МАСИВИ

Масиви - це індексований набір об'єктів одного типу. Індексація починається від 0 і не може бути змінена. Для їх реалізації в C # призначений клас System.Array. Для цього класу реалізована вбудована підтримка.

Одномірні масиви.

```
int [] a1 = new int [10];
int [] a2 = {1, 2, 3, 4};
int [] a3 = new int [] {1, 2, 3, 4};
int [] a4 = new int [4] {1, 2, 3, 4};
```

У першому випадку створюється масив їх заданої кількості елементів, ініціалізованих значеннями за замовчуванням. Наступні три способи абсолютно аналогічні за своєю дією. Різниця в тому, що якщо програміст хотів створити масив з 4 елементів і помилився, то в другому випадку програма не видасть ніякої помилки, а в четвертому виникне помилка часу компіляції. У будь-якому випадку довжина масиву задається при його створенні, а не при його оголошенні і повинна бути константою. Можливе створення масивів структур, об'єктів, інтерфейсів. Якщо не форматувати елементи масиву, на які посилаються типів, то кожен елемент буде містити нульову посилання null. Місце в купі під кожен елемент потрібно виділити окремо, можливо за допомогою різних конструкторів.

MyClass [] ac1 = new MyClass [3] {new MyClass (), new MyClass (), new MyClass ()};

Доступ до елементів так само, як і в C - через оператор «індекс масиву». При спробі доступу (читання / запис) до елементу масиву за межами можливого діапазону, виникає виключення System.IndexOutOfRangeException.

Для вказівки розміру масиву зазвичай використовуються константи, але це може бути і змінна (або вираз), якщо не визначаються значення його елементів.

```
int size = 10;
```

```
        int [] a1 = new int [size];
        int [] a4 = new int [size] {1, 2, 3, 4};
```

Приклад. Створимо клас для роботи з одновимірним масивом, а саме для генерації значень масиву і для виведення їх DataGridView, а імені в Label.

опис класу

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace prim1_lek7
{
   class Arrs
   {
      private static Random rnd = new Random ();
      public static void CreateOneDimAr (int [] A)
      {
         for (int i = 0; i <A.GetLength (0); i ++)
            A [i] = rnd.Next (1, 100);
      } // CreateOneDimAr
      public static void PrintAr1 (string name, int [] A, DataGridView d, Label l)
      {
         l.Text = "Масив -" + name;
         d.Rows.Clear ();
         for (int i = 0; i <A.GetLength (0); i ++)
            d.Rows.Add (A [i] .ToString ());
      } // PrintAr1

   }
```

```
}
```

Робота з класом

```
namespace prim1_lek7
{
   public partial class Form1: Form
   {
      public Form1 ()
      {
         InitializeComponent ();
      }

      private void button1_Click (object sender, EventArgs e)
      {
         int [] a = new int [5];
         Arrs.CreateOneDimAr (a);
         Arrs.PrintAr1 ( "A", a, dg, label1);
      }
   }
}
```

динамічні масиви

У всіх вищенаведених прикладах оголошувалися статичні масиви, оскільки нижня межа дорівнює нулю за визначенням, а верхня завжди задавалася в цих прикладах константою. Нагадаю, що в C # всі масиви незалежно від того, яким виразом описується межа, розглядаються як динамічні і пам'ять їм розподіляється в купі. Вважаю, що це відображення розумної точки зору на те, що статичні масиви швидше виняток, а правилом є використання динамічних масивів. Дійсно реальні потреби в розмірі масиву, швидше за все, з'ясовуються в процесі роботи в діалозі з користувачем.

Чисто синтаксично немає суттєвої різниці в оголошенні статичних і динамічних масивів. Вираз, що задає кордон зміни індексів, в динамічному

випадку містить змінні. Єдина вимога - значення змінних повинні бути визначені в момент оголошення. Це обмеження в C # автоматично виконується, оскільки добре відомо, наскільки вимогливо C # контролює ініціалізацію змінних.

багатовимірні масиви
Масиви з двома і більше вимірами

тип [,] імя_массіва
Ініціалізація двовимірних масивів

```
int [,] aa5 = new int [3,4];
int [,] aa6 = {{1,2}, {4,5}, {7,8}};
int [,] aa7 = new int [3,2] {{1,2}, {4,5}, {7,8}};
```

Приклад. Створимо клас для роботи з двовимірними масивами, а саме для генерації значень масивів, для виведення їх DataGridView, а імені в Label і для перемноження массівов.Прі цьому розмірність масивів буде задаватися динамічно.

опис класу

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace prim1_lek7
{
   class Arrs2
   {
      private static Random rnd = new Random ();
```

```
public static void CreateOneDimAr (int [,] A)
{
   for (int i = 0; i <A.GetLength (0); i ++)
      for (int j = 0; j <A.GetLength (1); j ++)
      A [i, j] = rnd.Next (1, 10);
} // CreateOneDimAr
public static void PrintAr1 (string name, int [,] A, DataGridView d, Label l)
{
```

l.Text = "Масив -" + name; // Тип колонки DataGridView DataGridViewTextBoxColumn dgvAge; // Створюємо потрібну кількість колонок for (int i = 0; i <A.GetLength (0); i ++) {dgvAge = new DataGridViewTextBoxColumn (); // Задаємо ширину колонки dgvAge.Width = 40; // Додали колонку d.Columns.Add (dgvAge); } D.Rows.Clear (); // Вказуємо контролери в який пишемо кількість рядків і стовпців d.RowCount = A.GetLength (0); d.ColumnCount = A.GetLength (1); for (int i = 0; i <A.GetLength (0); i ++) for (int j = 0; j <A.GetLength (1); j ++) //d.Rows.Add(A[i,j] .ToString ()); d.Rows [i] .Cells [j] .Value = A [i, j] .ToString (); } // PrintAr1 // Метод MultMatr виконує множення прямокутних матриць public static void MultMatr (int [,] A, int [,] B, int [,] C) {if (A.GetLength (1)! = B.GetLength (0)) MessageBox.Show ("MultMatr: помилка розмірності!"); else for (int i = 0; i <A.GetLength (0); i ++) for (int j = 0; j <B.GetLength (1); j ++) {int s = 0; for (int k = 0; k <A.GetLength (1); k ++) s + = A [i, k] * B [k, j]; C [i, j] = s; }} // MultMatr}} РеалізаціяПерехід на нову форму Form2 f = new Form2 (); f.Show (); Робота з массівамиprivate void button1_Click (object sender, EventArgs e) {int n1, m1, n2, m2, n3, m3; n1 = Convert.ToInt16 (textBox1.Text); m1 = Convert.ToInt16 (textBox2.Text); n2 = Convert.ToInt16 (textBox3.Text); m2 = Convert.ToInt16 (textBox3.Text); n3 = n1; m3 = m2; int [,] a, b, c; a = new int [n1, m1]; b = new int [n2, m2]; c = new int [n3, m3]; Arrs2.CreateOneDimAr (a); Arrs2.CreateOneDimAr (b); Arrs2.PrintAr1 ("A", a, dataGridView1, label1); Arrs2.PrintAr1 ("B", b, dataGridView2, label2); Arrs2.MultMatr (a, b, c);

Arrs2.PrintAr1 ("C", c, dataGridView3, label3); } Масиви масивів (невирівняному масиви (jagged arrays), ступінчасті масиви, Порізані масиви) Це масив масивів. Невирівняному тому, що складові його масиви можуть мати різну довжину, на відміну від прямокутних. Такий масив масивів можна розглядати як одновимірний масив, елементи якого є масивами, елементи яких, в свою чергу знову можуть бути масивами і так може тривати до деякого рівня вложенності.Есть деякі особливості в оголошенні та ініціалізації таких масивів. Якщо при оголошенні типу багатовимірних масивів для вказівки розмірності використовувалися коми, то для порізаних масивів використовується більш ясна символіка - сукупності пар квадратних дужок, наприклад int [] [] задає масив, елементи якого одномірні масиви елементів типу int.Сложнее зі створенням самих масивів і їх инициализацией. Тут не можна викликати конструктор new int [3] [5], оскільки він не задає порізаний масив. Фактично потрібно викликати конструктор для кожного масиву на найнижчому рівні. В цьому і полягає складність оголошення таких масивів. int [] [] ja = new int [3] []; ja [0] = new int [7]; ja [2] = new int [5]; ja [1] [3] = 12; Приклад. Створимо клас для виведення на екран значень ступеневої масиву в DataGridView. У самому масиві буде зберігатися генеалогічне древо.Опісаніе классcausing System; using System.Collections.Generic; using System.Linq; using System.Text; using System.Windows.Forms; using System.Drawing; namespace prim1_lek7 {class Arrs3 {public static void PrintAr (string [] F, string [] [] C, DataGridView d) {int n = 0; // Знаходимо максимальну кількість дітей for (int i = 0; i <F.Length; i ++) {if (C [i] .Length> = n) n = C [i] .Length; } // Тип колонки DataGridView DataGridViewTextBoxColumn dgvAge; // Створюємо потрібну кількість колонок for (int i = 0; i <n; i ++) {dgvAge = new DataGridViewTextBoxColumn (); // Задаємо ширину колонки dgvAge.Width = 100; // Додали колонку d.Columns.Add (dgvAge); } D.Rows.Clear (); // Вказуємо контролери в який пишемо кількість рядків і стовпців d.RowCount = F.Length * 2; d.ColumnCount = n; int ii = 0; for (int i = 0; i <F.Length; i ++) {// d.Rows.Add ("Батько", F [i], "Його діти"); // Змінюємо

колір рядка d.Rows [ii] .DefaultCellStyle.BackColor = Color.Aqua; d.Rows [ii] .Cells [0] .Value = "Батько"; d.Rows [ii] .Cells [1] .Value = F [i]; d.Rows [ii] .Cells [2] .Value = "Його діти"; ii ++; for (int j = 0; j <C [i] .Length; j ++) {d.Rows [ii] .Cells [j] .Value = C [i] [j]; } Ii ++; }} // PrintAr}} Реалізаціяprivate void button1_Click (object sender, EventArgs e) {int Fcount = 3; string [] Fathers = new string [Fcount]; Fathers [0] = "Микола"; Fathers [1] = "Сергій"; Fathers [2] = "Петро"; string [] [] Children = new string [Fcount] []; Children [0] = new string [] { "Ольга", "Федір"}; Children [1] = new string [] { "Сергій", "Валентина", "Іра", "Дмитро"}; Children [2] = new string [] { "Марія", "Ірина", "Надія"}; Arrs3.PrintAr (Fathers, Children, dataGridView1); } Цикл foreachДля роботи з масивами може використовуватися цикл foreach, який перебирає всі елементи масиву. Його синтаксис: foreach (тіп_елемента імя_тек_ел in імя_массіва) тіло цікладля одновимірного масиву foreach (int e in a4) Console.WriteLine (e); для прямокутного масиву foreach (int e in aa7) Console.WriteLine (e); Для невирівняному масивів повинні використовуватися вкладені цикли: foreach (int [] e1 in ja) foreach (int e2 in e1) Console.WriteLine (e2); Пе редача масивів в методи, масиви як повертаються значеніяМассів - це контрольний тип. При визначенні формальних параметрів не вказується число елементів масиву, але вказується його тип: одновимірний, прямокутний, невирівняному. void M1 (int [] a) void M2 (int [,] a) void M3 (int [] [] a) У іншому синтаксис такий же як для звичайних параметрів. Також можуть бути вказані атрибути для позначення вхідного, вихідного і параметра за посиланням. void M1 (int [] a) {a [2] = 5; } Void M1 (int [] a) {a = new int [4] {5, 6, 7, 8}; } І той же для атрибута ref.Массів також може бути повертається значенням. int [] NewArray (int i) {return new int [i]; } Атрибут paramsПредположім нам потрібно визначити метод, який виводить на екран будь-яку кількість цілих чисел. Можна скористатися попереднім способом. Але можна обійтися і без явного створення масиву. Для цього параметр повинен бути описаний з атрибутом params void Print (params int [] nums) {foreach (int e in nums) Console.WriteLine (e); } Даний параметр це обов'язково одновимірний масив,

такий параметр у методу може бути тільки один, і якщо він є, то в списку параметрів вказується последнім.Пользовательскій індексатор Це фактично оператор індекс масиву. Але в C # він не може бути перевантажений як оператор, це і призвело до появи індексаторів. Індексатор зазвичай використовується для тих класів, які являють собою колекції об'єктів. Синтаксис його наступний: тип this [тип аргумент] {get; set;}. Можна визначити індексатор для читання, для запису і для читання / запису. Індексатор не може бути статичним. Прийнято в якості індексу використовувати цілочисельний тип, але це не обов'язкова вимога. Це може бути рядок або будь-який інший тип данних.Індексатор перевірки меж масиву. class IndexClass {private int a = 12; private int b = 10; public int this [int index] {get {switch (index) {case 1: return a; case 2: return b; default: throw new IndexOutOfRangeException (); }} Set {switch (index) {case 1: a = value; break; case 2: b = value; break; default: throw new IndexOutOfRangeException (); }}}} Якщо необхідно викликати індексатор з методу цього ж класу використовується ключове слово this: public void f () {Console.WriteLine (this [2]); } У класу одночасно може бути визначено будь-яку кількість індексаторів (правила ті ж, що і при перевантаженні методів). public int this [string index] {get {switch (index) {case "A": return a; case "B": return b; default: throw new IndexOutOfRangeException (); }} Set {switch (index) {case "A": a = value; break; case "B": b = value; break; default: throw new IndexOutOfRangeException (); }}} Можна визначати індексатори декількома індексами: public int this [int index1, int index2] {get {if (index1 == index2) return a; else return b; } Set {if (index1 == index2) a = value; else b = value; }}