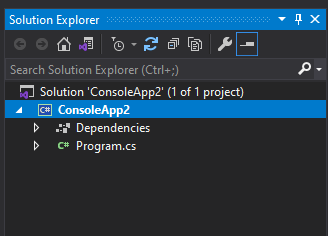
## Visual Studio (VS)

VS-ը ծրագրավորողի արհեստանոցն է, որտեղ կան գործիքներ արահ և հեշտ գրելու համար։

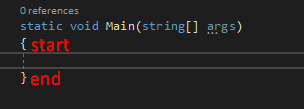
Ստեղծում ենք Console App(Консольное прилажение):



ScreenShot 1

VS-ը պատրաստում է հետևյալ պռոեկտը (այն կարող է տարբերվել ձերից կախված VS-ի վերսիայից)։ C# ծրագիրը գրվում է \_.cs ընդարձակումով ֆայլում, քանի որ C# կոմպիլիատորը կարող է թարգմանել միայն .cs ընդհանրացումով ֆայլերը։

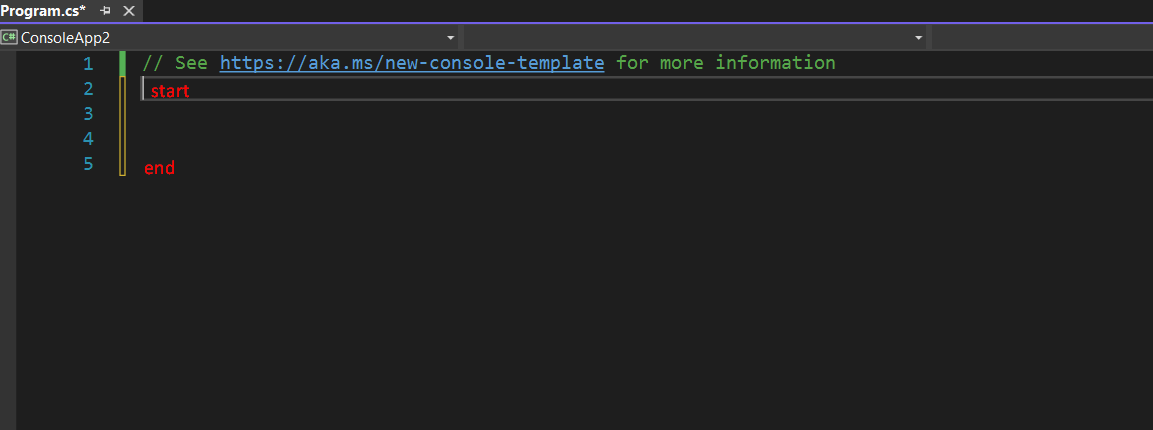
C ընտանիքի լեզուներում (C, C++, Java, C#, ObjC) պարտադիր է ***Main*** անունով ֆունկցիա, քանի որ ցանկացած ծրագիր սկսվում է ***Main***-ի առաջին օպերատորով և վերջանում վերջինով։



ScreenShot 2

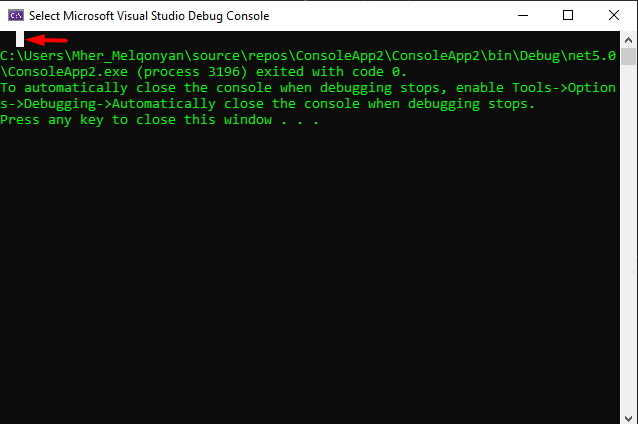
***Main***-ը կոչվում է մուտքի կեր (Enter point):

Visual Studio 2022-ում .Net 6-ից սկսած Main-ը գրվում է ներքին կարգով և ծրագրում չի երևում, ուստի մեր ծրագիրը սկսվում և վերջանում է Program.cs ֆայլի հենց առաջի տողով:

ScreenShot 3

## Console App

***Console app***-ի համար սիստեմը ինքնուրույն ստեղծում է պատուհան։ *Console* պատույանը հորիզունական և ուղղահայաց գծերով բաժանված է ուղղանկյունների։

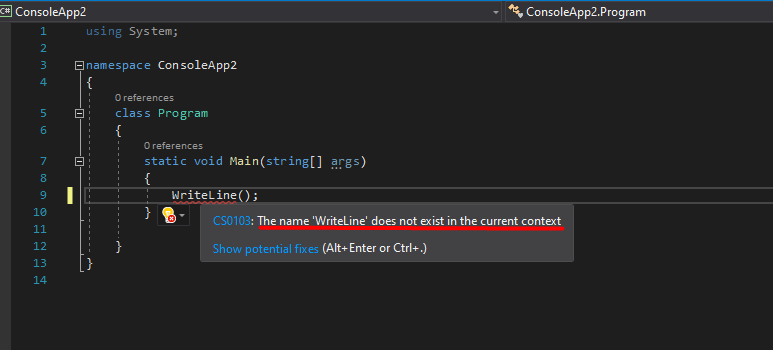


ScreenShot 4

***Console***-ի հետ աշխատելու համար **.Net**-ում ստեղծված է ***Console*** անունով ստատիկ դաս։ Դասի ֆունկցիաներն ու հատկությունները թույլ են տալիս՝

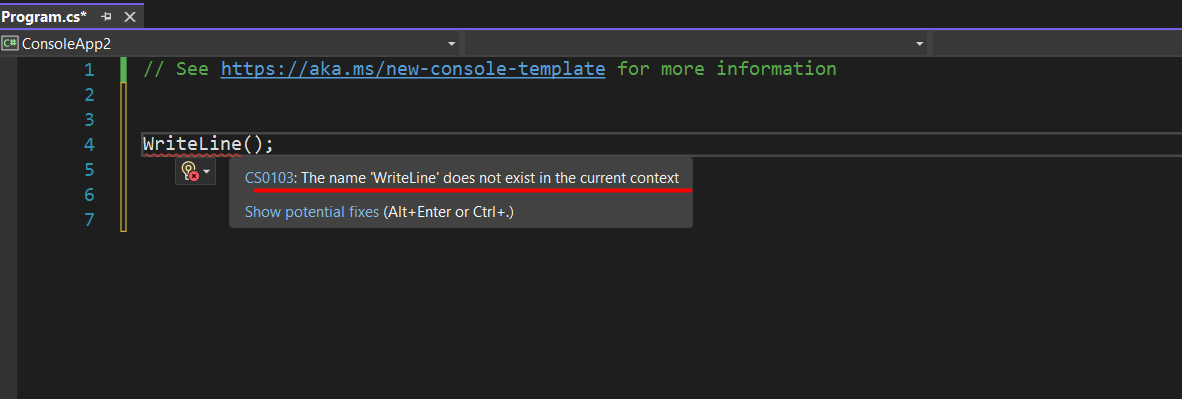
1. Գրել Console-ի վրա
2. Կարդալ ***Console***-ից
3. Փոխել գույները
4. Փոխել չափսերը
5. Փոխել կուրսորի դիրքը, այսինքն տեքստի սկիզբը

**C#**-ում ցանկացած ամփոփ միտք ավարտվում է **;**-ով։

Պատկերացնենք մի իրավիճակ, երբ զրեցակցին ասում ենք օտարերկրյա բառ, որը իրեն ծանոթ չէ։ Իրավիճակը շտկելու համար կարղ ենք ցույց տալ տվյալ բառը բառարանում (այսինքն ցույց տալ հղում այդ բառի գտնվելու վայրի մասին)։ Նույն սցենարը տեղի է ունենում նաև ծրագրավորման մեջ։ Երբ մենք փորձում ենք օգտագործել ինչ որ ֆունկցիաներ, որոնց գտնվելու վայրի մասին ծրագիրը տեղեկություն չունի ստանում ենք սխալ <<Տվյալ անունը չկա այս կոնտեքստում>>

ScreenShot 5

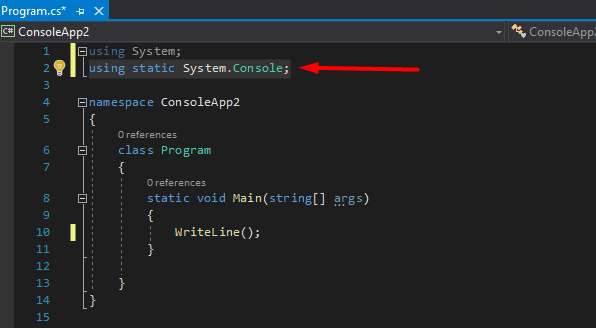
VS 22`



ScreenShot 6

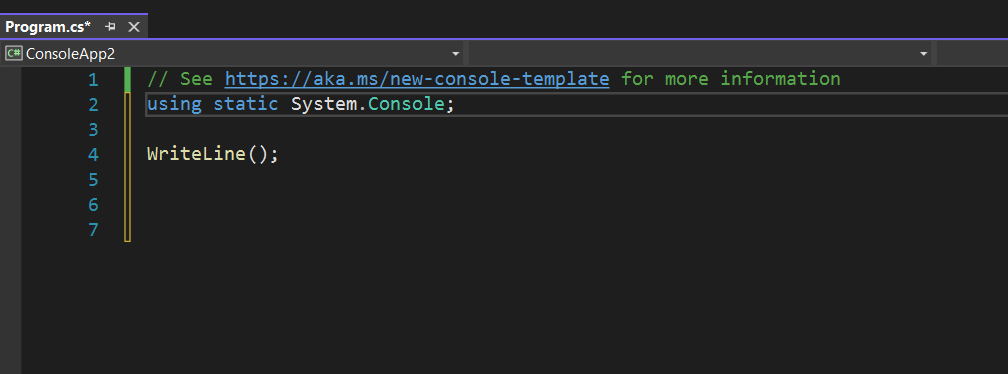
Օգտագործում ենք ***using*** բանալի բառը ֆունկցիայի գտնվելու վայրի մասին տեղեկություն տալու համար։

**using static System.Console;**



ScreenShot 7

VS 22`



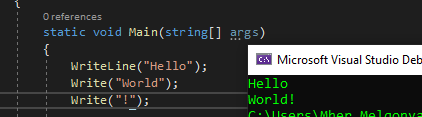
ScreenShot 8

## Ֆունկցիաներ և հատկություններ

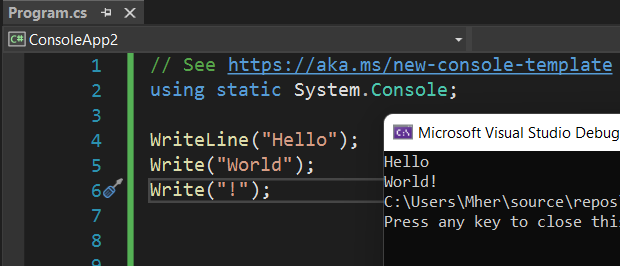
Առաջիկա դասընթացներին օգտագործվելու ենք մի շարք ֆունկցիաներ և հատկություններ խնդիրները լուծելու համար։ Դրանք են՝

### Ֆունկցիաներ

1. Write(“Տեքստ”) - չակերդում գրված տեքստը տպում է Console պատուհանում
2. WriteLine(“Տեքստ”) - չակերդում գրված տեքստը տպում է պատուհանում և կուրսորը տեղափոխում նոր տող
3. SetCursorPosition(x , y) - կուրսորը տեղափոխում է x միավոր աջ և y միավոր ներքև (x և y-ը թվեր են)։
4. ReadLine() - կարդում և մեզ է վերադարձնում կոնսոլում գրված տեքստը
5. ReadKey() - կարդում և մեզ է վերադարձնում կոնսոլում սեխմված ստեղնը(Key, клавиша)
6. ResetColor() - գույները վերադարձնում է սկզբնական կարգավորումներին
7. Clear() - մաքրում է պատուհանը



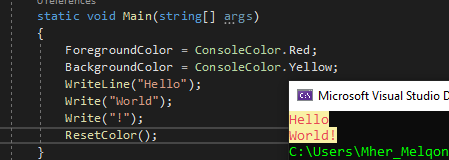
ScreenShot 9



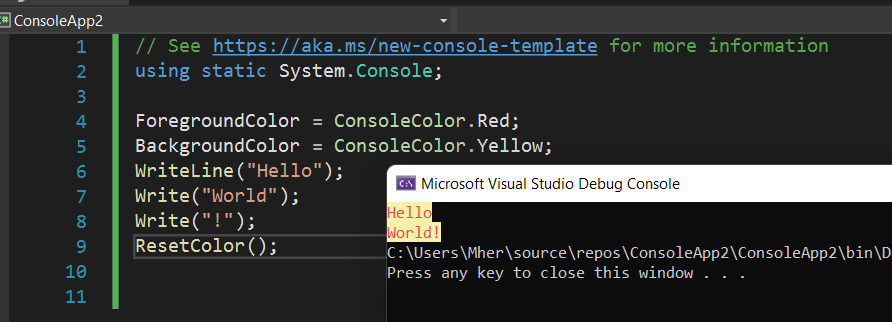
ScreenShot 10

### Հատկություններ

1. WindowHeight - ցույց է տալիս Console պատուհանի բարձրությունը
2. WindowWidth - ցույց է տալիս Console պատուհանի երկարությունը
3. ForegroundColor - պատասխանատու է տեքստի գույնի համար
4. BackgroundColor - պատասխանատու է տեքստի ֆոնի գույնի համար։



ScreenShot 11



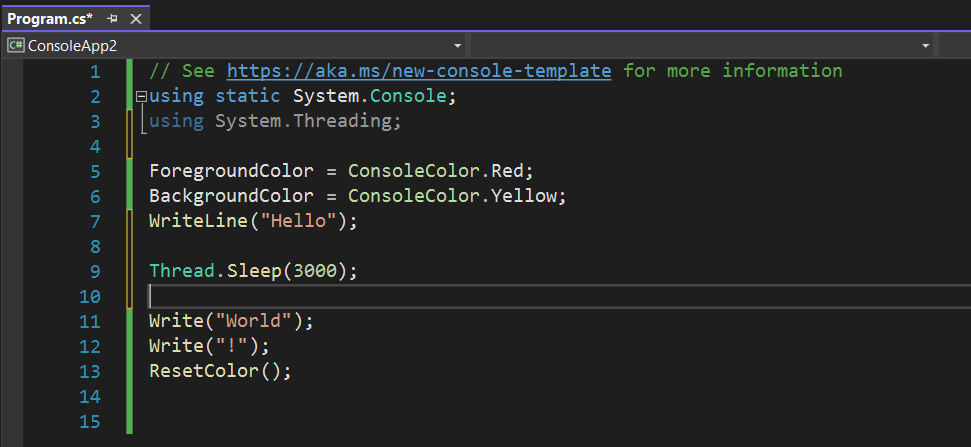
ScreenShot 12

Ծրագիրը կարելի է կանգնացնել հղվելով

**using System.Threading;**

և օգտագործելով՝

***Thread.Sleep(t)*** ֆունկցիան որտեղ t-ն 1000\*վարկյան, այսինքն 3 վարկյան կանգնեցնելու համար ***Thread.Sleep(3000)***;



ScreenShot 13

## Հիշողություն

### Bit

Մեքենայի հիշողությունը միլիարդավոր ***բիթ***երի հաջորդականություն է։ Բիթը հիշողության փոքրագույն մասնիկ է, որի մեջ կարելի է գրել 0 կամ 1։ Իրականում ֆիզիկական բիթի վրա ոչինչ չենք գրում, այլ ընդունում ենք, որ 1 է, երբ բիթով հոսանք է անցնում և 0 է , երբ հոսանք չի անցնում։

### Byte

Պահպանվող ինֆորմացիայի հասցեներ տալու համար միացրեցին 8 բիթ և ստեղծվեց բայթը։ Բայթը հիշողության փոքրագույն մասնիկն է, որը ունի հասցե։ Հասցեները հիշողության մեջ սովորական ամբողջ թվեր են, սկսած 0-ից, բայց 0 հասցեով բայթը փակ է ինչպես ծրագրավորողի այնպես էլ օպերացիոն համակարգի համար։ Ցանկացած փորձ մտնել այդ բայթը կառաջացնի արտակարգ իրադրություն (Exception):

Ինֆորմացիայի չափսը և տիպը որոշելու համար յուրաքանչյուր լեզվի հիմքում դրված է տիպերի համակարգ, բայց քանի որ C#-ը աշխատում է .Net միջավայրում, նրա տիպերի համակարգը իրականում .Net-ինն է։ .Net-ը օգտագործում է՝

1. ***Բազային կամ էլեմենրար տիպեր -*** ամբողջ և իրական թվեր,սիմվոլներ, բուլիան տիպեր
2. ***Բաղադրիչ տիպեր***  - զանգվածներ, տեքստ
3. ***Ծրագրավորղի սեփական տիպեր*** - *class,struct,enum*

## Բազային կամ էլեմենրար տիպեր

**Ամբողջ թվերը** պահելու համար օգտագործում ենք՝

|  |  |  |  |
| --- | --- | --- | --- |
| ․Net | Հիշողություն (byte) | C# | Արժեքներ |
| Byte | 1 | byte | 0\_ից 255 |
| SByte | 1 | sbyte | -128\_ից 128 |
| Int16 | 2 | short | -32000\_ից 32000 |
| UInt16 | 2 | ushort | 0\_ից 65000 |
| Int32 | 4 | int | -2մլրդից 2մլրդ |
| UInt32 | 4 | uint | 0\_ից 4մլրդ |
| Int64 | 8 | long | -9\*1018\_ից 9\*1018 |
| UInt64 | 8 | ulong | 0\_ից 18\*1018 |

Հիշողության մեջ տեղ հատկացնելու ու արժեք պահելու համար, այսինքն փոփոխական հայտարարելու համար գրում ենք՝

*տիպ անուն(type name)*

*Oրինակ՝*

*int age;*

*short t = 25;*

*UInt64 myLong = 151651551;*

Անուններ մտածելիս հարկավոր է իմանալ, որ անունները կարող են պարունակել տառեր, թվեր և ընդգծման սիմվոլ ( \_ ) սակայն չի կարող սկսվել թվով։

**Իրական թվեր՝**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ․Net | Հիշողություն (byte) | C# | Ճշտություն | Տիպի հրաման |
| Single | 4 | float | 6 թիվ ․ից հետո | f |
| Double | 8 | double | 17 թիվ ․ից հետո | d |
| Decimal | 16 | decimal | ․․․ | m |

Իրական թվեր հայտարարելիս անհրաժեշտ է նշել, թե որ տիպին է պատկանում թիվը, այսինքն եթե գրենք *0,56* ծրագիրը այս թիվը կընդունի որպես **double**։ Որպեսզի ծրագրին հասկացնենք , որ նշված թիվը **float** է պետք է թվի վերջում գրել համապատասխան տիպի հրամանը՝ **f** տառը։ Օրինակ՝

float f = 0.65f;

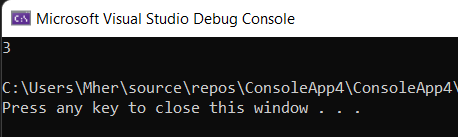
double d = 5.98;

decimal d = 1563.548m;

Եթե ծրագրում **int**-ը բաժանում ենք **int**-ի ծրագիրը հաշվում և վերադարձնում է **int** կորցնելով ստացված թվի կոտորակային մասը:

**int i = 10;**

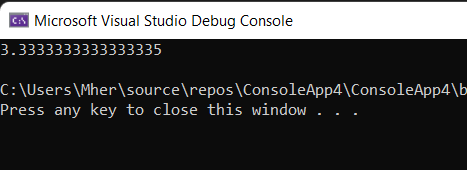
**Console.WriteLine(i/3);**

ScreenShot 14

Պետք է ծրագրին հուշել, որ մենք ցանկանում ենք կոտորակային թիվ ստանալ, դրա համար բաժանման էլեմենտներից մեկը դարձնում ենք **double`**

**double d = 10;**

**Console.WriteLine(d/3);**

ScreenShot 15

**Սիմվոլներ**

Սիմվոլները նախկինում պահվում էին 1 բայթ հիշողության մեջ որտեղ տեղավորվում էր 255 սիմվոլ: Ամեն սիմվոլին համապատասխանում է մի թիվ և ամեն թվին մի սիմվոլ: Այդ աղյուսակը կոչվում է ASCII: Հետագայում վերցրեցին 2 բայթ որտեղ կարող եր պարունակել 65000 սիմվոլ, որից այս պահին զբաղված է մոտ 32000 սիմվոլ:

C#-ում սիմվոլները պահում ենք **char** տիպի մեջ՝

**char comma = ‘,’**;

**char**-երը գրում ենք մեկական չակերդներով ի տարբերություն **string**-ի:

### **Տեքստ**

Հիշողության մեջ տեքստ պահելու համար օգտագործում ենք ***string*** տիպը:

**string srt;**

**string name = " Hunan ";**

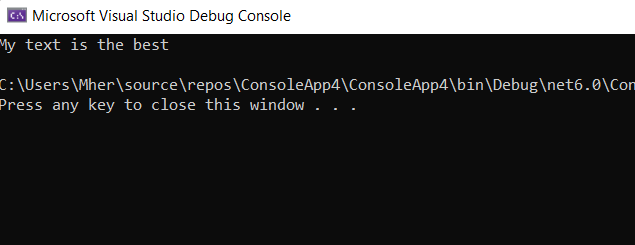
***String***-ների միմիյանց միցնելը կոչվում է կոնկատինացիյա (Concatination):

Օրինակ՝

**string s = " My text ";**

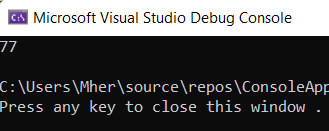
**string s1 = s + " is the best ";**

**Console.WriteLine(s1);**

ScreenShot 16

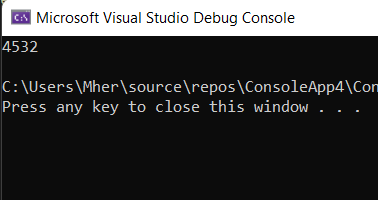
Եթե **WriteLine-**ի մեջ գրենք մաթեմատիկական գործողություն ՝

**Console.WriteLine(45+32);**

ScreenShot 17

Ապա կստանանք 77:Իսկ եթե գրենք՝

**Console.WriteLine("45" +32);**

ScreenShot 18

Կստանանք կոնկատինացիյա քանի որ ծրագիրը **"45"** ընդունում է որպոս տեքստ և կատարում կոնկատինացիյա:

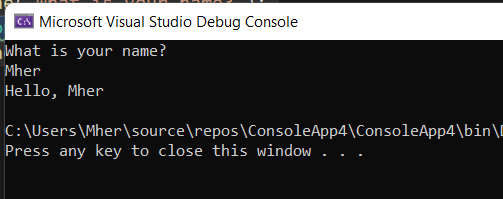
## ***ReadLine()***

ReadLine() ֆունկցիան օգտագործվում է Console պատուհանից ինֆորմացիյա կարդալու համար: Ծրագիրը հասնում է ReadLine() հրամանին և սպասում քանի դեռ օգտագործողը Console-ում մուտք անի ինչ որ բան և սեղմի Enter կոճակը:

**Console.WriteLine("What is your name?");**

**string name = Console.ReadLine();**

**Console.WriteLine("Hello, " + name);**

ScreenShot 19

Ինչպես տեսնում ենք **ReadLine()**-ը վերադարձնում է **string**, որի հետ չենք կարող կատարել մաթեմատիկական գործողություններ (+,- ...): Այդ իսկ պատճառով պետք է կատարենք տիպի փոփխություն (typecast), որը int, double և մնացած պարզ տիպերի համար լինում է ***Parse*** ֆունկցիայով: Օրինակ՝

***int-***ի համար՝  ***int age = int.Parse(“45”);***

***double-***ի համար՝  ***double weight= double.Parse(“102.5”);***

***float …...***

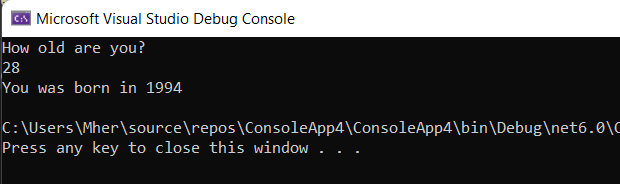
Օրինակ՝

***Console.WriteLine("How old are you?");***

***string ageString = Console.ReadLine();***

***int age = int.Parse(ageString);***

***Console.WriteLine("You was born in " + (2022 - age));***

ScreenShot 20

### **Boolean**

Boolean տիպը կամ C# **bool** տիպը ստանում է 2 արժեք՝ true կամ false (ճիշտ կամ սխալ): Օրինակ՝

**bool b = true;**

**bool isRaining = false;**

### **Տրամաբանական ԵՎ(&&), ԿԱՄ(||)**

Տրամաբանական  **ԵՎ(&&)-**ը համընկնում է լեզվի խոսակցական և-ին:Նույնը վերաբերում է նաև **ԿԱՄ(||):**

Եթե ունենք 2 **bool** արտահայտություն և կատարումենք տրամաբանական միավորում **ԵՎ(&&)** ապա**՝**

|  |  |  |
| --- | --- | --- |
| **a && b** | **true** | **false** |
| **true** | **true** | **false** |
| **false** | **false** | **false** |

Տեսնում ենք, որ միայն **a** և **b-**ն  **true** լինելու դեպքում ենք ստանում **true:**

**Կամ (||)-**ի դեպքում՝

|  |  |  |
| --- | --- | --- |
| **a || b** | **true** | **false** |
| **true** | **true** | **true** |
| **false** | **true** | **false** |

Ինչպես տեսնում ենք ԿԱՄ-ի դեպքում էլ 2 **false-**ի դեպքում ենք **false** ստանում:

**Համեմատություն**

**Ինչպես արդեն բազմիցս տեսել ենք փոփխական հայտարարելիս գրում ենք ՝**

**Int age = 25;**

**Օգտագործելով = (հավասարման) նշանը հիշողության մեջ արժեք պահելու համար:Այլ կերպ ասած փոփոխականին արժեք վերագրելիս: Որի արդյունքում ծրագրավորման մեջ = անվանում ենք վերագրման օպեռատոր:**

**Իսկ ինչպե՞ս համեմատենք երկու արժեքները իրար հետ: Դրա համար օգտագործում ենք ==,իսկ հավասար չէ-ի համար != օրինակ՝**

**WriteLine(5==5); //true**

**WriteLine(5!=5); //false**

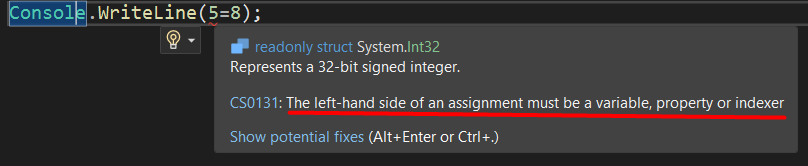
**WriteLine(5!=8); //true**

**Ինչպես նաև՝**

**մեծ՝ > ,փոքր՝ < ,մեծ կամ հավասար՝ >= , փոքր կամ հավասար՝ <= :**

**Հիշեք, որ համեմատելիս = օգտագործելիս կստանաք սխալ:**

**WriteLine(5=8);**



## Ճյուղավորման արտահայտություններ

C#-ում ծրագիրը հնարավոր է ճյուղավորել 4 եղանակով՝

**if**

**if else**

**Հարցական գործողություն(Ternary operatior )**

**switch**

**If (եթե)**

**Գրելաձևը՝**

**if(պարամետր)**

**{**

**Օպերատոր 1;**

**Օպերատոր 2;**

**…...............**

**}**

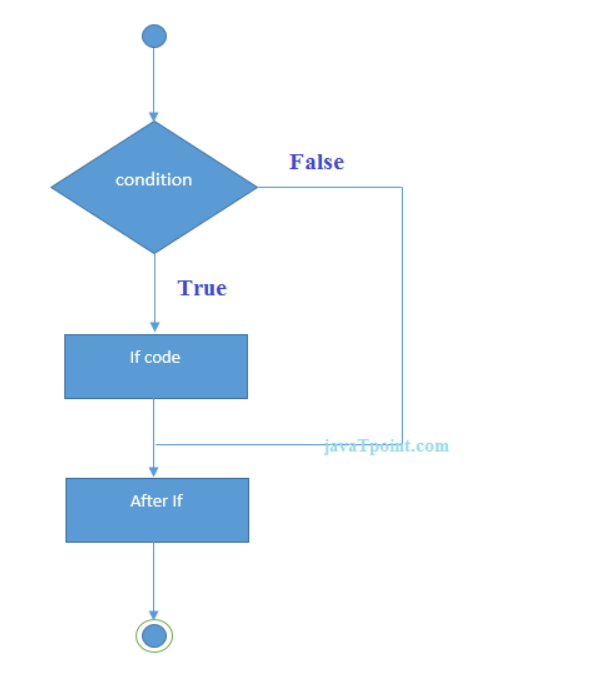
**Օրինակ՝**

**if (5 > 10)**

**{**

**Console.WriteLine("5>10");**

**}**

ScreenShot21

**if ...else(եթե...հակարակ դեպքում)**

**Գրելաձևը՝**

**if(պարամետր)**

**{**

**Օպերատոր 1;**

**Օպերատոր 2;**

**…...............**

**}**

**else**

**{**

**Օպերատոր 3;**

**Օպերատոր 4;**

**…...............**

**}**

**Օրինակ՝**

**int x = 0;**

**if (x > 10)**

**{**

**Console.WriteLine("x>10");**

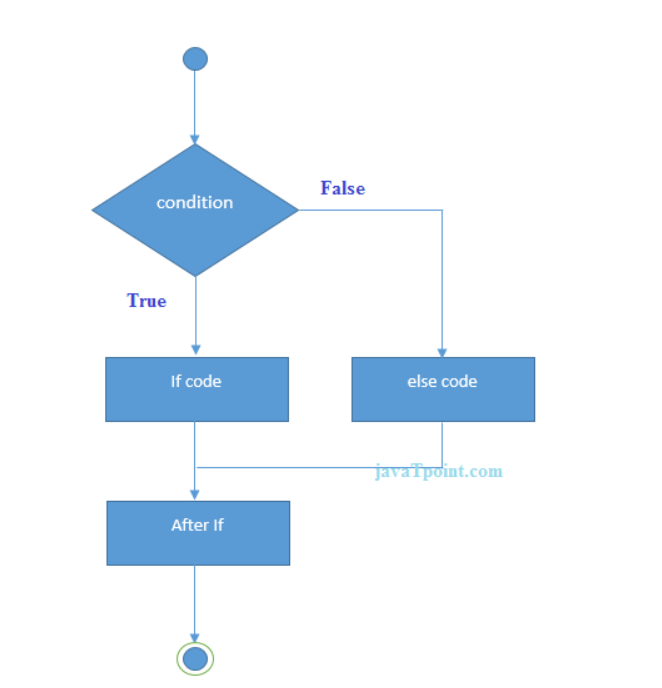
**}**

**else**

**{**

**Console.WriteLine("x<=10");**

**}**

ScreenShot22

Եթե if-ի մեջ գրված է մեկ արտահայտություն(օպերատոր) կարելի է բաց թողնել ձևավոր փակագծերը:

**if (5 > 10)**

**Console.WriteLine("5 > 10");**

կամ

**int x = 0;**

**if (x > 10)**

**Console.WriteLine("x > 10");**

**else**

**Console.WriteLine("x <= 10");**

Լինում են դեպքեր, երբ **եթե**-ի **հակառակ դեպք-**երը կարող են մի քանիսը լինել:

**int x = 0;**

**if (x > 10)**

**{**

**Console.WriteLine("x>10");**

**}**

**else**

**{**

**if (x > 5)**

**{**

**Console.WriteLine("x>5");**

**}**

**else**

**{**

**Console.WriteLine("x<=5");**

**}**

**}**

Եվ ինչպես արդենք գիտենք , եթե if-ի մեջ ունենք մի արտահայտություն ապա կարող ենք ձևավոր փակագծերը բաց թողնել, իսկ if-ը համարվում է մեկ արտահայտություն արդյունքում ստանում ենք ՝

**int x = 0;**

**if (x > 10)**

**{**

**Console.WriteLine("x>10");**

**}**

**else if (x > 5)**

**{**

**Console.WriteLine("x>5");**

**}**

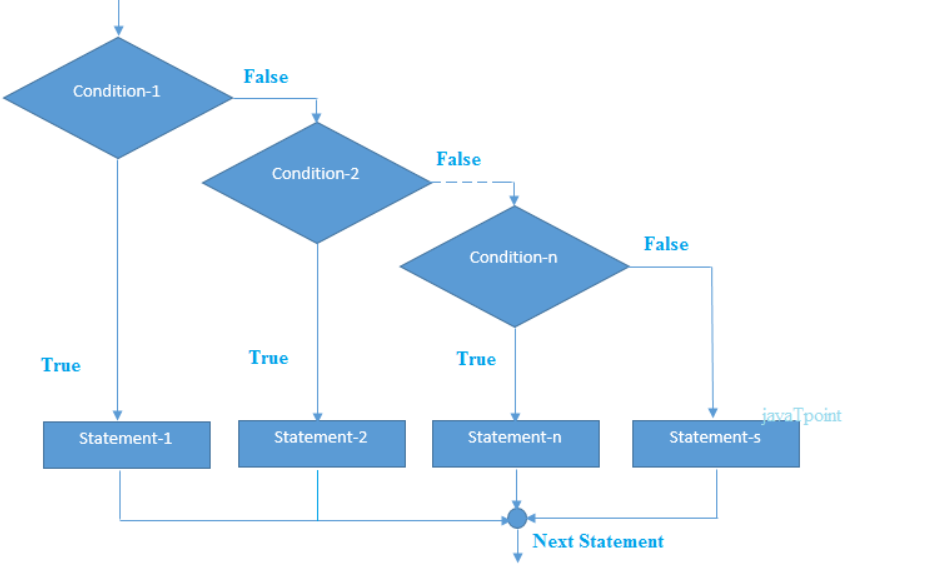
**else**

**{**

**Console.WriteLine("x<=5");**

**}**

**else if** նոր օպերատոր:

ScreenShot23

**Հարցական գործողություն(Ternary operatior )**

Հարցական գործողությունը if..else-ից տարբերվում է նրանով, որ այն վերադարձնում է արժեք և հնարավոր է օգտագործել WriteLine-ի մեջ կամ վերագրել ինչ որ փոփոխականի:Գրելաձևը՝

**Պայման ? Արժեք1 : Արժեք2**

Օրինակ՝

**int x = 0;**

**Console.WriteLine(x > 10 ? "x>10" : "x<=10");**

**int y = (x > 10) ? (x \* x) : (x \* 10);** (փակագծերը դրված ենք արտահայտության մասերը

ակնհայտ դարձնելու համար և պարտադիր չեն)

Հնարավոր է նաև բազմակի օգտագործում, օրինակ որևէ Արժեքի փոխարեն կարող ենք ունենալ ևս մեկ հարցական գործողություն:

**Պայման ? Արժեք1 : (Պայման ? Արժեք2 : Արժեք3)**

Օրինակ՝

int x = 0;

Console.WriteLine(**x > 10 ? "x>10" : x>5?"x>5":"x<=5"**);

**Switch**

**switch-**ը օգտագործվում է մեծ քանակով հավասարումներ ստուգելու ժամանակ(հիմնական 5 և ավելի): Գրելաձևը ՝

switch(արժեք)

{

case համեմատվող1: օպերատոր1;

օպերատոր2;

…..................

break;

case համեմատվող2: օպերատոր3;

օպերատոր4;

…..................

break;

default: օպերատոր1;

օպերատոր2;

…..................

break;

}

Ծրագիրը արժեքը համեմատում է համեմատվող1..ու մնացածի հետ և հավասարության դեպքում կատարում համապատսխան օպերատորները:

**string name = Console.ReadLine();**

**switch (name)**

**{**

**case "A":**

**Console.WriteLine("Armen");**

**break;**

**case "B":**

**Console.WriteLine("Babken");**

**break;**

**default:**

**Console.WriteLine("No Name");**

**break;**

**}**

## Ընդհատման օպերատոր

C#-ում կամ 4 ընադհատման օպերատորներ՝

* break
* continue
* return;
* **return արտահայտությեուն**;

**break** կարելի է օգտագործել միայն **switch-**ում և ցիկլի օպերատորներում։ **break**-ը ընդհատում է արտաքին օպերատորի աշխատանքը և ղեկավարումը փոխանցում անմիջապես հաջորդ օպերատորին։

**continue-**ն կարելի է օգտագործել ցիկլի օպերատոևում։ Այն բաց է թողնում ցիկլի տվյալ քայլը։

**return-**ը օգտագործվում է միայն void վերադարձնող ֆունկցիաների մեջ։ Այն ընդհատում է ֆունկցիան և ղեկավարումը փոխանցում կանչող ֆունկցիային։

**return արտահայտությեուն**-ը կարելի է օգտագործել արժեք վերադարձնող ֆունկցիաներում, որը բացի ընդհատելուց նաև արժեք վերադարձնելու միակ միջոցն է։

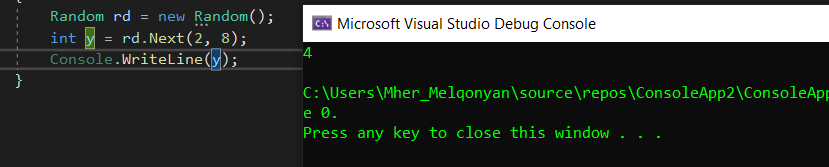
## Պատահական թվեր

C#-ում պատահական թիվ հայտարարելու համար մեկ անգամ ծրագրի սկզբում ստեղծում ենք Random դասի օբյեկտ՝

**Random rd = new Random();**

Ստեղծելուց հետո հնարավոր է ստանալ պատահական թիվ 4 եղանակով՝

1. **int x = rd.Next();** (վերադարձնում է պատահական թիվ **[0,4մլրդ]** միջավայրից)
2. **int x = rd.Next(a);** (վերադարձնում է պատահական թիվ **[0,a)** միջավայրից)
3. **int x = rd.Next(a,b);** (վերադարձնում է պատահական թիվ **[a,b)** միջավայրից)
4. **double d = rd.NextDouble();** վերադարձնում է պատահական թիվ **[0,1)** միջավայրից)

ScreenShot 24

## Կրկնությունների ծրագրավորում (ցիկլի օպերատորներ)

C#-ում ցիկլի օպերատորները 4-ն են՝

1. **while**
2. **do while**
3. **for**
4. **foreach**

### While

**while**-ը առաջի ցիկլի օպերատորն է, որին կծանոթանանք։ Գրելաձևը՝

**while(պայման)**

**{**

**Արտահայտություն 1;**

**Արտահայտություն 2;**

**․․․․․․․․**

**}**

Ծրագիրը հասնում է **while-**ին ստուգում է պայմանը, եթե պայմանը true է, ապա կմտնի ձևավոր փակագծերի մեջ և կկատարի արտահայտորթյունները, հակառակ դեպքում կանցնի **while-**ից անմիջապես հետո գրված օպերատորին և այսպես ցիկլի ամեն քայլին։

Օրինակ՝

**while (true)**

**{**

**Console.WriteLine("Hello");**

**}**

Այս ծրագիրը Console պատուհանում անվերջ կտպի **Hello** բառը, քանի որ պայմանը միշտ true է։

Դիտարկենք այլ օրինակ, եթե ցանկանում ենք տպել 1-ից 10 թվերը՝

**int x = 1;**

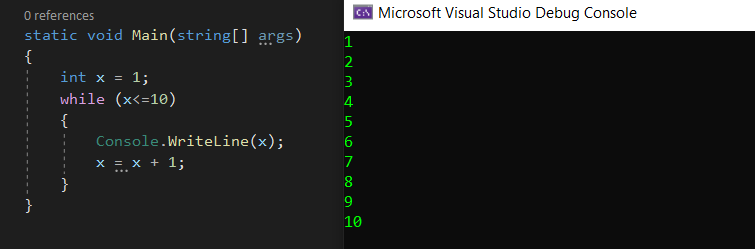
**while ( x <= 10 )**

**{**

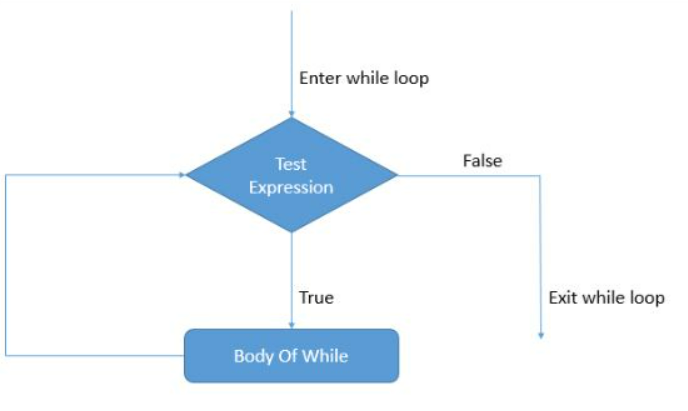
**Console.WriteLine(x);**

**x = x + 1;**

**}**

ScreenShot 25

Ծրագրում ստեղծում ենք int x = 1; Ստուգում ենք, որ այն մեծ չլինի 10-ից, տպում ենք էկրանին և ավելացնում մեկով։Երբ x-ի արժեքը ավելացումների արդյունքում մեծ է լինում 10-ից ծրագիրը սուրս է գալիս ցիկլից և ավարտվում։

ScreenShot 26

#### Ինկրեմենտ և դեկրեմենտ

Ինչպես տեսանք հնարավոր է փոփոխականի արժեքը փոխել 1-ով գրելով

**x = x + 1;**

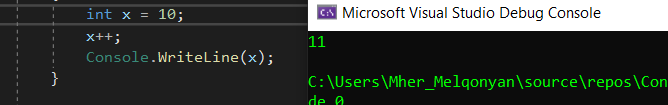
Հավասարման աջ մասում կարդում ենք x-ի արժեքը հիժողությունից, ավելացնում 1-ով և հետ գրում հիշողության մեջ օգտագործելով հավասարման նշանը(վերագրում)։

Այս գրելաձը հեշտացնելու համար ստեղծվեցին ++ (ինկրեմենտ) և --(դեկրեմենտ) օպերատորները, որոնք օգտագործվում են փոփոխականը մեկով ավելացնելու կամ պակասեցնելու համար։

**int x = 10;**

**x++;**

**Console.WriteLine(x); //11**

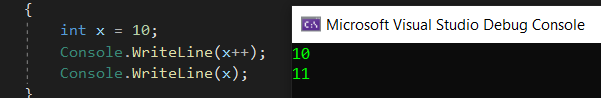
ScreenShot 26

Կարող ենք օգտագործել 2 եղանակով՝ **x++ կամ ++x։** Տարբերությունը այն է, որ երբ արտահայտությունը օգտագործում ենք այլ ֆունկցիայի մեջ ինչպիսին է WriteLine()-ը առաջինի դեպքում տպում է էկրանին նոր ավելացնում արժեքը։

**int x = 10;**

**Console.WriteLine(x++); //10**

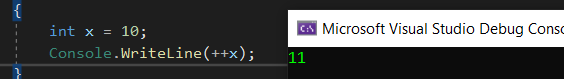
**Console.WriteLine(x); //11**

ScreenShot 27

Իսկ ++x- դեպքում ավելացնում է արժեքին նոր տպում էկրանին՝

**int x = 10;**

**Console.WriteLine(++x);**

ScreenShot 28

Դեկրեմենտի համար նույնպես գործում են նույն կանոնները։

Իսկ եթե ուզում ենք ավելացնել ոչ թե 1 այլ ուրիշ թիվ կամ ցանկանում ենք կատարել այլ մաթեմատիկական գործողություն։ Այդ դեպքերի համար նույնպես կա կրճատ գրության ձևեր՝

**x = x + 10;** կարող ենք գրել **x+=10;**

**x = x - 8;՝ x -= 8;** և այսպես բոլոր մաթեմատիկական գործողությունների համար։

### do... while

**do while** ցիկլի քայլերը շատ նման են **while-**ին միայն մի տարբերությամբ, որ **do while-**ի դեպքում ցիկլի մեջ եղած արտահայտությունները գոնե մի անգամ կատարվում են։

**do**

**{**

**Արտահայտություն 1;**

**Արտահայտություն 2;**

**․․․․․․․․**

**} while ( պայման);**

Օրինակ՝

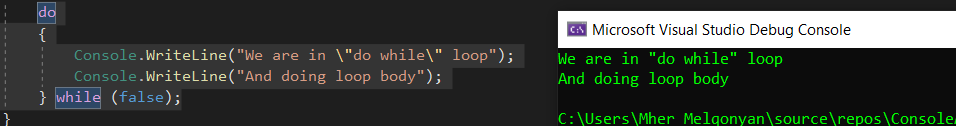
**do**

**{**

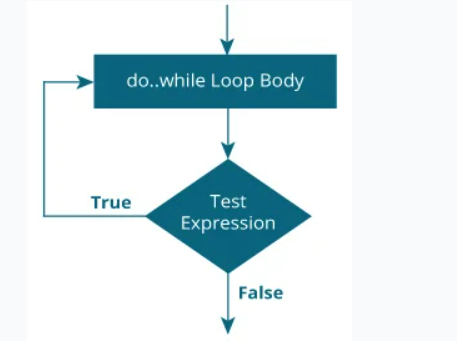
**Console.WriteLine("We are in \"do while\" loop");**

**Console.WriteLine("And doing loop body");**

**} while (false);**

ScreenShot29

Ծրագիրը մտնում է ցիկլ կատարում մարմնի արտահայտություննեը հետո ստուգում պայմանը և քանի որ պայմանը false է դուրս է գալիս ցիկլից։

ScreenShot 30

### for

for-ը հաջորդ ցիկլն է, որը ուսումնասիրելու ենք։Եկեք վերհիշենք while-ի օրինակը, որտեղ ցիկլին քանակը որոշելու համար հայտարարեցինք փոփոխական՝

**int t = 0;**

**while ( t < 10)**

**{**

**Console.WriteLine(t);**

**t++;**

**}**

Ծրագիրը կկատարի 10 պտույտ և էկրանին կտպի 0-ից 9 թվերը։ Սակայն մենք տեսնում ենք որ ցիկլի քանակը կարգավորող մասնիկները ցրված են և կարող են լինել ծրագրի տարբեր հատվածներում ինչը կառաջացնի դժվարություն առաջին հայացքից հասկանալ քանի պտույտ է կատարելու ծրագիրը։ Դրա համար ստեղծվել է **for**-ը։Գրելաձևը՝

**for (Արտ․ 1; Bool արտ․; Արտ․ 3)**

**{**

**Արտահայտություն 4;**

**Արտահայտություն 5;**

**․․․․․․․․․․․**

**}**

Վերը նշված օրինակը **for-**ով կլինի

for (int i = 0; i < 10; i++)

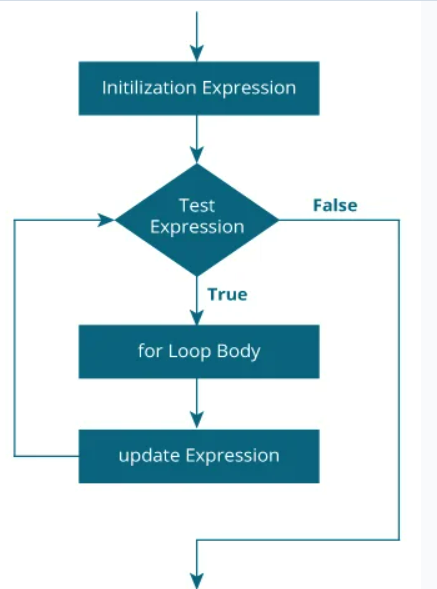
{

Console.WriteLine(i);

}

Ծրագիրը հասնելով **for-**ին

1. մեկ անգամ կատարում է **Արտահայտություն 1**-ը և հայտարարում փոփխական
2. Ստուգում է պայմանը կամ **Bool արտ․**
3. Եթե **true** է կատարում է մարմինը **արտահայտություն 4, 5 ․․․․**
4. Հետո կատարում **արտահայտություն 3**
5. Վերադառնում **2 կետին**
6. Սխալի դեպքում դուրս է գալիս ցիկլից։

ScreenShot 31

**for**-ի ցանկացած արտահայտություն կարելի է բաց թողնել սակայն ;-ները նշել պարտադիր է:

ScreenShot 32

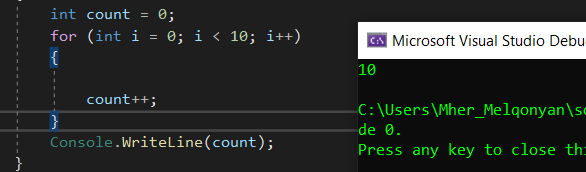
**for (int i = 0; i < 10; i++)**

**{**

**Արտահայտություն**

**}**

Ինչպես արդեն գիտենք **Արտահայտություն**ը կկատարվի 10 անգամ։

ScrrenShot32

Իսկ ինչ կլինի, եթե **Արտահայտությունը** լինի ցիկլ(for):

**for (int i = 0; i < 10; i++)**

**{**

**for (int j = 0; j < 10; j++)**

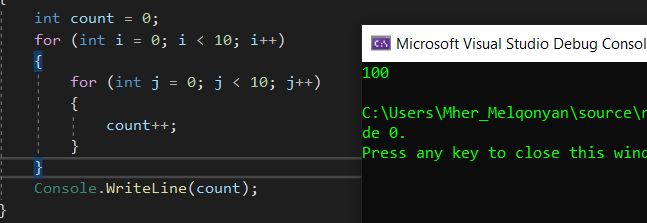
**{**

**Արտահայտություն**

**}**

**}**

Կարող ենք հեշտությամբ հաշվել ու համոզվել, որ այս դեպքում ծրագիրը կաշխատի 10\*10=100 անգամ։

ScrrenShot33

Օրինակի համար կարող ենք նկարել քառակուսի կոնսոլում։

**for (int i = 0; i < 10; i++)**

**{**

**for (int j = 0; j < 10; j++)**

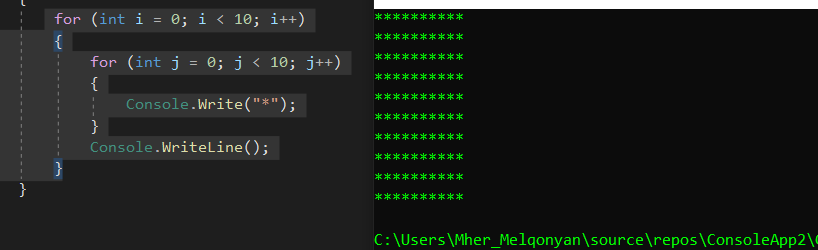
**{**

**Console.Write("\*");**

**}**

**Console.WriteLine();**

**}**

ScrrenShot34

## Զանգվածներ

Հաճախ ծրագրերում գործ ենք ունենում խոշոր ծավալի ներքուստ փոխկապակցված տվյալների հետ: Այսպիսի տվյալներ պահելու համար օգտագործում ենք զանգված(Array, массив):

Զանգվածներ բաժանվում են 2 մասի՝

1. Կամպակցված
2. Սովորական

2. Սովորական զանգվածը նույնատիպ էլեմնտների շարք է, որոնք հիշողության մեջ դասավորված են հաջորդաբար: Այս զանգվածի վրա i-րդ էլեմենտը գտնելու համար կատարում ենք հասարակ գումարման գործողություն և հասնում նշված էլեմենտին մեկ քայլով:

1.Կապակցված զանգվածը նույնատիպ էլեմենտների շարք է,որոնք ցրված են հիշողության մեջ, բայց յուրաքանչյուր էլեմենտ պարունակում է ինֆորմացիա 1 կամ ավել այլ էլեմենտների գտնվելու տեղի մասին: C#-ում բոլոր զանգվածները ժառանգ են Array դասից: Դա նշանակում է զանգվածը նույնպես դաս է, հետեվաբար հղումային տիպ է (Reference type), այսինքն պահվում դինամիկ հիշողության մեջ՝ heap-ում:

Զանգված ստեղծելու համար պարտադիր է new-ի առկայությունը: Գրելաձև՝

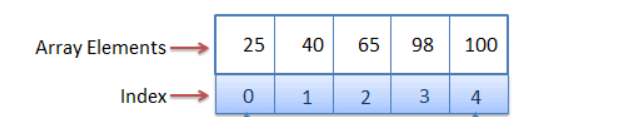
**Տիպ[] անուն = new Տիպ[քանակ];**

Օրինակ՝

**int[] arr = new int[5];**

Հիշողության մեջ կստեղծի 5 երկարությամբ թվերի զանգված և կտա default արժեքներ, որը int-ի համար 0-ն է, որը մենք կարող ենք փոփոխել մեր արժեքներով:

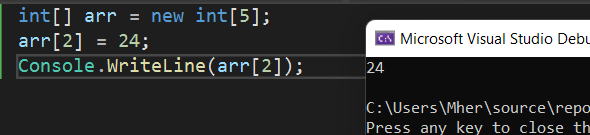
Երբ ստեղծում ենք զանգված ստանում ենք նմանատիպ պատկեր, որտեղ ցանկացած վանդակ ունի իր համապատասխան համարը՝ ինդեքսը: Ինչպես տեսնում ենք համարակալելիս սկսում ենք 0-ից հետևաբար 5 երկարությամբ զանգվածի վերջի էլեմենտը կլինի 4-րդ ինդեքսում:

Screenshot 35

ԵՎ այսպես զանգվածի էլեմենտները փոփոխելու կամ կարդալու համար օգտագործում ենք ինդեքսները, օրինակ՝

arr[2] = 24;

նշանակում է եկրերդ ինդեքսի արժեքը դարձրեցինք 24:

ScreenShot 36

Զանգվածը ինիցիալիզացնելու կամ արժեքներ տալու համար կա մի քանի եղանակ:

Առաջինը վերոնշյալ տարբերակն է, երբ ինդեքսով գնում ենք հիշողության համապատասղան տեղը և տալիս մեր արժեքը: Եկրերդ տարբերակը մի տողով ինիցիալիզացիան է՝

**int[] arr = new int[3] { 15,24,-6};**

Ինչպես տեսնում ենք թվերի քանակը հավասար է զանգվածի երկարությանը: Դա պարտադիր պայման է, հակառակ դեպքում կստանանք սխալ:

Մի տողով ինիցիալիզացիա անելիս կարող ենք **new int[3]** բաց թողնել՝

**int[] arr2 = { 15,24,-6,-256,0,45};**

Ծրագիրը կոնտեքստից կհասկանա և կստեղծի 6 երկարությամբ զանգված և կինիցիալիզացնի նշված արժեքներով:

Հիմա փորձենք տպել զանգվածի բոլոր էլեմենտները:Այսինքն պետք է ունենանք ցիկլ, որը կլինի 0->4 թվերը, այսինքն՝

**int[] arr = new int[5];**

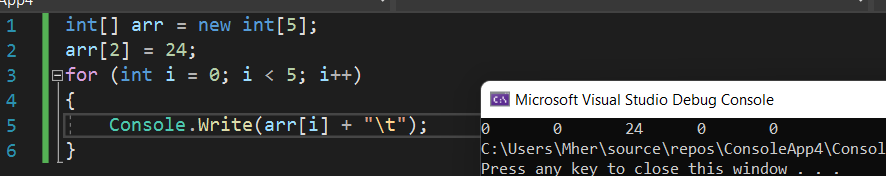
**arr[2] = 24;**

**for (int i = 0; i < 5; i++)**

**{**

**Console.WriteLine(arr[i]);**

**}**

ScreenShot 37

Ինչպես արդեն նշել էի չարժևորված դաշտերում կլինեն 0-ներ:

### foreach

foreach-ը մեր 4-րդ ցիկլն է, որը օգրագործվում է հավաքածուների (collections) արժեքները կարդալու համար: Գրելաձև՝

**foreach(տիպ անուն in հավաքածուի-անուն){**

// **անունը** կլինի մեր յուրաքանչյուր էլեմենտը

**}**

Օրինակ՝

**int[] arr = new int[5];**

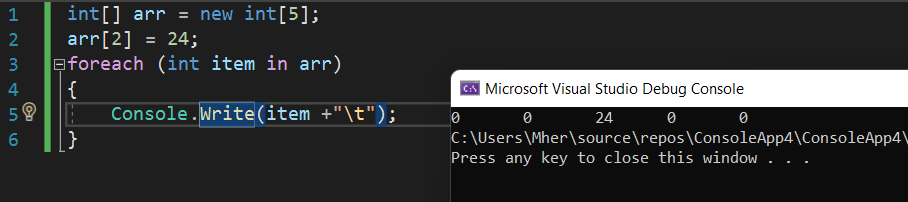
**arr[2] = 24;**

**foreach (int item in arr)**

**{**

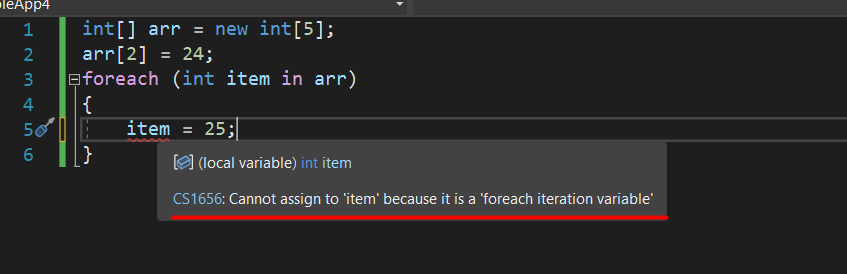
**Console.Write(item +"\t");**

**}**

ScreenShot38

Կարևոր

**foreach-**ով կարդալիս էլեմենտները readonly (միայն կարդալ) տիպի են և չենք կարող փոխարինել:

ScreenShot39

## Բազմաչափ զանգվածներ

C#-ում բազմաչափ զանգվածի չափայնության վրա սահմանափակում չկա. Միակ խնդիրը կառաջանա, եթե զանգվածը չտեղավորվի հիշողության մեջ:

Գրելաձև՝

**Տիպ[,…,] անուն = new Տիպ[s0,…,sn]**, որտեղ s0,…,sn-ը թվեր են, որոնք ցույց են տալիս զանգվածի չափսերը:

Որպես օրինակ կդիրարկենք երկչափ զանգվածը, որը այլ կերպ անվանում են մատրից:

**int[,] matrix = new int[4, 6];**

**for (int i = 0; i < matrix.GetLength(0); i++)**

**{**

**for (int j = 0; j < matrix.GetLength(1); j++)**

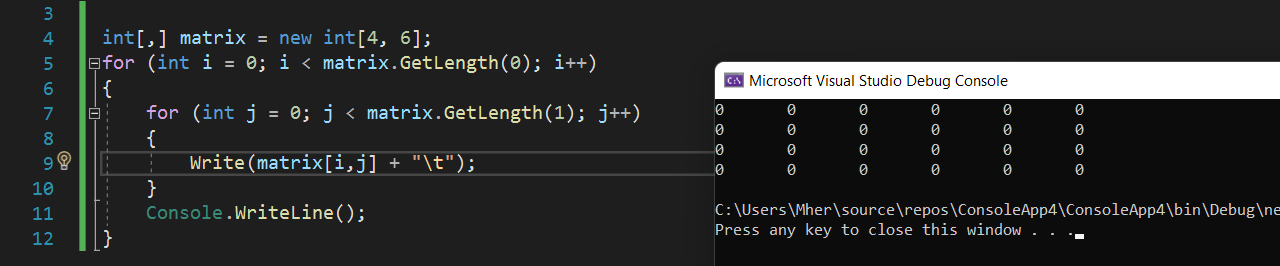
**{**

**Write(matrix[ i, j ] + "\t");**

**}**

**Console.WriteLine();**

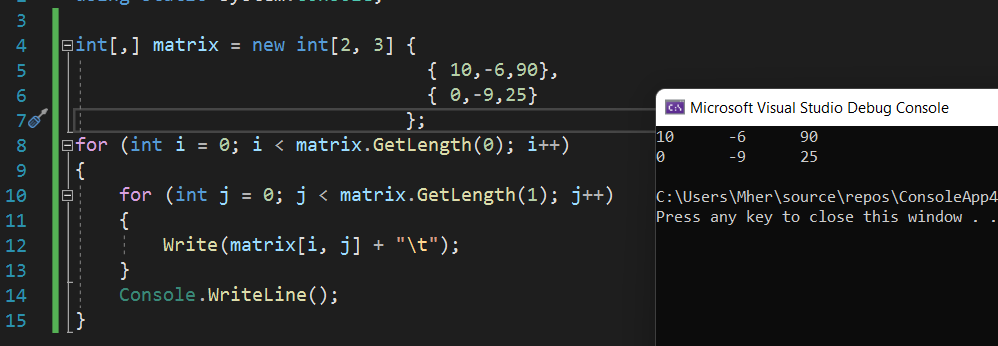
**}**

ScreenShot40

Ինչպես տեսնում ենք մատրիցի հետ աշխատելու ու էլեմենտները տպելու համար կատարում ենք ուղղանկյան հետ աշխատելու քայլերը, քանի որ երկու դեպքում ել գործ ունենք կորդինատների հետ:

**.GetLength(0)** ֆունկցիան վերադարձնում է բազմաչափ զանգվածի նշված ինդեքսում տրված թիվը: Մեր օրինակում քանի որ չափը **[4,6]** է **.GetLength(0)**-ը կվերադարձնի **4**, իսկ **.GetLength(1)-ը՝ 6:**

Կարող ենք բազմաչափ զանգվածը նույնպես ինիցիալիզացնենք մեկ տողով`

ScreenShot41

## Jagged( згичатый)

Սա իրենից ներկայացնում է զանգվածների զանգված: Այսինքն այս զանգվածի յուրաքանչյուր էլեմենտ մեկ այլ զանգվածի հղում է:

Գրելաձև՝

**Տիպ[,…,]…[,..,] անուն = new Տիպ[s0,…,sn] …[,..,]**

**Օրինակ՝**

**int[][] jag= new int[3][];**

Սա նշանակում է, որ ունենք 3 երկարությամբ զանգված, որի յուրաքանչյուր էլեմենտի մեջ պետք է պահենք զանգված՝

**int[][] jag = new int[3][];**

**jag[0] = new int[5];**

**jag[1] = new int[3];**

**jag[2] = new int[8];**

**for (int i = 0; i < jag.Length; i++)**

**{**

**for (int j = 0; j < jag[i].Length; j++)**

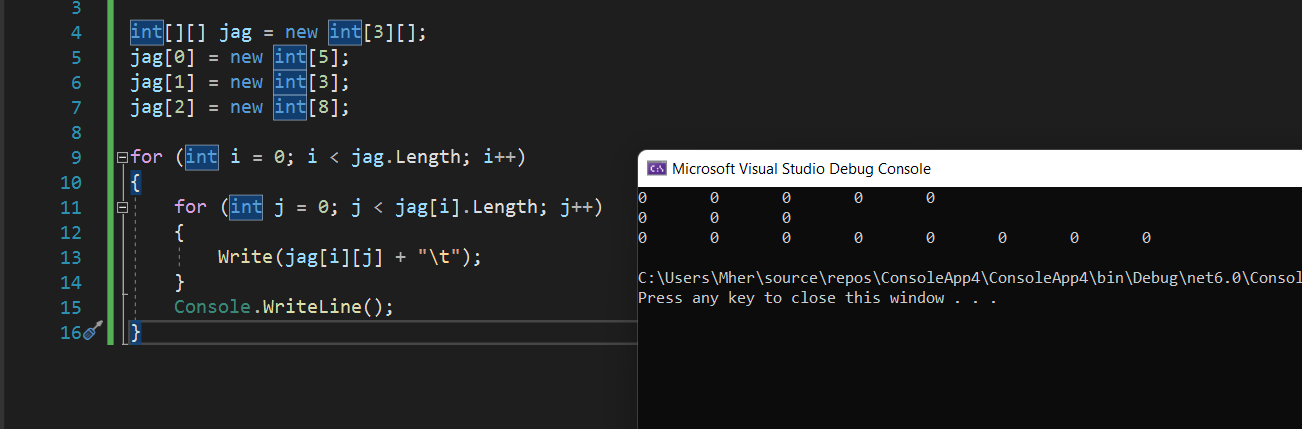
**{**

**Write(jag[i][j] + "\t");**

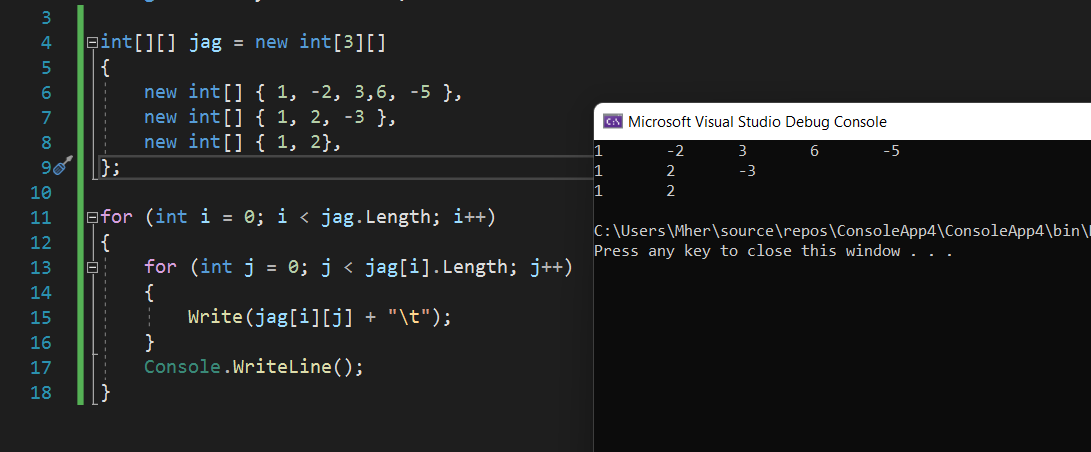
**}**

**Console.WriteLine();**

**}**

ScreenShot42

Սա նույնպես կարելի է ինիցիալիզացնել մեկ արտահայտությամբ՝

ScreenShot43

## Ֆունկցիաներ/Մեթոդներ

Ֆունկցիան կոդի հատված է, որը ունի անուն և մենք կաող ենք անունի օգնությամբ տվյալ կոդը կարղանում ենք կանչել և օգտագործել բազմակի անգամներ։

Գրելաձև՝

**Վերադարձվող\_Տիպ Անուն(Տիպ\_1 անուն\_1, ․․․ ,Տիպ\_n անուն\_n){**

**Կոդ;**

**}**

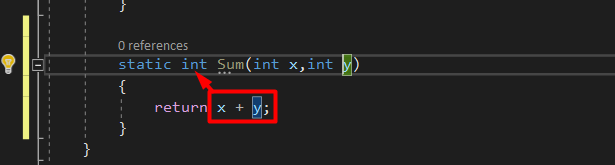
**static int Sum(int x,int y)**

**{**

**return x + y;**

**}**

Երբ ֆունկցիան վերադարձնում է արժեք, ապա օգտագործում ենք **return արժեք;** և արժեքի տիպը պետք է համընկնի **Վերադարձվող\_Տիպ** -ին։

ScreenShot44

Հիմա կարող ենք այս ֆունկցիան օգտագործել քանի անգամ կուզենք։

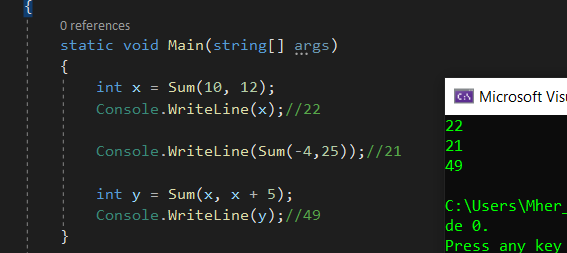
**int x = Sum(10, 12);**

**Console.WriteLine(x);//22**

**Console.WriteLine(Sum(-4,25));//21**

**int y = Sum(x, x + 5);**

**Console.WriteLine(y);//49**

ScreenShot45

Այն ֆունկցիաները, որոնք ոչ մի արժեք չեն վերադարձնում **Վերադարձվող\_Տիպ**-ի փոխարեն գրում ենք **void ։** Օրինակ՝

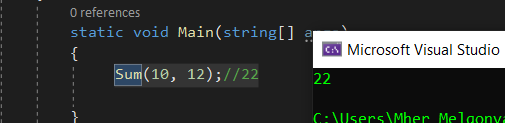
**static void Sum(int x,int y)**

**{**

**Console.WriteLine(x + y);**

**}**

Սա այն ֆունկցիաներնեն, որոնք տպում են կամ կատարումեն գործողություններ որոնք ինչ որ բան են փոխում ծրագրում, սակայն բան չեն վերադարձնում։

ScreenShot46

**void** Ֆունկցիայում կարելի է օգտագործել **return**; օպերատորը ֆունկցիայի աշխատանքը ժամանակից շուտ ընդհատելու համար։ Հիմնականում օգտագործվում է փոխանցված ոչ կոռեկտ արժեքների պատճառով։

1. Ֆունկցիաները հանդիսանում են բազմակի օգտագործման կոդ։
2. Ֆունկցիաների օգտագործումը մոդելավորում է ծրագիտանում է րը ինչի արդյունքում հեշտանում է սխալների տեղայնացումը։
3. Ֆունկցիաների օգտագործումը կարող է բերել հիշողության խնայողության։
4. Ֆունկցիաների օգտագործումը միշտ բերում է ժամանակի կորստի։

Դեռևս C լեզվից ֆունկցիանրը հանդիսացել են գլոբալ օբյեկտին։ Այսինքն ֆունկցիան չի կարող կառուցվել մեկ այլ ֆունկցիայի ներսում, բայց **C#** 3.5-ում մտցվեցին անանուն ֆունկցիա և λ (լյամդա) արտահայտությունները, որոնք ֆունկցիաներ հանդիսանալով պարտադիր պետք է կառուցվեն այլ ֆունկցիաների ներսում։ Իսկ **C#** 6-ում մտցվեց լոկալ ֆունկցիա, որն իրենից ներկայացնում է սովորական ֆունկցիա, բայց կառոցվում է այլ ֆունկցիաների ներսում։

Ֆունկցիան պետք է կանչել ծրագրի կատարման ճանապարհներից մեկի վրա։ Ֆունկցիայի կանչը կախված է ֆունկցիայի անունից,կլոր փակագծերից՝ (), և իրական փոփոխականներից, որոնց քանակը և տիպը պետք է համընկնի ֆորմալներին։ Իրական պարամետրը կամայական բարդության և երկարության արտահայտություն է ընդհուպ մինչև ֆունկցիայի կանչ, այդ թվում նույն ֆունկցայի կանչը։ Միակ սահմանափակումը՝ իրական պարամետրի տիպը պետք է համընկնի ֆորմալ պարամետրի տիպին։ Եթե ֆունկցիան **void** է, ապա այն չի կարելի օգտագործել ոչ մի արտահայտության ներսում։

***C#-****ը հանդիսանում է մաքուր գոյամետ լեզու, ինչը նշանակում է ցանկացած ֆունկցիա պետք է լինի դասի կամ ստրուկտուրայի անդամ։*

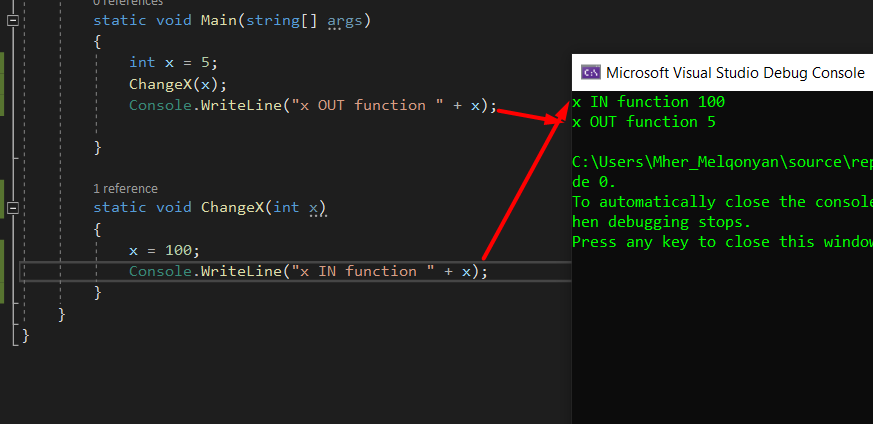
## Ֆունկցիայի կանչի իրացման ալգորիթմը

1. Ծրագիրը ղեկավարումը թռիչքաձև անցնում է կատարում կանչող ֆունկցիայի տիրույթից (ժամանակի մեծ կորուստ)։
2. Սիսիտեմը սկսում է կառուցել ֆունկցիայի սեփական հիշողությունը՝ ֆունկցիայի **Stack**։ Յուրաքանչյուր ֆունկցիայի հիշողության է հատկացվում ծրագրի ստատիկ հիշողությունից (իսկ դինամիկ հիշողությունը չի տրոհվում, այլ մնում է մեկ հատ ողջ ծրագրի համար)
   1. Տեղ է հատկացվում ֆունկցիայի վերադարձվող արժեքի համար, եթե այդպիսին կա։
   2. Տեղ է հադկացնում յուրաքանչյուր պարամետրի, որը ինիցիալիզացվում է կանչում տրված իրական արժեքով։
   3. Տեղ է հատկացվում ֆունկցիայի լոկալ փոփոխականներին։
3. Ֆունկցիայի կատարումը ավարտվում է, եթե հանդիպում է **returne**...-ի կամ փակող ձևավոր փակագծերին։
4. Ծրագրի ղեկավարումը թռիչքաձև անցում է կատարու կանչող ֆունկցիայի տիրույթ (ժամանակի խոշոր կորուստ) անմիջապես կանչի օպերատորի վրա, նախապես կանչի փոխարեն տեղադրելով ֆունկցիայի վերադարձվող արժեքը, եթե այդպիսին կա։
5. Հմակարգը ոչնչացնում է ֆունկցիային հատկացված ամբողջ հիշողությունը։

## Պարամետրերի փոխանցման ձևերը

C#-ում թույլ է տալիս ֆունկցիայի պարամետրեր փոխանցել ըստ արժեքի(by value) և ըստ հղման(by reference):Հղումով փոխանցման ժամանակ օգտագործում ենք **ref** կամ **out** մոդիֆիկատորներ։ Եթե չկա ապա կփոխանցվի ըստ արժեքի։

Ըստ արժեքի փոխանցման դեպքում ֆունկցիայի Stack-ում կառուցվում է իրական բլոկների կոպյաների և ֆունկցիան աշխատում է հենց դրա վրա, որը անհասանելի է իրականներից։ Այսպիսի ֆունկցիաները ոչ պատահաբար, ոչ էլ դիրավորյալ չեն կարող փչացնել իրական տվյալները։

ScreenShot47

Մյուս կողմից հաճախ անհրաժշտ է կառոցել տվյալներ, որոնք հենց իրականների վրա են աշխատում։Կանչի ժամանակ **ref** մոդիֆիկատորը նորից պետք է գրել։ **ref-**ով փոխանցելու շնորհիվ փոխանցում ենք ոչ թե պարամետրերը այլ դրանց հղումները։

**int x = 5;**

**ChangeX(ref x);**

**Console.WriteLine("x OUT function " + x);**

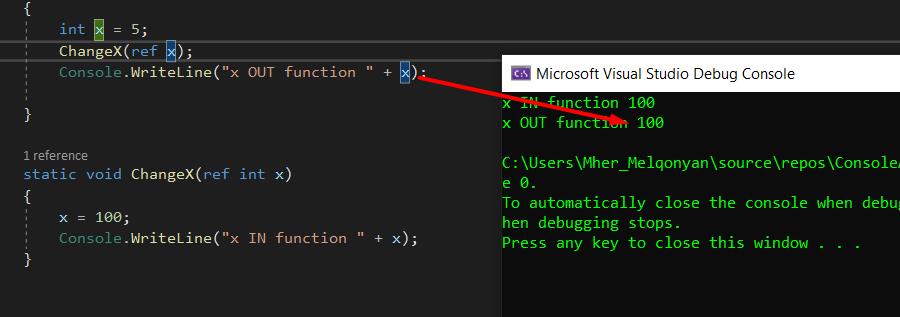
**static void ChangeX(ref int x)**

**{**

**x = 100;**

**Console.WriteLine("x IN function " + x);**

**}**

ScreenShot48

**Out-**ը նույնպես փոփոխականաը հղումով փոխանցելու համր է։

**int x = 5;**

**ChangeX(out x);**

**Console.WriteLine("x OUT function " + x);**

**static void ChangeX(out int x)**

**{**

**x = 100;**

**Console.WriteLine("x IN function " + x);**

**}**

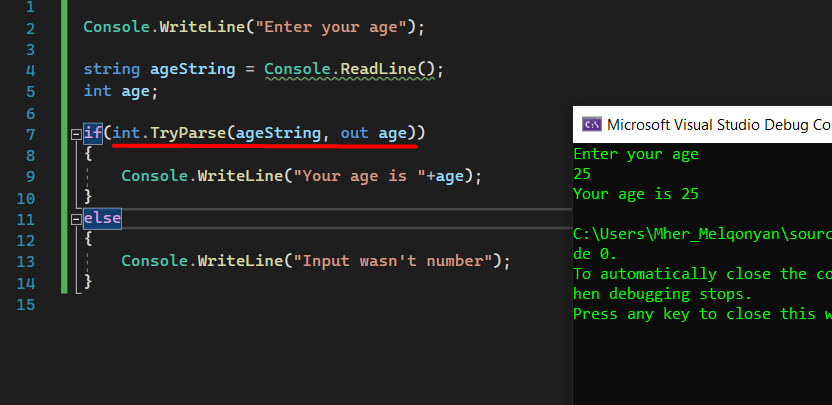
Տարբերությունը **ref** ու **out-**ի մեջ ՝

1. **Ref-**ի դեպքում փոփոխականը անպայման պետք է ինիցիալիզացված լինի։ **out-**ի դեպքում դա պարտադիր չէ։
2. **Ref-**ի արժեքը կարող ենք օգտագործել։ **out-**ին պետք է արժեք տալ օգտագործելուց առաջ ինչը նշանակում է, որ մինչ այդ ունեցած արժեքը չենք կարող օգտագործել։

Մինչև **C# 6** մեկից ավել արժեք վերադարձնելու համար օգտագործում էին **out**-ը։

Օրինակ՝

**TryParse** ֆունկցիան վերադարձնում է **bool,** իսկ **out**-ի միջողով վերադարձնում է արժեքը, եթե նրա տիպը հնարավոր է փոխել։

ScreenShot49

### Params մոդիֆիկատորը

1. **params-ը** կարելի է օգտագործել միայն 1 չափանի զանգված պարամետրի հետ։
2. Ֆունկցիան չի կարող ունենալ 1-ից ավելի **params** մոդիֆիկատրով։
3. Եթե ֆունկցիան ունի այլ պարամետրեր, ապա **params-**ը ցուցակում պետք է լինի վերջինը։

**Params-**ի դեպքում ֆունկցիան կարելի է կանչել նրան փոխանցելով պատրաստի զանգված կամ համապատասխան տիպի արժեքների բազմություն։ Այս դեպքում ֆունկցիան ինքնուրույն ստեղծում է զանգվածը և փոխանցված արժեքները լցնում դրա մեջ։

**Params-**ի օգտագործումը բերում է նրան, որ ստեղծում ենք փոփոխական քանակի պարամետրերով ֆունկցիաներ։

## Ֆունկցիաների վերաբեռնում (overload)

2 և ավելի նույնանուն ֆունկցիաներ կոչվում են վերաբեռնված, եթե

1. Տարբերվում են պարամետրերի քանակով
2. Տարբերվում են տիպերով

Վերաբեռնումը պետք է կատարել այն դեպքում, երբ մի քանի տարբեր ֆունկցիաներ կատարում են նույն գործողությունը, բայց տարբեր քանակի կամ տիպի պարամետրերով։

## Tuple types

**C# 6**-ից սկսած ստեղծվեց թուփլ տիպերը։ Սրանք մեզ հայտնի տիպերի խմբեր են։

Օրինակ՝

**(double, int) t = (4.5, 3)**

**Console.WriteLine("Tuple with elements "+ t.Item1 + " and " + t.Item2);**

Այս տիպը շատ հարմար է ֆունկցիանրի դեպքում միանգամից մի քանի արժեքներ վերադարձնելու համար։

**(string name, double value) = F();**

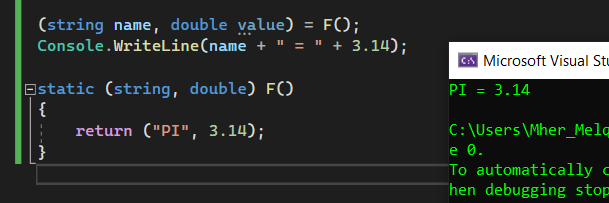
**Console.WriteLine(name + " = " + 3.14);**

**static (string, double) F()**

**{**

**return ("PI", 3.14);**

**}**

ScreenShot50

Թուփլ տիպը կարելի է սահմանել 3 եղանակով՝

**(string, double) p = ("PI", 3.14);**

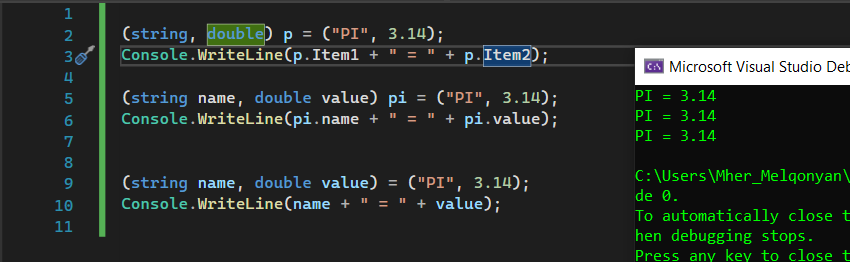
**Console.WriteLine(p.Item1 + " = " + p.Item1);**

**(string name, double value) pi = ("PI", 3.14);**

**Console.WriteLine(pi.name + " = " + pi.value);**

**(string name, double value) = ("PI", 3.14);**

**Console.WriteLine(name + " = " + value);**

ScreenShot51