### Modulul 2. Bazele programării în C#. Operatori de bază.

#### Tipuri de date

1. Tipuri numerice întregi

Reprezintă numere întregi (0,1,2,3, etc.)

Toate tipurile numerice întregi pot utiliza operații:

* Aritmetice (+, -, \*, /, %, ++, --)
* De comparație (==, >, <, >=, <=) - dacă unul din operatori nu este număr obținem rezultatul false
* De egalitate(==, !=).

|  |  |  |  |
| --- | --- | --- | --- |
| **Tipul** | **Intervalul de valori** | **Dimensiunea** | **Tipul în .NET** |
| **sbyte** | -128, 127 | Număr întreg de 8 biți (cu semn) | [System.SByte](https://docs.microsoft.com/en-us/dotnet/api/system.sbyte) |
| **byte** | 0, 255 | Număr întreg de 8 biți (fără semn) | [System.Byte](https://docs.microsoft.com/en-us/dotnet/api/system.byte) |
| **short** | -32,768, 32,767 | Număr întreg de 16 biți (cu semn) | [System.Int16](https://docs.microsoft.com/en-us/dotnet/api/system.int16) |
| **ushort** | 0, 65,535 | Număr întreg de 16 bits (fără semn) | [System.UInt16](https://docs.microsoft.com/en-us/dotnet/api/system.uint16) |
| **int** | -2,147,483,648  2,147,483,647 | Număr întreg de 32 bits (cu semn) | [System.Int32](https://docs.microsoft.com/en-us/dotnet/api/system.int32) |
| **uint** | 0, 4,294,967,295 | Număr întreg de 32 bits (fără semn) | [System.UInt32](https://docs.microsoft.com/en-us/dotnet/api/system.uint32) |
| **long** | -9,223,372,036,854,775,808 9,223,372,036,854,775,807 | Număr întreg de 64 bits (cu semn) | [System.Int64](https://docs.microsoft.com/en-us/dotnet/api/system.int64) |
| **ulong** | 0, 18,446,744,073,709,551,615 | Număr întreg de 32 bits (fără semn) | [System.UInt64](https://docs.microsoft.com/en-us/dotnet/api/system.uint64) |

1. Tipuri numerice cu virgulă mobilă

Reprezintă numerele reale(**float x = 0.1f, double n = 0.005** etc.)

Toate tipurile numerice cu virgulă mobilă pot utiliza același set de operații ca și numerele întregi.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Tipul** | **Intervalul de valori** | **Cifre după virgulă** | **Dimensiunea** | **Tipul în .NET** |
| **float** | ±1.5 x 10−45 to ±3.4 x 1038 | ~6-9 | 4 bytes | [System.Single](https://docs.microsoft.com/en-us/dotnet/api/system.single) |
| **double** | ±5.0 × 10−324 to ±1.7 × 10308 | ~15-17 | 8 bytes | [System.Double](https://docs.microsoft.com/en-us/dotnet/api/system.double) |
| **decimal** | ±1.0 x 10-28 to ±7.9228 x 1028 | 28-29 | 16 bytes | [System.Decimal](https://docs.microsoft.com/en-us/dotnet/api/system.decimal) |

1. Tipuri de date caracter

Reprezintă un singur caracter unicod (**char x = ‘x’**)

Toate tipurile numerice cu virgulă mobilă pot utiliza același set de operații ca și numerele întregi.

|  |  |  |
| --- | --- | --- |
| **Tipul** | **Dimensiunea** | **Tipul în .NET** |
| **char** | 2 bytes | [System.Char](https://docs.microsoft.com/en-us/dotnet/api/system.char) |

1. Tipul bool

Reprezintă o valoare logică care poate fi **true sau false.(bool x = true; bool y = false;)**

|  |  |  |
| --- | --- | --- |
| **Tipul** | **Dimensiunea** | **Tipul în .NET** |
| **bool** | 1 byte | [System.Boolean](https://docs.microsoft.com/en-us/dotnet/api/system.boolean) |

1. Tipul enumerare

|  |  |  |
| --- | --- | --- |
| **Tipul** | **Dimensiunea** | **Tipul în .NET** |
| **enum** | 4 bytes | [System.Enum](https://docs.microsoft.com/en-us/dotnet/api/system.enum?view=netcore-3.1) |

Tipul enumerare sau **enum** reprezintă un set de constante. Tipul enumerare poate fi utilizat pentru a determina starea unui obiect si de executa o anumita functie in dependenta de aceasta stare.

enum Anotimpuri

{

Primavara,

Vara,

Toamna,

Iarna

}

#### Constante

Constantele sunt valori care nu pot fi modificate în timpul execuției programului, dar pot fi accesate pentru citire sau operațiuni(se declară cu **const**).

Exemplu:

**class Calendar1**

**{**

**public const int Luni = 12;**

**}**

În clasa Calendar1 variabila Luni este o constantă de tip int care are valoarea 12, aceasta valoare este substituită direct în limbajul intermediar și nu poate fi modificată în timpul execuției.

#### Variabile

O variabilă este un container care păstrează date. O variabilă poate avea o mulțime de tipuri(discutate anterior), și își poate schimba valoarea în timpul execuției programului(cu unele excepții).

private string motan = “Cot”;

O variabilă are mai multe elemente:

1. Nivelul de accesibilitate:
2. Tipul variabilei
3. Numele variabilei
4. Valoarea variabilei

|  |  |
| --- | --- |
| **Nivelul de accesibilitate** | **Explicație** |
| [public](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/public) | Acces liber. |
| [protected](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/protected) | Acces în interiorul clasei sau în claselor derivate. |
| [internal](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/internal) | Acces în interiorul clasei și în clasele din același assembly. |
| [protected internal](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/protected-internal) | Acces în interiorul clasei, în cadrul claselor derivate și în cadrul claselor din același assembly. |
| [private](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/private) | Acces doar în interiorul clasei. |
| [private protected](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/private-protected) | Acces în interiorul clasei, și în clasele derivate DOAR dacă fac parte din același assembly.  Disponibil de la lansarea C# 7.2. |

#### Tipuri anonime

Tipurile anonime reprezintă o metodă comodă de a include un set de parametri diferiți într-un singur obiect. (Parametrii pot fi citiți dar nu pot fi modificați în runtime).

var RAM = new { Producator = “Corsair”, Memorie = 4096 };

Console.WriteLine(RAM .Producator+ ”:” + RAM.Memorie);

#### Introducerea și citirea datelor din consolă

1. Introducerea datelor:

* Console.WriteLine()
* Console.Write()

1. Citirea datelor:

* Console.[ReadKey()](https://msdn.microsoft.com/en-us/library/471w8d85(v=vs.110).aspx)
* Console.[ReadLine()](https://msdn.microsoft.com/en-us/library/system.console.readline(v=vs.110).aspx)
* Console.[Read()](https://msdn.microsoft.com/en-us/library/system.console.read(v=vs.110).aspx)

#### Tipuri struct și referință

Tipurile **struct** în C# reprezintă o metodă simplă de a stoca mai multe variabile de tipuri diferite în interiorul unui structuri.

public struct Student

{

int nr;

string nume;

float notaMedie;

string biografie;

}

Student student = new Student ();

student .nr = 1;

student .nume = “Ion”;

student .notaMedie = 9.5f;

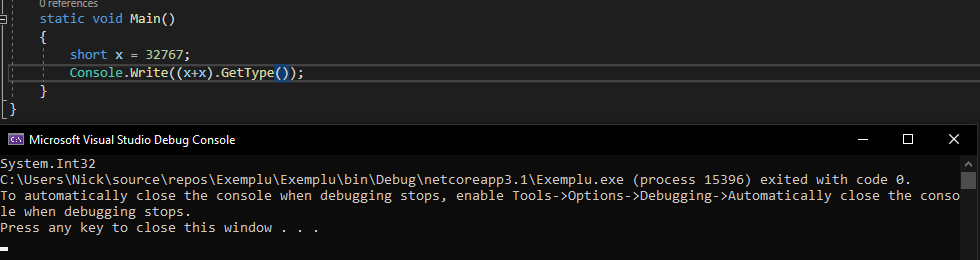
student .biografie = “Ion a crescut in Bulboaca, departe de civilizatie”;

#### Conversia tipurilor

În C# conversia se folosește pentru a modifica tipul unei valori în altul. Conversia poate fi efectuată prin 2 metode:

1. Conversie implicită - este cea efectuată de C# în mod automat și este considerată sigură(nu poate genera erori).

**De exemplu:** când suma a două variabile numerice de tip short va fi mai mare decât valoarea maximă short de 32 767 suma acestora va fi automat convertită în tip int.



1. Conversie explicită- este cea efectuată de către utilizator utilizând funcții predefinite. Acest tip de conversie este considerat nesigur pentru că poate genera erori în cazul în care conversia este imposibilă din cauza incompatibilității tipurilor.

**Exemplu:**

float x = 20.0005f;

int y = Convert.ToInt32(x);

Rezultat: 20

#### Operatori

1. Operatori aritmetici. (A=10; B=20)

|  |  |  |
| --- | --- | --- |
| **Operator** | **Descriere** | **Exemplu** |
| + | Adună două valori | A + B = 30 |
| - | Scade a doua valoare din prima | A - B = -10 |
| \* | Înmulțește valorile | A \* B = 200 |
| / | Împarte primul operator la al doilea operator | A / B = 0 |
| % | Indică restul împărțirii | A % B = 10 |
| ++ | Crește valoarea operatorului cu 1 | A++ = 11 |
| -- | Scade valoarea operatorului cu 1 | A-- = 9 |

1. Operatori relaționali. (A=10; B=20)

|  |  |  |
| --- | --- | --- |
| **Operator** | **Descriere** | **Exemplu** |
| == | Returnează adevăr dacă operatorii sunt egali | (A == B) **false** |
| != | Returnează adevăr dacă operatorii NU sunt egali | (A != B) **true** |
| > | Verifică dacă operatorul din stânga este mai mare decât cel din dreapta | (A > B) **false** |
| < | Verifică dacă operatorul din stânga este MAI MIC decât cel din dreapta | (A < B) **true** |
| >= | Verifică dacă operatorul din stânga este mai mare sau egal decât cel din dreapta | (A >= B) **false** |
| <= | Verifică dacă operatorul din stânga este MAI MIC sau egal decât cel din dreapta | (A <= B) **true** |

1. Operatori logici (A = true; B = false;)

|  |  |  |
| --- | --- | --- |
| **Operator** | **Descriere** | **Exemplu** |
| && | Operația logică ȘI. Returnează adevăr doar dacă ambii operatori sunt true. | **(A && B) false** |
| || | Operația logică ORI. Returnează adevăr dacă oricare dintre operatori este true. | **(A || B) true.** |
| ! | Operația logică OPUS. Returnează valoarea inversă valorii inițiale. | **!(A && B) true.** |

#### Condiții

1. Condiția if, if else și else

Condițiile if, if else și else vă sunt bine cunoscute deja. Scopul lor este să determine valoarea logică a unei afirmații(true, false) și să execute un fragment de cod respectiv în dependență de aceasta.

**Exemplu:**

int x = 5;

int y = 6;

if(x>y)

{

x++;

}

else if(x==y)

{

y++;

}

else

{

x+=y;

}

În exemplul de mai sus avem declarate două variabile de tip int x și y. X este 5, iar Y 6.

1. În urma comparației făcute în prima condiție i**f(x>y)** vom obține **if(5>6)** ceea ce putem spune logic că e fals, deci programul nu va executa comanda x++
2. Programul continua catre următoarea condiție care este **else if(x==y**), în urma substituției variabilelor obținem **else if(5==6**), ceea ce iarăși este fals. Comanda y++ nu va fi executată.
3. În urma excluderii primei condiții știm că **x nu este mai mare ca y**, în urma excluderii condiției 2 știm că **x nu este egal cu y,** deci ultima condiție else va presupune că x<y. În urma substituției variabilelor cu valoare lor obținem 5<6 ceea ce este adevărat. **Programul va executa comanda x+=y**; în urma la care valoarea lui x = 5 va fi adunată cu valoarea lui y = 6, și x va obține valoarea sumei, **deci x = 11**.
4. Condiția switch

Condiția switch este o alternativă viabilă pentru condiția if else, **care verifică dacă o anumită variabilă coincide cu case-urile indicate de programator**, dacă variabila coincide este executat codul din case-ul respectiv, dacă variabila nu coincide cu nici un case se executa codul indicat în **default:**

**Exemplu:**

Color color = new Color();

switch (c)

{

case Color.Red:

Console.WriteLine("The color is red");

break;

case Color.Green:

Console.WriteLine("The color is green");

break;

case Color.Blue:

Console.WriteLine("The color is blue");

break;

default:

Console.WriteLine("The color is unknown.");

break;

}

}

1. Operatorul ?

Operatorul ?(operator ternar) reprezintă încă o alternativă pentru condiția if else, dar acesta este mai compact.

**Exemplu:**

int x = 20, y = 10;

int rezultat = x > y ? x : y;

În exemplul de mai sus operatorul ? verifică dacă **x este mai mare ca y,** în cazul în care această expresie **este adevărată int-ul rezultat va fi egal cu x,** în caz că nu rezultat va fi egal cu y.

În cazul nostru x este 20, iar y - 10, deci x este mai mare, respectiv rezultat = 20;

#### Cicluri

1. Ciclul for

Se folosește în cazul în care știm exact de câte iterații avem nevoie.

for (int i = 0; i < 3; i++) {Console.WriteLine(i);}

Rezultat:

0

1

2

1. Ciclul while

Ciclul while va executa codul din interior atâta timp cât verificarea logică din interior este true. Se folosește când nu știm de câte iterații avem nevoie dar are potențial de a cauza cicluri infinite dacă logica nu este corectă.

int i = 0;

while (i < 3)

{

Console.WriteLine(i);

i++;

}

1. Ciclul foreach

Ciclul foreach se folosește doar în cazul în care vrem să procesăm toate elementele de un anumit tip dintr-un tablou.

De exemplu:

string[] masini= {"Mazda", "Nisan", "Toyota", "Mitsubishi"};

foreach (string str in cars)

{

Console.WriteLine(str);

}

În exemplul de mai sus ciclul foreach va procesa tabloul mașini și va printa fiecare element de tip string din el, avantajul de bază este că nu trebuie să indicăm manual lungimea ciclului precum în ciclul for.

1. Instrucțiunea break și continue

Break se folosește în cazul în care vrem să oprim un ciclu, de exemplu atunci când căutăm o valoare, iar după ce o găsim nu dorim să mai continuăm execuția.

string[] masini= {"Mazda", "Nisan", "Toyota", "Mitsubishi"};

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

{

if(masini[i]==”Mazda”)

{

Console.Write(“Mazda are numarul de ordine:”+i);

break;

}

}

Break pe de altă parte se va utiliza când vrem să trecem la următoarea iterație a ciclului fără a executa codul de după continue.

string[] masini= {"Mazda", "Nisan", "Toyota", "Mitsubishi"};

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

{

if(masini[i]==”Mazda”)

{

continue;

}

Console.Write(masini[i]+“ are numarul de ordine:”+i);

}

1. Intrucțiunea goto

Se folosește pentru a trece într-o anumită parte a unui program care este predeterminată din timp, în felul acesta îndată ce condiția care este căutată se îndeplinește restul codului care este inclus după va fi ignorat.

static void Main()

{

string[] masini = { "Mazda", "Nisan", "Toyota", "Mitsubishi" };

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

{

if (masini[i] == "Mazda")

{

goto Found;

}

}

goto Finish;

Found:

Console.WriteLine("Mazda a fost gasita");

Finish:

Console.WriteLine("Mazda nu a fost gasita");

// Keep the console open in debug mode.

Console.WriteLine("Press any key to exit.");

Console.ReadKey();

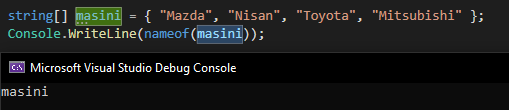
}

În exemplul de mai sus, ciclul for indicat va căuta în tabloul masini stringul Mazda, îndată ce acesta este găsit execuția programului “sare” la regiunea Found și ignoră restul ciclului și codul după el.

Acest tip de cod se recomandă când știm deja scenariile de execuție al codului.

#### Utilizarea nameof

Nameof se utilizează pentru a obține numele unei variabile, clase, funcții etc.



#### Tablouri

1. Tablouri unidimensionale

int[] myNumbers = {5, 1, 8, 9};

int[] myNumbers = new int[4];

int[] myNumbers = null;

1. Tablouri multidimensionale

int[,] biDim = new int[2, 2];

int[,] biDim = {2,2}{2,2};

int[,,] triDim = new int[3, 3, 3];

Console.WriteLine(triDim.Length);

//traversal

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

{

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

{

for (int r = 0; r < triDim.GetLength(2); r++)

{

Console.Write(triDim[i, j, r] + " ");

}

Console.WriteLine();

}

Console.WriteLine();

}

//Initializarea manuala

int[,] biDim = new int[2, 2]{{2,2}, {2,2}};

int[,,] triDim = new int[3, 3, 3]

{

{ { 1, 2, 3 },

{ 4, 5, 6 },

{ 4, 5, 6 }

},

{

{ 7, 8, 9 },

{ 10, 11, 12 },

{ 10, 11, 12 }

},

{

{ 7, 8, 9 },

{ 10, 11, 12 },

{ 10, 11, 12 }

},

};

1. Tablouri neordinare(jagged arrays)

Uneori putem avea nevoie de a declara unele tablouri cu dimensiuni neordinare, ca de exemplu pentru a reprezenta statutul locurilor ocupate intr-un cinema pentru aceste ocazii putem utiliza

string[][] numere = new string[4][]

{

new string[]{"1", "2", "3"},

new string[]{"4", "5", "6", "7", "8"},

new string[]{"9", "1", "2", "3", "4", "5"},

new string[]{"6", "7", "8", "9", "1", "2", "3"},

};

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

{

Console.Write("Name List ({0}): ", i + 1);

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

{

Console.Write(numere[i][j] + "\t");

}

Console.WriteLine();

}

Console.ReadKey();

#### Șiruri de caractere(string)

1. Crearea șirurilor de caractere

//Manual

string s1 = "A string is more ";

//Dintr-un sir de caractere

char[] letters = { 'A', 'B', 'C' };

string str6 = new string(letters);

//Prin repetarea unui caracter de X ori

string tabs = new String('X', 5);

1. Operații cu string-uri

//Formatare

var pw = (firstName: "Phillis", lastName: "Wheatley", born: 1753, published: 1773);

Console.WriteLine("{0} {1} was an African American poet born in {2}.", pw.firstName, pw.lastName, pw.born);

Console.WriteLine("She was first published in {0} at the age of {1}.", pw.published, pw.published - pw.born);

Console.WriteLine("She'd be over {0} years old today.", Math.Round((2018d - pw.born) / 100d) \* 100d);

|  |  |
| --- | --- |
| **Functie** | **Explicatie** |
| **Clone()** | Produce o copie exacta a unui string |
| **CompareTo()** | Compara doua stringuri, returnează true dacă sunt identice și false dacă nu. **(Returnează eroare dacă unul din stringuri e null)** |
| **Contains()** | Verifica dacă un string contine un anumit caracter sau secvența de caractere |
| **EndsWith()** | Verifica daca stringul se termina cu un anumit caracter |
| **Equals()** | Compara doua stringuri, returnează true dacă sunt identice și false dacă nu. |
| **GetHashCode()** | Returneaza HashValue a unui string |
| **IndexOf()** | Identifica poziția unui caracter in string(prima litera găsită dacă exista mai multe) |
| **ToLower()** | Transforma toate caracterele stringului în caractere minuscule |
| **ToUpper()** | Transforma toate caracterele stringului în caractere majuscule |
| **Insert()** | Insereaza in string un alt string sau un caracter in poziția indicată |
| **LastIndexOf()** | Găsește locul ultimii apariții a unui caracter |
| **Length** | Returnează lungimea stringului |
| **Remove()** | Șterge toate caracterele incepand cu valoarea inițială indicată și finisand cu cea finală indicată |
| **Replace()** | Înlocuiește toate caracterele indicate de utilizator cu alt caracter selectat |
| **Split()** | Împarte un string în mai multe stringuri în dependență de valoare separator |
| **StartsWith()** | Verifică dacă stringul începe cu un anumit caracter |
| **Substring()** | Returnează o parte componentă din string |
| **ToCharArray()** | Transformă un string într-un tablou de caractere |
| **Trim()** | Șterge locurile libere de la începutul și sfârșitul string-ului |