Масиви

1. **Какво е "масив"?**

Определение: Масивът е структурен тип данни, представляващ крайна редица от еднотипни елементи с пряк достъп до всеки елемент.

Елементите на масивите в C# са номерирани с числата 0, 1, 2, ... N-1. Тези номера на елементи се наричат **индекси**. Броят елементи в даден масив се нарича **дължина на масива**.

Всички елементи на даден масив са от един и същи тип. Това ни помага да представим група от еднородни елементи като подредена свързана последователност и да ги обработваме като едно цяло.

Масивите могат да бъдат от различни размерности, като най-често използвани са **едномерните** и **двумерните** масиви. Едномерните масиви се наричат още вектори, а двумерните – матрици.

## Деклариране и заделяне на масиви

В C# масивите имат фиксирана дължина, която се указва при инициа­лизирането им и определя броя на елементите им. След като веднъж е зададена дължината на масив при неговото създаване, след това не е възможно да се променя.

### а) Деклариране на масив

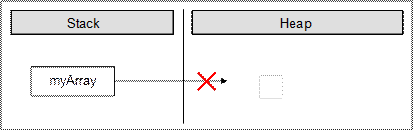
Масиви в C# декларираме по следния начин:

int[] myArray;

В примера променливата **myArray** е името на масива, който е от тип (**int[]**), т.е. декларирали сме масив от цели числа. С **[]** се обозначава, че промен­ливата, която декларираме е масив от елементи, а не единичен елемент.

При декларация на променливата от тип масив, тя пред­ставлява референция (**reference**), която няма стойност (сочи към **null**), тъй като още не е заделена памет за елементите на масива.

Ето как изглежда една променлива от тип масив, която е декларирана, но още не е заделена памет за елементите на масива:



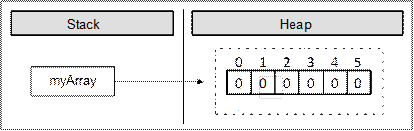
В стека за изпълнение на програмата се заделя променлива с име **myArray** и в нея се поставя стойност **null** (липса на стойност).

### б) Създаване (заделяне) на масив – оператор new

В C# масив се създава с ключовата дума **new**, която служи за заделяне (алокиране) на памет:

int[] myArray = new int[6];

В примера заделяме масив с размер 6 елемента от целочисления тип **int**. Това означава, че в динамичната памет (heap) се заделя участък от 6 после­дователни цели числа, които се инициализират със стойност 0:



Картинката показва, че след заделянето на масива променливата **myArray** сочи към адрес в динамичната памет, където се намира нейната стойност. Елементите на масивите винаги се съхраняват в динамичната памет (в т. нар. **heap**).

При заделянето на масив в квадратните скоби се задава броят на елемен­тите му (цяло неотрицателно число) и така се фиксира неговата дължина. Типът на елементите се пише след **new**, за да се укаже за какви точно елементи трябва да се задели памет.

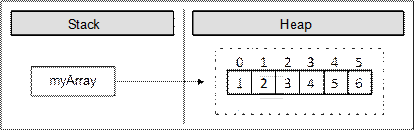
### в) Инициализация на масив. Стойности по подразбиране

Преди да използваме елемент от даден масив той трябва да има начална стойност. В някои езици за програмиране не се задават начални стойности по подразбиране, и тогава при опит за достъпване на даден елемент може да възникне грешка. В C# всички променливи, включително и елементите на масивите имат начална стойност по подразбиране (default initial value). Тази стойност е равна на 0 при числените типове или неин еквивалент при нечислени типове (например **null** за референтни типове и **false** за булевия тип).

Разбира се, начални стойности можем да задавам и изрично. Това може да стане по различни начини. Ето един от тях:

int[] myArray = { 1, 2, 3, 4, 5, 6 };

В този случай създаваме и инициализираме елементите на масива едно­временно. Ето как изглежда масивът в паметта, след като стойностите му са инициализирани още в момента на деклариране:



При този синтаксис къдравите скоби заместват оператора **new** и между тях са изброени началните стойности на масива, разделени със запетаи. Техния брой определя дължината му.

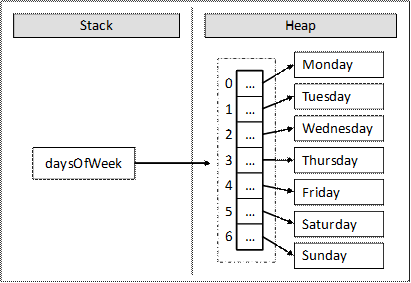
### г) Деклариране и инициализиране на масив – пример

Ето още един пример за деклариране и непосредствено инициализиране на масив:

string[] daysOfWeek = { "Monday", "Tuesday", "Wednesday","Thursday", "Friday", "Saturday", "Sunday" };

В случая масивът се заделя със 7 елемента от тип **string**. Типът **string** е референтен тип (обект) и неговите стойности се пазят в динамичната памет. В стека се заделя променливата **daysOfWeek**, която сочи към участък в динамичната памет, който съдържа елементите на масива. Всеки от тези 7 елемента е обект от тип символен низ (**string**), който сам по себе си сочи към друга област от динамичната памет, в която се пази стойността му.

Ето как е разположен масивът в паметта:



### Граници на масив

Масивите по подразбиране са **нулево-базирани**, т.е. номерацията на елемен­тите започва от **0**. Първият елемент има индекс 0, вторият 1 и т.н. Ако един масив има **N** елемента, то последният елемент се намира на индекс **N-1**.

## д) Достъп до елементите на масив

Достъпът до елементите на масивите е пряк и се осъществява по индекс. Всеки елемент може да се достъпи с името на масива и съответния му индекс (пореден номер), поставен в квадратни скоби. Можем да осъще­ствим достъп до даден елемент както за четене така и за писане т.е. да го третираме като най-обикновена променлива.

Пример за достъп до елемент на масив:

myArray[index] = 100;

В горния пример присвояваме стойност 100 на елемента, намиращ се на позиция **index**.

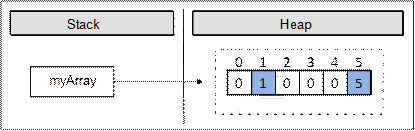
Ето един пример, в който заделяме масив от числа и след това променяме някои от елементите му:

int[] myArray = new int[6];

myArray[1] = 1;

myArray[5] = 5;

След промяната на елементите, масивът се представя в паметта по следния начин:



Както се вижда, всички елементи, с изключение на тези, на които изрично сме задали стойност, са инициализирани с 0 при заделянето на масива.

Масивите могат да се обхождат с помощта на някоя от конструкциите за **цикъл**, като най-често използван е класическият **for** цикъл:

int[] arr = new int[5];

for (int i = 0; i < arr.Length; i++)

{

      arr[i] = i;

}

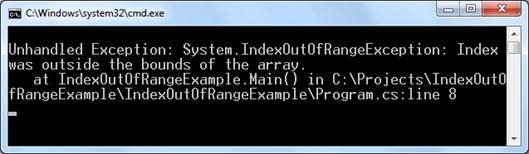
### е) Излизане от границите на масив

При всеки достъп до елемент на масив .NET Framework прави автоматична проверка, дали индексът е валиден или е извън границите на масива. При опит за достъп до невалиден елемент се хвърля изключение от тип**System.IndexOutOfRangeException**. Автоматичната проверка за излизане от границите на масива е изключително полезна за разработчиците и води до ранно откриване на грешки при работа с масиви. Естест­вено, тези проверки си имат и своята цена и тя е леко намаляване на про­изводи­тел­ността, която е нищожна в сравнение с избягването на грешки от тип "излизане от масив", "достъп до незаделена памет" и други.

Ето един пример, в който се опитваме да извлечем елемент, който се намира извън границите на масива:

|  |
| --- |
| **IndexOutOfRangeExample.cs** |
| classIndexOutOfRangeExample  {        static void Main()        {              int[] myArray = { 1, 2, 3, 4, 5, 6 };              Console.WriteLine(myArray[6]);        }  } |

В горния пример създаваме масив, който съдържа 6 цели числа. Първият елемент се намира на индекс 0, а последният – на индекс 5. Опитваме се да изведем на конзолата елемент, който се намира на индекс 6, но понеже такъв не съществува, това води до възникване на изключение:

[](https://introprogramming.info/wp-content/uploads/2011/07/clip_image014.jpg)

### Обръщане на масив в обратен ред – пример

В следващия пример ще видим как може да променяме елементите на даден масив като ги достъпваме по индекс. Целта на задачата е да се подредят в обратен ред (отзад напред) елементите на даден масив. Ще обърнем елементите на масива, като използваме помощен масив, в който да запазим елементите на първия, но в обратен ред. Забележете, че дължината на масивите е еднаква и остава непроменена след първоначал­ното им заделяне:

|  |
| --- |
| **ArrayReverseExample.cs** |
| classArrayReverseExample  {        static void Main()        {              int[] array = { 1, 2, 3, 4, 5 };              // Get array size              int length = array.Length;              // Declare and create the reversed array              int[] reversed = new int[length];                // Initialize the reversed array              for (int index = 0; index < length; index++)              {                    reversed[length - index - 1] = array[index];              }                // Print the reversed array              for (int index = 0; index < length; index++)              {                    Console.Write(reversed[index]+ " ");              }        }  }  // Output: 5 4 3 2 1 |

Примерът работи по следния начин: първоначално създаваме едномерен масив от тип**int** и го ини­циа­ли­зи­ра­ме с цифрите от 1 до 5. След това запазваме дължината на масива в целочислената променлива **length**. Забележете, че се използва свойството **Length**, което връща броя на елементите на масива. В C# всеки масив знае своята дъл­жина.

След това декларираме масив **reversed** със същия размер **length**, в който ще си пазим елементите на оригиналния масив, но в обратен ред.

За да извършим обръщането на елементите използваме цикъл **for**, като на всяка итерация увеличаваме водещата променлива **index** с единица и така си осигуряваме последователен достъп до всеки елемент на масива **array**. Критерият за край на цикъла ни подсигурява, че масивът ще бъде обходен от край до край.

Нека проследим последователно какво се случва при итериране върху масива **array**. При първата итерация на цикъла, **index** има стойност 0. С **array[index]** достъпваме първия елемент на масива **array**, а съответно с **reversed[length - index - 1]**достъпваме последния елемент на новия масив **reversed** и извършваме присвояване. Така на последния елемент на **reversed** присвоихме първия елемент на **array**. На всяка следваща итерация **index** се увеличава с единица, позицията в **array** се увеличава с единица, а в **reversed** се намаля с единица.

## Четене на масив от конзолата

Нека разгледаме как можем да прочетем стойностите на масив от конзолата. Ще използваме **for** цикъл и средствата на .NET Framework за четене от конзолата.

Първоначално, прочитаме един ред от конзолата с помощта на **Console.ReadLine(),**след това преобразуваме прочетения ред към цяло число с помощта на **int.Parse()** и го присвояваме на **n**. Числото **n** ползваме по-нататък като размер на масива.

int n = int.Parse(Console.ReadLine());

int[] array = new int[n];

Отново използваме цикъл, за да обходим масива. На всяка итерация присвояваме на текущия елемент прочетеното от конзолата число. Цикълът ще се завърти **n** пъти т.е. ще обходи целия масив и така ще прочетем стойност за всеки един елемент от масива:

for (int i = 0; i < n; i++)

{

      array[i] = int.Parse(Console.ReadLine());

}

### Проверка за симетрия на масив – пример

Един масив е симетричен, ако първият и последният му елемент са еднакви и същевременно вторият и предпоследният му елемент също са еднакви и т.н. На картинката са дадени няколко примера за симетрични масиви:

clip_image016

В следващия примерен код ще видим как можем да проверим дали даден масив е симетричен:

Console.Write("Enter a positive integer: ");

int n = int.Parse(Console.ReadLine());

int[] array = new int[n];

 Console.WriteLine("Enter the values of the array:");

 for (int i = 0; i < n; i++)

{

      array[i] = int.Parse(Console.ReadLine());

}

 bool symmetric = true;

for (int i = 0; i < array.Length / 2; i++)

{

      if (array[i] != array[n - i - 1])

      {

            symmetric = false;

      }

}

 Console.WriteLine("Is symmetric? {0}", symmetric);

Тук отново създаваме масив и прочитаме елементите му от конзолата. За да проверим дали масивът е симетричен, трябва да го обходим само до средата. Тя е елементът с индекс **array.Length / 2**. При нечетна дължина на масива този индекс е средният елемент, а при нечетна – елементът вляво от средата (понеже средата е между два елемента).

За да определим дали даденият масив е симетричен ще ползваме **булева** променлива, като първоначално приемаме, че масивът е симетричен.

Обхождаме масива и сравняваме първия с последния елемент, втория с предпоследния и т.н. Ако за някоя итерация се окаже, че стойностите на сравняваните елементи не съвпадат, булевата променлива получава стойност **false**, т.е. масивът не е симетричен.

Накрая извеждаме на конзолата стойността на булевата променлива.

## Отпечатване на масив на конзолата

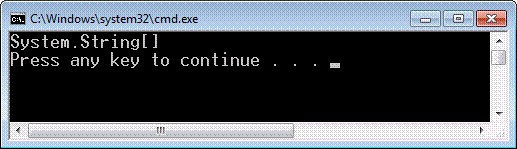
Често се налага след като сме обработвали даден масив да изведем елементите му на конзолата, било то за тестови или други цели.

Отпечатването на елементите на масив става по подобен начин на инициализирането на елементите му, а именно като използваме цикъл, който обхожда масива. Няма строги правила за начина на извеждане на елементите, но често пъти се ползва подходящо форматиране.

Често срещана грешка е опит да се изведе на конзолата масив директно, по следния начин:

|  |
| --- |
| string[] array = { "one", "two", "three", "four" };  Console.WriteLine(array); |

Този код за съжаление не отпечатва съдържанието на масива, а неговия тип. Ето как изглежда резултатът от изпълнението на горния пример:

[](https://introprogramming.info/wp-content/uploads/2011/07/clip_image018.gif)

За да изведем коректно елементите на масив на конзолата можем да използваме **for** цикъл:

string[] array = { "one", "two", "three", "four" };

 for (int index = 0; index < array.Length; index++)

{

      // Print each element on a separate line

      Console.WriteLine("Element[{0}] = {1}", index, array[index]);

}

Обхождаме масива с цикъл **for**, който извършва **array.Length** на брой итерации, и с помощта на метода **Consolе.WriteLine()** извеждаме поред­ния му елемент на конзолата чрез форматиращ стринг. Резултатът е следният:

|  |
| --- |
| **Element[0] = one**  **Element[1] = two**  **Element[2] = three**  **Element[3] = four** |

## Итерация по елементите на масив

Както разбрахме до момента, итерирането по елементите на масив е една от основните операции при обработката на масиви. Итерирайки последо­ва­телно по даден масив можем да достъпим всеки елемент с помощта на индекс и да го обработваме по желан от нас начин. Това може да стане с всички видове конструкции за цикъл, които разгледахме в предната тема, но най-подходящ за това е стандартният **for** цикъл. Нека разгледаме как точно става обхождането на масиви.

### Итерация с for цикъл

Добра практика е да използваме **for** цикъл при работа с масиви и изобщо при индексирани структури. Ето един пример, в който удвояваме стой­ността на всички елементи от даден масив с числа и го принтираме:

|  |
| --- |
| int[] array = new int[] { 1, 2, 3, 4, 5 };   Console.Write("Output: ");  for (int index = 0; index < array.Length; index++)  {        //Doubling the number        array[index] = 2 \* array[index];        //Print the number        Console.Write(array[index] + " ");  }  // Output: 2 4 6 8 10 |

Чрез **for** цикъла можем да имаме постоянен поглед върху текущия индекс на масива и да достъпваме точно тези елементи, от които имаме нужда. Итерирането може да не се извършва последователно т.е. индексът, който **for** цикъла ползва може да прескача по елементите според нуждите на нашия алгоритъм. Например можем да обходим част от даден масив, а не всичките му елементи:

|  |
| --- |
| int[] array = new int[] { 1, 2, 3, 4, 5 };   Console.Write("Output: ");  for (int index = 0; index < array.Length; index += 2)  {        array[index] = array[index] \* array[index];        Console.Write(array[index] + " ");  }  // Output: 1 9 25 |

В горния пример обхождаме всички елементи на масива, намиращи се на четни позиции и повдигаме на квадрат стойността във всеки от тях.

Понякога е полезно да обходим масив отзад напред. Можем да постигнем това по напълно аналогичен начин, с разликата, че **for** цикълът ще започва с начален индекс, равен на индекса на последния елемент на масива, и ще се намаля на всяка итерация докато достигне 0 (включи­телно). Ето един такъв пример:

|  |
| --- |
| int[] array = new int[] { 1, 2, 3, 4, 5 };   Console.Write("Reversed: ");  for (int index = array.Length - 1; index >= 0; index--)  {        Console.Write(array[index] + " ");  }  //Reversed: 5 4 3 2 1 |

В горния пример обхождаме масива отзад напред последователно и извеждаме всеки негов елемент на конзолата.

1. **Основни алгоритми за работа с масиви**

**а) сумиране елементите на масив**

Console.Write("Enter a positive integer: ");

int n=int.Parse(Console.ReadLine());

Console.WriteLine("Enter the values of the array1:");

int[] array = new int[n];

for(int index=0;index<n;index++)

{

array[index] = int.Parse(Console.ReadLine());

}

int Sum = 0;

for (int index = 0; index < n; index++)

{

Sum += array[index];

}

Console.WriteLine(Sum);

**б) преброяване елементите на масив, отговарящи на дадено условие**

Задачата преброява елементите на масива, които са равни на предварително зададено число **к**.

Console.Write("Enter a positive integer: ");

int n=int.Parse(Console.ReadLine());

Console.WriteLine("Enter the values of the array1:");

int[] array = new int[n];

for(int index=0;index<n;index++)

{

array[index] = int.Parse(Console.ReadLine());

}

Console.Write("Enter a positive integer: ");

int k = int.Parse(Console.ReadLine());

int br = 0;

for (int index = 0; index < n; index++)

{

if (array[index] == k)

{

br++;

}

}

Console.WriteLine(br);

**в) намиране на максимален елемент на масив**

Console.Write("Enter a positive integer: ");

int n=int.Parse(Console.ReadLine());

Console.WriteLine("Enter the values of the array1:");

int[] array = new int[n];

for(int index=0;index<n;index++)

{

array[index] = int.Parse(Console.ReadLine());

}

int max = array[0];

for (int index = 1; index < n; index++)

{

if (array[index]>max)

{

max = array[index];

}

}

Console.WriteLine(max);

**г) търсене на елемент в масив**

задачата проверява има ли елемент от масив равен на предварително въведено число **х**.

Console.Write("Enter a positive integer: ");

int n = int.Parse(Console.ReadLine());

Console.WriteLine("Enter the values of the array1:");

int[] array = new int[n];

for (int index = 0; index < n; index++)

{

array[index] = int.Parse(Console.ReadLine());

}

Console.Write("Enter a positive integer: ");

int x = int.Parse(Console.ReadLine());

int i=0;

for (int index = 0; index < n; index++)

{

if (array[i] != x)

{

i++;

}

}

if (i == n)

{

Console.WriteLine("No");

}

else

{

Console.WriteLine("Yes");

}

**Упражнения**

**Задача 1**: Да се напише програма, която създава масив с 20 елемента от целочислен тип и инициализира всеки от елементите със стойност равна на индекса на елемента умножен по 5. Елементите на масива да се изведат на конзолата.

**Задача 2**: Да се напише програма, която чете два масива от конзолата и прове­рява дали са еднакви.

**Задача 3**: Да се напише програма, която въвежда от клавиатурата цяло число 1<n<36, масив от **n** цели елемента и цяло число **x**. Програмата да извежда позицията на първото срещане на **x** в редицата. Ако **x** не се съдържа в редицата, да се изведе подходящо съобщение.

|  |  |  |  |
| --- | --- | --- | --- |
| **Примерен вход** | **изход** | **Примерен вход** | **изход** |
| 6  45 17 -5 -11 17 -9  17 | 2 | 4  18 15 3 85  17 | No |

**Задача 4**: Да се напише програма, която въвежда от клавиатурата цяло число 2<n<30 и масив от **n** цели елемента. Програмата да намира и извежда броя на четните числа от редицата.

|  |  |  |  |
| --- | --- | --- | --- |
| **Примерен вход** | **изход** | **Примерен вход** | **изход** |
| 6  15 28 -5 -12 17 -9 | 2 | 4  19 15 3 85 | 0 |

**Задача 5**: Да се напише програма, която въвежда от клавиатурата цяло число 2<n<25 и масив от **n** реални елемента. Програмата да намира и извежда средноаритметичното на положителните елементи от редицата. Резултатът да се изведе до 2 знака след десетичната запетая.

|  |  |  |  |
| --- | --- | --- | --- |
| **Примерен вход** | **изход** | **Примерен вход** | **изход** |
| 6  1.05 2.8 -5 -12.3 17 9.15 | 7.50 | 4  -1 -12 -3 0 | 0 |

**Задача 6**: Да се напише програма, която въвежда от клавиатурата цяло число 2<n<40 и масив от **n** цели елемента. Програмата да намира и извежда произведението на тези числа от месива, които са кратни на 7. Ако няма такива,да изведе No.

|  |  |  |  |
| --- | --- | --- | --- |
| **Примерен вход** | **изход** | **Примерен вход** | **изход** |
| 5  1 28 -5 -14 17 | -392 | 4  -1 -12 -3 6 | No |

**Задача 7**: Да се напише програма, която въвежда от клавиатурата цяло число 2<=n<=100 и масив от **n** цели елемента. Програмата да намира и извежда максималният елемент на масива и неговият индекс.

|  |  |
| --- | --- |
| **Примерен вход** | **изход** |
| 5  4 -17 -16 -8 16 | 16 4 |

**Задача 8**: Да се напише програма, която въвежда от клавиатурата цяло число 2<=n<=25 и масив от **n** реални елемента. Програмата да намира максималния и минималния елемент, раменя местата им и извежда новополучения масив.

|  |  |
| --- | --- |
| **Примерен вход** | **изход** |
| 5  3.2 8 -4 12 7 | 3.2 8 12 -4 7 |

**Задача 9**: Да се напише програма, която въвежда от клавиатурата цяло число 2<=n<=30 и масив от **n** цели елемента. Програмата да намира и извежда броя на елементите на масива, които са по-големи от средноаритметичната стойност на всичките му елелменти.

|  |  |
| --- | --- |
| **Примерен вход** | **изход** |
| 7  2 6 10 -5 12 1 15 | 4 |

**Задача 10**: Да се напише програма, която въвежда от клавиатурата цяло число 2<n<25 и масив от **n** цели елемента. Програмата да намира и извежда броя на тези елементи, чийто сбор от цифрите е двуцифрено число. Ако няма такива, да се изведе No.

|  |  |  |  |
| --- | --- | --- | --- |
| **Примерен вход** | **изход** | **Примерен вход** | **изход** |
| 5  26 87 -57 238 125 | 3 | 4  23 81 -53 231 | No |

**Задача 11**: Да се напише програма, която въвежда от клавиатурата цяло число 2<n<116 и масив от **n** цели елемента. Програмата да намира и извежда тези елелменти, които са прости числа. Ако няма такива, да се изведе No.

|  |  |  |  |
| --- | --- | --- | --- |
| **Примерен вход** | **изход** | **Примерен вход** | **изход** |
| 6  4 5 43 12 15 7 | 5 43 7 | 4  25 81 56 231 | No |

**Задача 12**: Да се напише програма, която сравнява два масива от тип char лексикографски (буква по буква) и проверява кой от двата е по-рано в лексикографската подредба.

*Забележка:* При лексикографската наредба символите се сравняват един по един като се започва от най-левия. При несъвпадащи символи по-рано е масивът, чийто текущ символ е по-рано в азбуката. При съвпадение се продължава със следващия символ вдясно. Ако се стигне до края на единия масив, по-краткият е лексикографски по-рано. Ако всички съответни символи от двата масива съвпаднат, то масивите са еднакви и никой о тях не е по-рано в лексикографската наредба.