Масиви

Масив в C# е обект:

– базов клас System.Array**;**

– свойство Length **– брой на елементите в масива.**

1. Декларация на масив

<тип>[ ] <име_на_масив>;

2. Създаване на масив

<име_на_масив> = new <тип>[размер];

Пример: Едномерен масив

```
using System;
class SingleArray
{
   private int[] a;
   public SingleArray (int number)
   {
      a = new int [number];
      Console.WriteLine ("Въведете {0} цели числа.",number);
      for (int i = 0; i < number; i++)
      {
         Console.Write ("a[{0}]=", i);
         a[i] = int.Parse(Console.ReadLine());
      }
   }
```

```
   public void PrintArray()
   {
      Console.WriteLine ("Масив");
      for (int i = 0; i < a.Length; i++)
         Console.Write (a[i] + " ");
      Console.WriteLine ();
   }
   static void Main()
   {
      SingleArray x = new SingleArray(5);
      x.PrintArray ();
   }
}
```

3. Многомерен масив

<тип>[, , ...,] <име_на_масив>;

или

<тип>[][]...[] <име_на_масив>;

Метод Array.GetLength (int dimension) **– определя дължината на даден размер** dimension **на масива.**

Метод Aray.Rank () **– връща реда (броя на размерностите на масива).**

Пример: Двумерен масив

```
using System;
class TwoDimArray
{
   private float[,] a;
   public TwoDimArray (int number1, int number2)
   {
      a = new float [number1, number2];
      Console.WriteLine ("Въведете матрица {0}x{1} " +
                         "от реални числа.", number1, number2);
      for (int i = 0; i < a.GetLength(0); i++)
         for (int j = 0; j < a.GetLength(1); j++)
         {
            Console.Write ("a[{0},{1}]=", i, j);
            a[i,j] = float.Parse (Console.ReadLine());
         }
   }
```

```
   public void PrintArray()
   {
      Console.WriteLine ("Масив");
      for (int i = 0; i < a.GetLength(0); i++)
      {  for (int j = 0; j < a.GetLength(1); j++)
            Console.Write ("{0,5:f2}", a[i,j]);
         Console.WriteLine ();
      }
   }
   static void Main()
   {
      TwoDimArray x = new TwoDimArray (5, 3);
      x.PrintArray ();
   }
}
```

Пример: Масив с различна дължина на редовете – масивът се разглежда като масив от масиви

```
using System;
class JaggedArray
{  private int[][] a;
   public JaggedArray (int rows, params int[] n)
   {  a = new int [rows][];
      for (int i = 0; i < rows; i++)
         a[i] = new int [n[i]];
      for (int i = 0; i < a.Length; i++)
      {  Console.WriteLine("Въведете {0} цели числа.",
                           a[i].Length);
         for (int j = 0; j < a[i].Length; j++)
         {  Console.Write ("a[{0},{1}]=", i, j);
            a[i][j] = int.Parse (Console.ReadLine ());
         }
      }
   }
```

```
   public void PrintArray ()
   {
      Console.WriteLine
               ("Масив с различна дължина на редовете");
      for (int i = 0; i < a.Length; i++)
      {
         for (int j = 0; j < a[i].Length; j++)
            Console.Write ("{0,5}", a[i][j]);
         Console.WriteLine ();
      }
   }
   static void Main()
   {
      JaggedArray x = new JaggedArray (3, 2, 5, 3);
      x.PrintArray ();
   }
}
```

Индексатори (Indexers)

Индексатор – разглежда обект като масив; използва се като елегантен масив.

1. **Дефиниране на индексатор**
   а) има индекс като аргумент;
   б) като име на индексатор се използва this.

```
class <име_на_клас>
{
   public <тип> this [<тип_индекс> индекс]
   {
      get {  // Връща необходимите данни       }
      set {  // Установява необходимите данни  }
   }
}
```

2. **Създаване на индексатор – създаваме обект от този клас и разглеждаме обекта като масив:**

```
<име_на_клас> <име_на_обект> = new <име_на_клас>();
<име_на_обект> [индекс] = <произволен_обект>;
```

Пример: Масив от низове (списък от низове)

Клас System.Collections.ArrayList **– създава динамичен масив от обекти.**

Метод ArrayList.Add **– добавя обект в края на колекцията.**

Свойство ArrayList.Count **– връща текущия брой на елементите в колекцията.**

```
using System;
using System.Collections;
class MyListBox
{
   protected ArrayList data = new ArrayList();

   public object this[int idx]
   {
      get
      {  if (idx>-1 && idx < data.Count)
            return data[idx];
         else
            throw new InvalidOperationException
                                 ("Индексът е извън областта");
      }
```

```
      set
      {  if (idx >-1 && idx < data.Count)
            data[idx] = value;
         else if (idx == data.Count)
            data.Add (value);
         else
            throw new InvalidOperationException
                                 ("Индексът е извън областта");
      }
   }
}
class IndexersApp
{  static void Main()
   {  MyListBox list = new MyListBox();
      list[0] = "Георги"; list[1] = "група 80"; list[2] = "ФКСУ";
      Console.WriteLine ("{0}, {1}, {2}", list[0], list[1], list[2]);
   }
}
```

Пример: Шахматна дъска 8x8 с индексатор с два параметъра: първият се изменя от A до H, а вторият – от 1 до 8.

```
using System;
class Grid
{
   const int NumRows = 8;
   const int NumCols  = 8;
   string[,] cells = new string[NumRows, NumCols+1];
```

```
   public string this[char c, int col]
   {
      get
      {
         c = Char.ToUpper (c);
         if (c < 'A' || c > 'H')
         {
            throw new ArgumentException();
         }
         if (col < 1 || col > NumCols)
         {
            throw new IndexOutOfRangeException();
         }
         return cells[c - 'A', col];
      }
```

```
      set
      {
         c = Char.ToUpper(c);
         if (c < 'A' || c > 'H')
         {
            throw new ArgumentException();
         }
         if (col < 1 || col > NumCols)
         {
            throw new IndexOutOfRangeException();
         }
         cells[c - 'A', col] = value;
      }
   }
}
```

```
class Test
{
   static void Main()
   {
      Grid g = new Grid();
      g['A',1] = "топ";
      g['H',8] = "пешка";
   }
}
```

Обработка на низове

Низ – последователност от символи.

Клас System.String **(псевдоним** string**) представя константен символен низ.**

Класове за обработка на низове:

- StringBuilder
- StringFormat
- StringCollection
- **и др.**

Пример:

```
public string Replace (char oldChar, char newChar);
public string Replace (string oldValue, string newValue);
public string Insert (int startIndex, string value);
public string ToUpper();
using System;
class TestStringApp
{  static void Main (string[] args)
   {  string a = "приход";
      // Замества всяко 'и' с 'е'
      string b = a.Replace ('и', 'е');
      Console.WriteLine (b);                  // преход
      string c = b.Insert (3, "раз");         // преразход
      string d = c.ToUpper ();
      Console.WriteLine (d);                  // ПРЕРАЗХОД
   }
}
```

Пример:

```
public int CompareTo (string strB);
// Сравнение на низове (метод CompareTo) и
// предефиниран оператор ==
if (d==c)                    // или if (d.CompareTo (c) == 0)
   Console.WriteLine ("еднакви");
else
   Console.WriteLine ("различни");
```

Резултати:

различни

Пример:

```
// Replace не променя низа.
// Резултатът се присвоява на променливата.
string q = "приход";
q = q.Replace ('и', 'е');
Console.WriteLine (q);
```

Пример:

```
public string Substring (int startIndex);
public string Substring (int startIndex, int length);
// Слепване и индексиране на низ
string e = "Здравей"+", ";
e += "приятел";
Console.WriteLine (e);                    // Здравей, приятел
string f = e.Substring(1,7);
Console.WriteLine (f);                    // дравей,
for (int i = 0; i < f.Length; i++)
    Console.Write("{0,-3}", f[i]);
Console.WriteLine();                      // д р а в е й ,
```

Пример:

```
public bool StartsWith(string value);
public string Remove(int startIndex, int count);
string g = null;
if (f.StartsWith ("др"))
    g = f.Remove(2,3);
Console.WriteLine (g);              // дрй,
```

Пример:

```
public static string Format(string format, params object[] args);
int x = 16;
decimal y = 3.57m;
string h = String.Format("Артикул {0} се продава за {1:C}", x, y);
Console.WriteLine (h);      // Артикул 16 се продава за 3,57 лв
```

Пример:

```
// Слепване на низ с произволен тип данна
// (всички типове наследяват object.ToString)
string t = "Артикул "+12+" се продава за "+3.45+" лв";
Console.WriteLine (t);
```

Резултати:

Артикул 12 се продава за 3,45 лв

Пример:

```
// String.Format и Console.WriteLine имат последен
// аргумент params object[]
Console.WriteLine("Здравей {0} {1} {2} {3} {4} {5} {6} {7} {8}",
    123, 45.67, true, 'A', 4, 5, 6, 7, '8');
string u=String.Format("Здравей {0} {1} {2} {3} {4} {5} {6} {7} {8}",
    123, 45.67, true, 'A', 4, 5, 6, 7, '8');
Console.WriteLine (u);
```

Резултати:

Здравей 123 45,67 True A 4 5 6 7 8
Здравей 123 45,67 True A 4 5 6 7 8

Пример:

```
public string[] Split (params char[] separator);
char[] separator = new char[]{' ',',','.',':','/','\\','\''};
// Методът String.Split разцепва низ на поднизове чрез
// разделителни символи
string str = "Две хубави очи";
char[] seps=new char[]{' '};
foreach (string ss in str.Split (seps))
    Console.WriteLine (ss);
```

Резултати:

Две
хубави
очи

Недостатък: Split **не работи добре, когато един до друг има няколко разделителя (получава се празен ред).**

Клас System.Text.StringBuilder – **представя променлив символен низ.**

Пример:

```
public String Builder Append (string value);
public StringBuilder AppendFormat (string format,
                                   params object[] args);
using System.Text;
StringBuilder sb =new StringBuilder("Приход"); // Приход
sb.Replace ('и', 'е');                         // Преход
sb.Insert (3, "раз");                          // Преразход
sb.Append (" на средства");          // Преразход на средства
sb.AppendFormat (",{0}:{1}", 123, 45.6789);
                        // Преразход на средства,123:45.6789
sb.Remove (sb.Length-3, 3);
                          // Преразход на средства,123:45.6
Console.WriteLine (sb.ToString().ToUpper());
                    // ПРЕРАЗХОД НА СРЕДСТВА,123:45.6
Console.WriteLine (sb.ToString().ToLower());
                        // преразход на средства,123:45.6
```

Изброим тип enum

Изброим тип – подреден списък от имена, които могат да се използват и отпечатват в програмата.

1. Дефиниране на изброим тип

<модификатор> enum <име_на_типа>
{идентификатор$_0$, ..., идентификатор$_N$}

идентификатор$_0$ **има стойност** 0**.**

Допуска се

идентификатор$_i$ = стойност

2. Въвеждане

<промелива_от_изброим_тип> =
(<име_на_типа>) Enum.Parse (typeof(<име_на_типа>), низ);

Ако низ **не съвпада с** идентификатор$_i$**, се получава изключението** System.ArgumentException**.**

3. Извеждане

<промелива_от_изброим_тип >.ToString ()

4. Присвояване

<промелива_от_изброим_тип> =
<име_на_типа>.идентификатор$_i$;

Пример:

```
using System;
class EnumClass
{   enum Season {Spring, Summer, Autumn, Winter};
    static void Main(string[] args)
    {   Season s=(Season)Enum.Parse(typeof(Season),"Winter");
        Console.WriteLine ("Сезонът е {0}.", s);
        Season first = Season.Spring;
        Console.WriteLine ("Първият сезон е {0}.", first);
        Console.WriteLine ("Сезонът {0} е с номер {1}.",
                           first, (int)first);
    }
}
```

Резултати:

```
Сезонът е Winter.
Първият сезон е Spring.
Сезонът Spring е с номер 0.
```

Интерфейси

Интерфейс – характеризира поведението на класовете, независимо от тяхната йерархия.

1. Дефиниране на интерфейс

[атрибути] [модификатори] interface <име_на_интерфейс> :
списък_от_интерфейси

```
{
    // Декларация на методи
    // Декларация на свойства
    // Декларация на индексатори
    // Декларация на събития
}
```

2. Реализиране на интерфейс – класът трябва да дефинира всички членове на интерфейса.

Пример: Полиморфизъм с интерфейси

```
using System;
interface ISpeaker                          // Интерфейс ГОВОРЯ
{
    void Speak();
}
class Philosopher : ISpeaker                // Философ
{
   private string philosophy;
    public Philosopher (string thoughts)
    {   philosophy = thoughts;   }
    public void Speak()
    {   Console.WriteLine (philosophy);   }
    public void Pontificate()
    {   for (int i = 1; i <= 3; i++)
        Console.WriteLine (philosophy);
    }
}
```

```
class Dog : ISpeaker                        // Куче
{   public void Speak()
    {   Console.WriteLine ("Джаф!");   }
}
class Talking                               // Разговарям
{   static void Main(string[] args)
    {   ISpeaker current;
        current = new Dog();
        current.Speak();
        current = new Philosopher("Аз мисля, следователно "+
                                  "съществувам.");
        current.Speak();
        ((Philosopher)current).Pontificate();
    }
}
```

Резултати:

```
Джаф!
Аз мисля, следователно съществувам.
Аз мисля, следователно съществувам.
Аз мисля, следователно съществувам.
Аз мисля, следователно съществувам.
```

3. Проверка за реализация

а) оператор is **– проверява по време на изпълнение дали даден тип е съвместим с друг тип;**

израз is тип
израз **– от референтен тип**

Стойността на израза е true**, ако:**
израз **≠** null **и**
израз **може да се преобразува до** тип
в противен случай
false

```
Dog dog = new Dog();
if (dog is ISpeaker)
  Console.WriteLine ("Реализира интерфейса ISpeaker");
```

б) оператор as **– конвертира съвместими типове;**
резултатът се запазва в локална променлива и се проверява дали е от валиден тип.

обект = израз as тип
израз**,** тип **– от референтен тип**
еквивалентно на
израз is тип ? (тип)израз : (тип)null

```
Speaker speaker;
speaker = dog as ISpeaker;
if (null != speaker)
   Console.WriteLine("Реализира интерфейса ISpeaker");
else
    Console.WriteLine("Не реализира интерфейса ISpeaker");
```

Интерфейси в .NET Framework

- ICollection
- IEnumerable
- IEnumerator
- IList
- IDictionary
- IDictionaryEnumerator
- IComparer
- IComparable

IEnumerator

```
// Извършва проста итерация на колекцията.
public interface IEnumerator
{
  // Връща текущия елемент в колекцията.
  object Current { get; }

  // Премества номератора към следващия елемент на
  // колекцията.
  bool MoveNext ();

  // Установява номератора в начална позиция, която е
  // преди първия елемент в колекцията.
  void Reset ();
}
```

IEnumerable

```
// Представя номератор, който позволява проста
// итерация на колекцията.
public interface IEnumerable
{
  // Връща номератор, който може да итерира
  // колекцията.
  IEnumerator GetEnumerator ();
}
```

IDictionaryEnumerator

```
// Номерира елементите на речника.
public interface IDictionaryEnumerator
{
  // Връща ключа и стойността на текущия елемент на
  // речника.
  DictionaryEntry Entry { get; }

  // Връща ключа на текущия елемент на речника.
  object Key { get; }

  // Връща стойността на текущия елемент на речника.
  object Value { get; }
}
```

ICollection

```
// Дефинира методи за всички колекции.
public interface ICollection : IEnumerable
{
  // Връща броя на елементите в ICollection.
  int Count {get; }

  // Връща стойност, показваща дали достъпът до
  // ICollection е синхронизиран.
  bool IsSynchronized { get; }

  // Връща обект, който се използва за синхронизиране на
  // достъпа до ICollection.
  object SyncRoot ( get; }

  // Копира елементите от ICollection в Array,
  // стартирайки от определен индекс от Array.
  void CopyTo (Array array, int index);
}
```

IList

```
// Представя колекция от обекти, до които достъпът може
// да се осъществи чрез индекс.
public interface IList
{
  // Добавя елемент към IList.
  void Add (object value);

  // Определя дали IList съдържа определена стойност.
  bool Contains (object value);

  // Премахва първото срещане на определен обект
  // от IList.
  void Remove (object value);
…
}
```

IDictionary

```
// Представя колекция от двойки ключ-стойност.
public interface IDictionary
{
  // Връща IDictionaryEnumerator за IDictionary.
  IDictionaryEnumerator GetEnumerator ();

  // Добавя елемент с даден ключ и дадена стойност към
  // IDictionary.
  void Add (object key, object value);

  // Определя дали IDictionary съдържа елемент с
  // определения ключ.
  bool Contains (object key);

  // Премахва елемента с определения ключ от
  // IDictionary.
  void Remove (object key);
…
}
```

IComparer

```
// Представя метод, който сравнява два обекта.
public interface IComparer
{
  // Сравнява два обекта и връща стойност,
  // показваща единият е по-малък, равен или по-голям
  // от другия.
  int Compare (object x, object y);
}
```

IComparable

```
// Дефинира общ метод за сравняване.
// Така стойностните типове и класовете,
// реализиращи този общ метод, създават метод за
// сравняване на специфични типове.
public interface IComparable
{
  // Сравнява текущия екземпляр с друг обект от
  // същия тип.
  int CompareTo (object obj);
}
```

Пример: Колекция ArrayList – реализира IList; едномерен масив с елементи от тип Object, чийто размер се увеличава автоматично; притежава методи за добавяне, вмъкване и премахване на елемент; лявата граница винаги е 0.

```
using System.Collections;

ArrayList list = new ArrayList();
list.Add ("Петър");
list.Add ("Ана");

IEnumerator myEnumerator = list.GetEnumerator();
while (myEnumerator.MoveNext())
  Console.WriteLine (myEnumerator.Current);
```

Пример: Колекция SortedList – **реализира** IDictionary, ICollection, IEnumerable**; представя колекция от двойки ключ-стойност; елементите на** SortedList **са сортирани спрямо техните ключове в съответствие със специфичната реализация на** IComparer **или** IComparable**; достъпни са чрез ключ или индекс.**

```
using System.Collections;

SortedList bookList = new SortedList();
bookList.Add (123456, "В името на розата");
bookList.Add (234567, "Доктор Живаго");

IDictionaryEnumerator myEnumerator=bookList.GetEnumerator();

while (myEnumerator.MoveNext())
  Console.WriteLine (myEnumerator.Key + "\t" +
                     myEnumerator.Value);
```

Пример: Използване на интерфейса IComparable **с методи за сортиране и търсене в масиви и колекции. Необходимо е предефиниране на метода** CompareTo**.**

```
public int CompareTo (object o);
public static void Sort (Array array, int startindex, int length);
```

```
using System;
class Rational : IComparable
{
  private int numerator, denominator;
  public Rational (int numer, int denom)
  {
    numerator = numer;
    denominator = denom;
    Reduce ();
  }
  private void Reduce()
  {
    int common = Gcd (Math.Abs(numerator), denominator);
    numerator /= common;
    denominator /= common;
  }
```

```
  private int Gcd (int n1, int n2)              // НОД
  {
    while (n1 != n2)
      if (n1 > n2)
        n1 -= n2;
      else
        n2 -= n1;
    return n1;
  }
  public override string ToString()
  {
    return numerator + "/" + denominator;
  }
```

```
  public int CompareTo (object o)
  {
    Rational op2 = (Rational)o;
    int difference = numerator*op2.denominator-
                     op2.numerator*denominator;
    if (difference < 0)
      return -1;
    else if (difference>0)
      return 1;
    else
      return 0;
    }
}
```

```
class TestRational
{
  static void Main()
  {
    Rational x, y;
    x = new Rational (1, 4);
    y = new Rational (1, 3);
    int flag = x.CompareTo(y);
    if (flag < 0) Console.WriteLine ("x<y");
    else if (flag>0) Console.WriteLine ("x>y");
    else Console.WriteLine ("x=y");
    Rational[] list = new Rational[3] {new Rational (1, 3),
                      new Rational (1, 4), new Rational (2, 5)};
    Array.Sort (list, 0, 3);
    foreach(Rational r in list)
      Console.WriteLine (r);
  }
}
```