

Масиви

1. Поняття масиву

Масив – набір елементів однакового типу, звертатися до яких можна за одним спільним ім'ям. При оголошенні масиву використовують наступний синтаксис:

```
<тип> [ ] <ім'я_масиву>;
```

де *тип* – тип даних, *ім'я_масиву* – ім'я масиву за яким в подальшому можна звертатись до його елементів.

Після оголошення масиву необхідно за допомогою оператора **new** задати його розмірність.

Наприклад:

```
int[] myArray;  
myArray = new int[10]; //масив цілих з десяти  
                        //елементів  
double[] NumArr = new double[5];  
                        // масив з 5ти дійсних
```

Можна також одразу ініціювати масив початковими значеннями:

```
int[] myArray = new int[5] { 1, 3, 5, 8, 2 };
```

або ще простіше:

```
int[] myArray = { 1, 3, 5, 8, 2 };
```

без ключового слова **new** розмір масиву визначиться автоматично. Всі перераховані нище способи будуть рівноцінні.

```
int[] nums2 = new int[4] { 1, 2, 3, 5 };  
  
int[] nums3 = new int[] { 1, 2, 3, 5 };  
  
int[] nums4 = new[] { 1, 2, 3, 5 };  
  
int[] nums5 = { 1, 2, 3, 5 };
```

Звертатися до елементів масиву можна виказавши його ім'я та індекс елемента у квадратних дужках. Проте слід зазначити, що у мові C# **нумерація елементів починається з нуля**, і перший елемент масиву матиме індекс 0.

2. Клас System.Array

Всі масиви C# є нащадками класу **System.Array** у якого є ряд корисних властивостей та методів. Наприклад властивість:

Length – повертає кількість елементів масиву;

Rank – повертає розмірність масиву (1 для одновимірного і т.д.).

Тепер розглянемо приклад програми для виводу на екран масиву за допомогою циклу **for**:

```
using System;
class Program {
    static void Main() {
        int[] arr = { 1, 3, 5, 8, 2 };
        for (int i = 0; i < arr.Length; i++) {
            Console.WriteLine("arr[{0}]={1}", i,
arr[i]);
        }
        Console.ReadKey();
    }
}
```

Ця програма виведе на екран наступне:

```
arr[0]=1
arr[1]=3
arr[2]=5
arr[3]=8
arr[4]=2
```

В цій програмі можна замінити цикл **for** на **foreach**:

```
foreach (int i in arr) {
    Console.WriteLine(i);
}
```

але слід враховувати, що у випадку циклу **foreach** ітераційну змінну можна використовувати тільки для читання.

```
int[] numbers = new int[] { 1, 2, 3, 4, 5 };
foreach (int i in numbers)
{
    Console.WriteLine(i);
}
```

```
int[] numbers = new int[] { 1, 2, 3, 4, 5 };
for (int i = 0; i < numbers.Length; i++)
{
    numbers[i] = numbers[i] * 2;
    Console.WriteLine(numbers[i]);
}
```

Серед методів **System.Array** слід виділити наступні:

Clear() – очищує масив, заповнивши всі елементи 0 або пустими значеннями **null**;

Copy() – копіює заданий діапазон значень з одного масиву в інший;

Clone() – створює повну копію масиву разом зі значеннями;

IndexOf(), **LastIndexOf()** – відповідно шукають перше та

останнє входження елементу в масив;

Sort() – сортує одновимірний масив за зростанням;

BinarySearch() – швидкий пошук в сортованому масиві;

Reverse() – змінює порядок слідування елементів масиву на протилежний.

Розглянемо приклад:

```
using System;
class Program {
    static void Main() {
        int[] arr = { 1, 8, 5, 3, 2 }; //створимо масив
                                     //створимо копію
        int[] arrNew = (int[])arr.Clone();
        Array.Sort(arrNew);           //відсортуємо масив
        for (int i = 0; i < arrNew.Length; i++) {
            //виведемо відсортовані значення
            Console.WriteLine(«arrNew[{0}]=»+i,
                              arrNew[i]);
        }
        Console.ReadKey();
    }
}
```

результатом роботи програми буде:

```
arrNew[0]=1
arrNew[1]=2
arrNew[2]=3
arrNew[3]=5
arrNew[4]=8
```

Розберемо найбільш використовувані методи. Наприклад, змінимо порядок елементів і розмір масиву:

```
1  int[] numbers = { -4, -3, -2, -1,0, 1, 2, 3, 4 };
2
3  // расположим в обратном порядке
4  Array.Reverse(numbers);
5
6  // уменьшим массив до 4 элементов
7  Array.Resize(ref numbers, 4);
8
9  foreach(int number in numbers)
10 {
11     Console.Write($"{number} \t");
12 }
```

Результат програми:

4 3 2 1

У функції **Resize** першим параметром є змінний масив, а другий - кількість елементів, які повинні бути в масиві. Якщо другий параметр менше довжини масиву, то масив буде скорочуватися. Якщо значення параметра, навпаки, більше, то масив доповнюється додатковими елементами, які мають значення за замовчуванням.

Метод **Copy** копіює частину одного масиву в інший:

```
1 int[] numbers = { -4, -3, -2, -1, 0, 1, 2, 3, 4 };
2 int[] numbers2 = new int[5];
3
4 // копіюємо из numbers с 2-го індекса 5 елементів
5 // и поместим их в массив numbers2, начиная с 0-го індекса
6 Array.Copy(numbers, 2, numbers2, 0, 5);
7
8 foreach(int number in numbers2)
9 {
10     Console.WriteLine($"{number} \t");
11 }
```

Результат програми:

```
-2 -1 0 1 2
```

3. Багатовимірні масиви

Багатовимірний масив – це масив який визначається двома або більше вимірами, а до його елементів треба звертатися за допомогою двох або більше індексів.

Найпростіші багатовимірні масиви – двовимірні прямокутні масиви (або матриці). Для їх оголошення використовують наступний синтаксис:

`<тип>[,] <ім'я_масиву>;`

де *тип* – тип даних, *ім'я_масиву* – ім'я масиву за яким в подальшому можна звертатись до його елементів.

Приклад створення матриці 5 на 7:

```
int[, ] matrix = new int[5, 7];
```

до її елементів можна звертатися наступним чином:

```
matrix[0,0] = 7; //лівий верхній елемент
matrix[3,5] = 10; //десь всередині
matrix[4,6] = 0; //правий нижній елемент
```

```
int[] nums1 = new int[] { 0, 1, 2, 3, 4, 5 };
int[, ] nums2 = { { 0, 1, 2 }, { 3, 4, 5 } };
```

Візуально обидва масиви можна представити таким чином:

- Одновимірний масив nums1

0	1	2	3	4	5
---	---	---	---	---	---

- Двомірний масив nums2

0	1	2
3	4	5

Оскільки масив nums2 двомірний, він являє собою просту таблицю. Всі можливі способи визначення двомірних масивів:

```
int[,] nums1;  
int[,] nums2 = new int[2, 3];  
int[,] nums3 = new int[2, 3] { { 0, 1, 2 }, { 3, 4, 5 } };  
int[,] nums4 = new int[,] { { 0, 1, 2 }, { 3, 4, 5 } };  
int[,] nums5 = new [,]{ { 0, 1, 2 }, { 3, 4, 5 } };  
int[,] nums6 = { { 0, 1, 2 }, { 3, 4, 5 } };
```

Тепер розглянемо як заповнити та вивести на екран прямокутну матрицю за допомогою циклів:

```
using System;  
class Program {  
    static void Main() {  
        //створюємо матрицю  
        int[,] matrix = new int[5, 7];  
        //заповнюємо елементи матриці сумою індексів  
        for (int i = 0; i < 5; i++) {  
            for (int j = 0; j < 7; j++) {  
                matrix[i, j] = i + j;  
            }  
        }  
        //виведемо отримані значення на екран  
        for (int i = 0; i < 5; i++) {  
            for (int j = 0; j < 7; j++) {  
                Console.Write("{0,3}", matrix[i, j]);  
                // {0,3} - Place Holder, де 3 це  
                //відстань до наступного елемента  
            }  
            Console.WriteLine();  
        }  
    }  
}
```

```

    }
    Console.ReadKey();
}
}

```

На екрані отримаємо:

```

0  1  2  3  4  5  6
1  2  3  4  5  6  7
2  3  4  5  6  7  8
3  4  5  6  7  8  9
4  5  6  7  8  9 10

```

Аналогічно до двовимірних можна створювати тривимірні, чотиривимірні і т.д. масиви:

```

int[, ,]matrix3 = new int[5, 7, 5];
int[, , ,]matrix4 = new int[4, 4, 3, 3];

```

Певну складність може представляти перебір багатовимірного масиву. Перш за все треба враховувати, що довжина такого масиву - це сукупна кількість елементів.

```

int[,] mas = { { 1, 2, 3 }, { 4, 5, 6 }, { 7, 8, 9 }, { 10, 11, 12 } };
foreach (int i in mas)
    Console.Write($"{i} ");
Console.WriteLine();

```

В даному випадку довжина масиву `mas` дорівнює 12. І цикл `foreach` виводить всі елементи масиву в рядок:

```
1 2 3 4 5 6 7 8 9 10 11 12
```

Але що якщо ми хочемо окремо пробігтися по кожному рядку в таблиці? В цьому випадку треба отримати кількість елементів в розмірності. Зокрема, у кожного масиву є метод **`GetUpperBound (dimension)`**, який повертає індекс останнього елемента в певній розмірності. І якщо ми говоримо безпосередньо про двовірному масиві, то перша розмірність (з індексом 0) по суті це і є таблиця. І за допомогою виразу **`mas.GetUpperBound(0) + 1`** можна отримати кількість рядків таблиці, представлені двовірним масивом. А через **`mas.Length / rows`** можна отримати кількість елементів в кожному рядку:

```

1  int[,] mas = { { 1, 2, 3 }, { 4, 5, 6 }, { 7, 8, 9 }, { 10, 11, 12 } };
2
3  int rows = mas.GetUpperBound(0) + 1;
4  int columns = mas.Length / rows;
5  // или так
6  // int columns = mas.GetUpperBound(1) + 1;
7
8  for (int i = 0; i < rows; i++)
9  {
10     for (int j = 0; j < columns; j++)
11     {
12         Console.Write($"{mas[i, j]} \t");
13     }
14     Console.WriteLine();
15 }

```

```

1 2 3
4 5 6
7 8 9
10 11 12

```

4 Зубчасті масиви (масив масивів)

Зубчасті масиви (jagged array) – це масиви рядки яких можуть мати різну довжину. Причому кожен рядок зубчастого масиву є одновимірним масивом. Ці масиви оголошуються на ступінним чином:

тип[] [] ім'я_масиву;

Щоб створити двовимірний зубчастий масив, спочатку треба задати кількість рядків, наприклад:

```
int[][] arrJ = new int[3][];
```

потім задати розмірність кожного з рядків:

```

arrJ[0] = new int[3];
arrJ[1] = new int[4];
arrJ[2] = new int[2];

```

звертатися до елементів масиву потрібно так:

```

arrJ[0][0] = 2;
arrJ[0][1] = 4;
//...
arrJ[2][1] = 17;

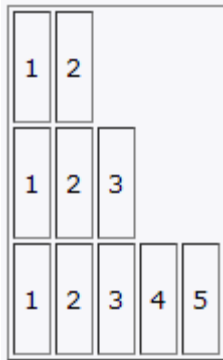
```

```

int[][] nums = new int[3][];
nums[0] = new int[2] { 1, 2 };           // выделяем память для первого подмассива
nums[1] = new int[3] { 1, 2, 3 };        // выделяем память для второго подмассива
nums[2] = new int[5] { 1, 2, 3, 4, 5 };  // выделяем память для третьего подмассива

```

Тут дві групи квадратних дужок вказують, що це масив масивів, тобто такий масив, який в свою чергу містить в собі інші масиви. Причому довжина масиву вказується тільки в перших квадратних дужках, всі наступні квадратні дужки повинні бути порожні: `new int[3][]`. В даному випадку масив *nums* містить три масиви. Причому розмірність кожного з цих масивів може не збігатися. Зовнішній вигляд масиву *nums*



Часто зубчасті масиви доцільніше використовувати замість прямокутних. Наприклад в методі Гауса для розв'язання системи лінійних рівнянь, якщо в матриці необхідно переставляти місцями рядки, то при використанні зубчастого масиву відпадає необхідність поелементного копіювання даних.

Приклад:

```
using System;
class Program {
    static void Main() {
        //створюємо зубчасту матрицю 3x4
        float[ ][ ] arrJ = new float[3][ ];
        for (int i = 0; i < arrJ.Length; i++)
        {
            arrJ[i] = new float[4];
        }
        //заповнюємо елементи рядка його номером
        for (int i = 0; i < arrJ.Length; i++)
        {
            for (int j = 0; j < arrJ[i].Length; j++)
            {
                arrJ[i][j] = i;
            }
        }
        //міняємо місцями перший рядок з останнім
        float[] x = arrJ[0];
        arrJ[0] = arrJ[2];
```



```
        arrJ[2] = x;
    }
}
```

5. Масив параметрів і ключове слово params

У всіх попередніх прикладах ми використовували постійне число параметрів. Але, використовуючи ключове слово `params`, ми можемо передавати невизначену кількість параметрів:

```
1  static void Addition(params int[] integers)
2  {
3      int result = 0;
4      for (int i = 0; i < integers.Length; i++)
5      {
6          result += integers[i];
7      }
8      Console.WriteLine(result);
9  }
10
11 static void Main(string[] args)
12 {
13     Addition(1, 2, 3, 4, 5);
14
15     int[] array = new int[] { 1, 2, 3, 4 };
16     Addition(array);
17
18     Addition();
19     Console.ReadLine();
20 }
```

Сам параметр з ключовим словом `params` при визначенні методу повинен представляти одновимірний масив того типу, дані якого ми збираємося використовувати. При виклику методу на місце параметра з модифікатором `params` можемо передати як окремі значення, так і масив значень, або взагалі не передавати параметри.

Якщо ж нам треба передати якісь інші параметри, то вони мають бути вказані до параметра з ключовим словом `params`:

```
1  //Так працює
2  static void Addition( int x, string mes, params int[] integers)
3  {}
```

Виклик подібного методу:

```
Addition(2, "hello", 1, 3, 4);
```

Однак після параметра з модифікатором `params` ми не можемо вказувати інші параметри. Тобто таке визначення методу неприпустимо:

```
1 //Так НЕ работает
2 static void Addition(params int[] integers, int x, string mes)
3 {}
```

Також цей спосіб передачі параметрів треба відрізнити від передачі масиву в якості параметра:

```
1 // передача параметра с params
2 static void Addition(params int[] integers)
3 {
4     int result = 0;
5     for (int i = 0; i < integers.Length; i++)
6     {
7         result += integers[i];
8     }
9     Console.WriteLine(result);
10 }
11 // передача массива
12 static void AdditionMas(int[] integers, int k)
13 {
14     int result = 0;
15     for (int i = 0; i < integers.Length; i++)
16     {
17         result += (integers[i]*k);
18     }
19     Console.WriteLine(result);
20 }
```

Так як метод *AdditionMas* приймає як параметр масив без ключового слова *params*, то при його виклику обов'язково треба передати в якості параметра масив.

Контрольні запитання та завдання

1. Що таке масив? Наведіть власні приклади оголошення масивів.
2. За допомогою «Бібліотеки MSDN» [1], [2] ознайомтеся з основними властивостями та методами класу `System.Array`.
3. Класифікація масивів.
4. В чому різниця між прямокутними та зубчастими масивами?
5. Навести приклади використання прямокутних та зубчастих масивів.