**Основни програмни части и масиви**

**Софтуерът** е съвкупността от цялата информация от инструкции и данни, необходими за работата на всяка електронноизчислителна машина. Употребява се главно в сферата на информационните технологии.

Обикновено инструкциите се задават като съвкупност от алгоритми, групирани като програми с различно предназначение. Освен самите алгоритми, за изпълнението на програмите са необходими и начални данни. Резултатът от действието на даден алгоритъм може да служи като начални данни за стартирането на друг и т.н., обединявайки програмите в едно. В този смисъл все повече се налага и терминът **софтуер**, който исторически се е наложил като антоним на хардуер – физическата част на компютърните системи. Все пак границата между софтуер и хардуер се размива, когато се отчете, че програмното осигуряване има все пак някакви физически носители, от които изчислителната машина чете програмите. Тези носители, както и инструкциите за работа с програмите и тяхната поддръжка (документация), са спомагателни елементи от софтуера.

По принцип хардуерът е неизползваем без програмно осигуряване или софтуер. Най-ясно разликата между софтуер и хардуер се обяснява така: Хардуерът е всяко едно видимо и осезаемо устройство, както самият компютър, така и всяко едно периферно устройство. Софтуерът е само видим, но не и осезаем, когато работи – това са именно програмите.

Със създаването на алгоритми и писането на програми се занимават програмистите. Според нивото на действие на написаните от тях инструкции има език за програмиране на ниско ниво, на средно ниво и на високо ниво. Езиците на високо ниво позволяват лесно редактиране от програмист, и това се нарича изходен код (програмен код). За да се превърне в изпълними инструкции от машината, той се компилира (превежда) към обектен код или машинен език.

**Методи**

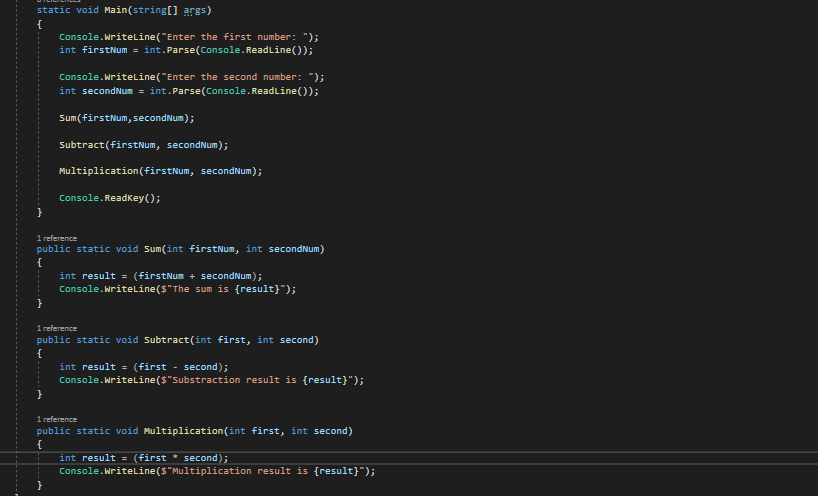
Използването на методи е много полезно не само в C #, но и в цялостното програмиране. Така че притежаването на тези знания е предимство. Не се страхувайте да ги използвате по време на писане на код, те ще направят вашия код по-чист, поддържан, четим и най-вече за многократна употреба.

Какво е метод – съставна част от програмата, която решава даден проблем, може да приема параметри и да връща стойности.

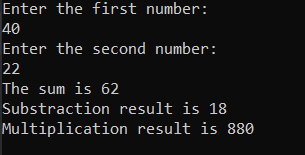
* **Защо да използваме методи**
* По – добро структуриране и по – добра четимост.
* Избягване на повторението на код.
* Преизползване на кода.

Методите могат да имат повече от един параметър. Параметрите трябва да имат свой собствен тип, име и да са разделени със запетая.

***Примерна задача :***



***Резултат:***



//result is 30

* **Деклариране на метод** – регистрирането на метод в програмата, за да бъде разпознаван в останалата част от нея.
* Имплементация – реалното написване на кода, който решава конкретната задача, която метода решава. Този код се съдържа в самия метод и реализира неговата логика.
* **Извикване** – процес на стартиране на изпълнението на вече декларирания и създаден метод, от друго място на програмата.

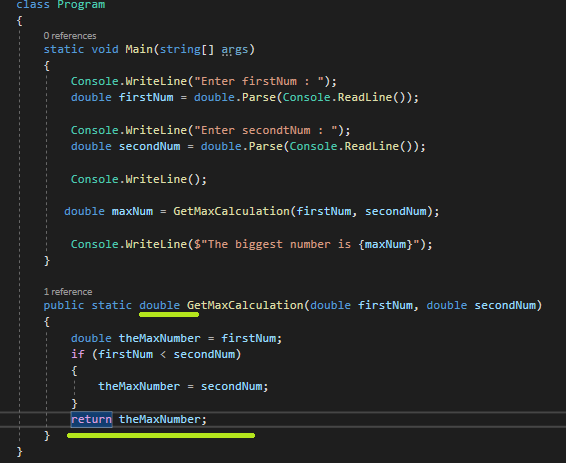
Методът се декларира единствено в рамките на даден клас между отварящите и затварящите скоби.

* **Модификатор** – ключова дума, която дава допълнителна информация на компилатора за даден код. Модификатори за достъп public, private, static, protected.

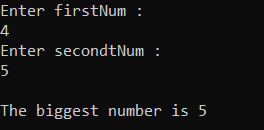
Методът не може да бъде деклариран в тялото на друг метод.

Има методи, които връщат стойност. Декларират се с double вместо с void (например).

***Пример : Намерете по- голямото число.***



***Резултат:***



**Видове променливи**

* **Локални променливи** - за тях се отделя място при използване на функцията, като след приключването й, това място се обявява за свободно. Достъпът до стойностите вече е невъзможен. При използване на друга функция тези стойности не могат да се използват („невидими" са).

Локалните променливи се дефинират вътре в метода.

Не може да има в метода две или повече локални променливи с едно и също име. Ще получим грешка при компилиране.

* **Глобални променливи** - стойностите им са валидни и достъпни в и извън функцията, защото отделянето на място за тях не е свързано с “извикването" (използването) на функцията. Изключение се прави само в случаите, когато функцията използва това име за друга цел. Тогава стойността на глобалната променлива се “замразява". Временно се забравя за съществуването й. След приключване на функцията, тя (стойността на глобалната променлива) може отново да се използва. Компютърът отново си “спомня" за съществуването й. Локалните променливи имат предимство пред глобалните.

Глобалните променливи могат да бъдат достъпвани и в други функции и

методи.

**Видове параметри**

* **Формални** – използват се при описване на функция. Формалните параметри задължително се описват с типа на всеки параметър поотделно.

***Пример*** *-*

Average ( int a, int b, int c ) /\*формални параметри, предавани по стойност – тоест получават стойности, с които да изчисли резултатът от функцията\*/  
{

return **( a + b + c ) / 3.0**;  
}

* **Фактически** – използват се при използването на функция. Фактическите параметри задават стойностите, с които се работи при използване на функцията.

***Пример –***

static void Main(string[] args)

{

double a = double.Parse(Console.ReadLine());

Console.WriteLine(Average(a,33, 12+2));/\*фактически параметри\*/

}

С формалните параметри се описва една функция, а фактическите са тези параметри, които подаваме на функцията да работи с тях.

**Масиви**

* **Какво са масивите?** - неизменна част от повечето езици за програмиране. Те представляват съвкупности от променливи, които наричаме елементи. Всички елементи на даден масив са от един и същи тип, независимо дали е примитивен или референтен. 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** - 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.

int[] myArray = new int[6];

*Така декларираме масив и с ключовата дума new създаваме масив с 6 елемента. Елементите се броят от 0,1,2,3,4,5.*

* **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 задаваме и изрично. Това може да стане по различни начини.

***Примери:***

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

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, която 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.

* **Достъп** -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.

***Пример:***

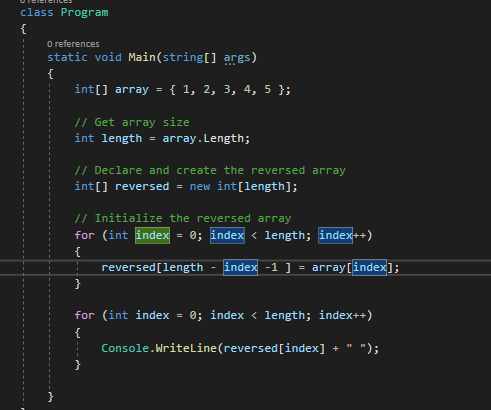
int[] myArray = new int[6];

myArray[1] = 1;

myArray[5] = 5;

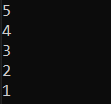
* **Обръщане на масив в обратен ред**

***Пример:***

******

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 с име reversed, имащ същия размер, равен на 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.

***Резултат:***

******

* **Четене на масив от конзолата**

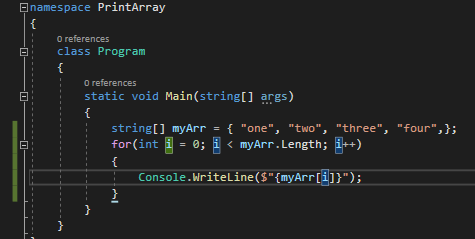
Първоначално прочитаме един ред от конзолата с помощта на Console.ReadLine(), след това преобразуваме прочетения ред към цяло число с помощта на int.Parse() и го присвояваме на n. Числото n ползваме по-нататък като размер на масива.

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

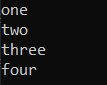
int[] array = new int[n];

* **Отпечатване на масив на конзолата**

***Пример:***

******

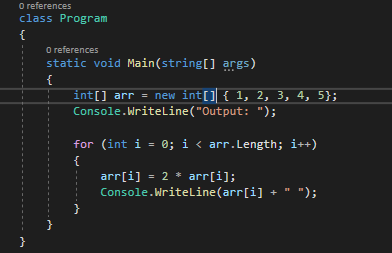
Обхождаме масива с цикъл for, който извършва array.Length на брой итерации, и с помощта на метода Consolе.WriteLine() извеждаме поредния му елемент на конзолата чрез форматиращ стринг.

***Резултатът е следният: ***

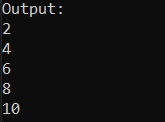
* **Итерация по елементите на масив**

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.

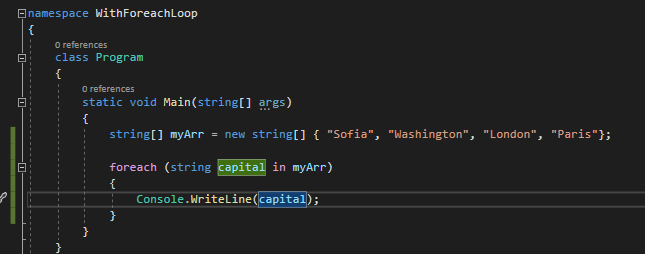
***Пример с for цикъл :***

******

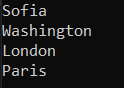
***Резултат:***

******

***Пример с foreach цикъл :***



***Резултат:***

******

**Многомерен масив**

* **Какво е многомерен масив?** - Всеки допустим в C# тип може да бъде използван за тип на елементите на масив. Масивите също може да се разглеждат като допустим тип. По този начин можем да имаме масив от масиви, който ще разгледаме по-нататък. Едномерен масив от цели числа декларираме с int[], а двумерен масив с int[,].

int[,] array;

Такива масиви ще наричаме двумерни, защото имат две измерения или още матрици. Масиви с повече от едно измерение ще наричаме многомерни.

Тримерен масив: int[,,] threeDimensionalArray;

* **Деклариране** - Многомерните масиви се декларират по начин, аналогичен на едномерните.

***Пример:*** int[,] intMatrix;

string[,,] strCube;

int[,] intMatrix = new int[3, 4];

string[,,] stringCube = new string[5, 5, 5];

* **Достъп** - Матриците имат две размерности и съответно всеки техен елемент се достъпва с помощта на два индекса – един за редовете и един за колоните. Многомерните масиви имат различен индекс за всяка размерност.

int[,] matrix = {

{1, 2, 3, 4},

{5, 6, 7, 8},

};

***Как да ги достъпваме ?***

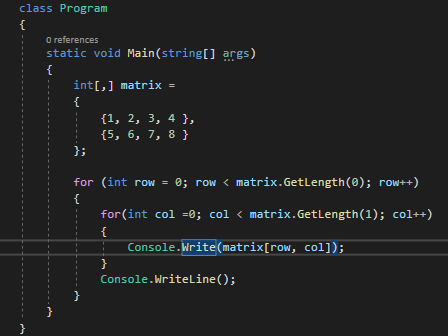
***Пример:***

matrix[0, 0] matrix[0, 1] matrix[0, 2] matrix[0, 3]

matrix[1, 0] matrix[1, 1] matrix[1, 2] matrix[1, 3]

В горния пример виждаме как да достъпим всеки елемент по индекс. Ако означим индекса по редове с row, а индекса по колони с col, тогава достъпът до елемент от двумерен масив има следния общ вид: matrix[row, col]

* **Отпечатване на матрица**

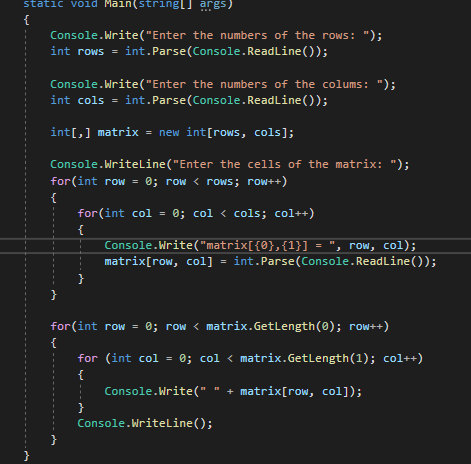
****

Първо декларираме и инициализираме масива, който искаме да обходим и да отпечатаме на конзолата. Масивът е двумерен и затова използваме един цикъл, който ще се движи по редовете и втори, вложен цикъл, който за всеки ред ще се движи по колоните на масива. За всяка итерация по подходящ начин извеждаме текущия елемент на масива като го достъпваме по неговите два индекса (ред и колона).

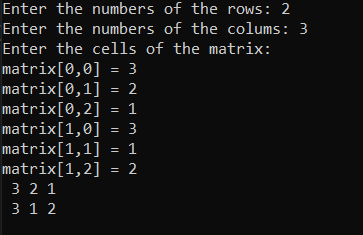
***Резултат:***

******

* **Четене на матрица**



***Резултат:***

******