2,0);
    foreach (int I In Arr2)
    {
      Console.WriteLlne(i.ToString());
    }
    Console.WriteLine("Поверненний масив");
    Array.Reverse(Arr1); // Перевертаємо масив
    foreach (Int I In An*1)
    {
      Console.WriteLine(i.ToString)));
    }
// Пошук індексу в несортованому масиві елемента "23"
    Console.WriieLine(Індекс елемента 23 =" +
       Array.lndexOf(Arrl, 23));
Array.Sort(Arri);                  // Сортуємо масив.
Console.WfitoLine("Відсортований масив");
```

```
 foreach (Int i in Arr1)
 {
Console.WrlteLine(i.ToString());
}
// Двійковий пошук індексу у відсортованому масиві елемента "23"
Console.WrileLine('Індекс елемента 23    =’’ +
Array Binary Search) Arr1 ,23));
Console.WriteLine("Індекс елемента 23 після сортування =’’+
Array.lndexOf(Arr1, 23));
Array.Clear(Arr1, 0, 5);                // Обнулясмо масив.
Console. WriteLine ("Обнуленньlй масив");
foreach (Int I in Arrl)
{
Console.WriteLine(i.ToString());
 }
//  Оголошення масиву float.  Елементи зчитуються із клавіатури
Console.WriteLine(“Введіть розмір масиву");
Int Zise_Arr3 = Int.Parse(Coneole.ReadLine());
float [ ] Arr3 = newfloat[Zise_Arr3];
Console.WriteLine(‘’Введіть елементи масиву’’);
for(lnt ii=0;ii<Zise_Arr3;ii**)
{
Arr3[ll]«float.Parse(Console.ReadLine());
}
Console.WnteLine("Введений відсортований масив");
Array.Sort(Arr3);
for (Int II - 0; II < 2Use Arr3; II++)
{
Coneole.WrlteLine(Arr3[ii].ToString());
}
float k = Arr3.Max();                        // Пошук максимуму
Console.Writeüne("Max="                  +                  k.ToString());
k = Arr3.Min();                              // Пошук мінімуму
Console.WriteLine("Min=" + k.ToStrin());
bool I = Array.Equals(Arr1, Arr2);           //  Порівняння двох масивів
(Equals) Console.WriteLine("BOOL=" + I.ToString());
I = Array.Equals(Arr1, Arr1);
Console.WriteLine(‘’BOOL=" + I.ToString());
}
}
}
```