

Laborator 2

Cuvinte cheie în C#

Cuvintele cheie sunt identificatori predefiniți cu semnificație specială pentru compilator.

Definim în C# următoarele cuvinte cheie:

abstract	as	base	bool	break
byte	case	catch	char	checked
class	const	continue	decimal	default
delegate	do	double	else	enum
event	explicit	extern	false	finally
fixed	float	for	foreach	goto
if	implicit	in	int	interface
internal	is	lock	long	namespace
new	null	object	operator	out
override	params	private	protected	public
readonly	ref	return	sbyte	sealed
short	sizeof	stackalloc	static	string
struct	switch	this	throw	true
try	typeof	uint	ulong	unchecked
unsafe	ushort	using	virtual	void
volatile	while			

Pentru a da semnificații specifice codului, în C# avem și **cuvinte cheie contextuale**:

ascending	by	descending	equals	from
get	group	into	join	let
on	orderby	partial	select	set
value	where	yield		

În general, cuvintele cheie nu pot fi folosite în programele pe care le scriem, dându-le o altă semnificație. În cazul în care, totuși, dorim să le dăm o altă semnificație, va trebui să le scriem cu simbolul „@” ca prefix. Datorită neclarităților care pot să apară, se va evita folosirea cuvintelor rezervate în alte scopuri.

Constante - În C# există două modalități de declarare a constantelor: folosind **const** sau folosind modificatorul **readonly**. Constantele declarate cu **const** trebuie să fie inițializate la declararea lor.

Exemplul 1:

```
const int x;           //gresit, constanta nu a fost initializata
const int x = 13;      //corect
```

Constantele declarate cu ajutorul lui **readonly** sunt doar variabilele membre ale claselor, ele putând fi inițializate doar de către constructorii claselor respective.

Exemplul 2:

```
readonly int x;         //corect
readonly int x = 13;    //corect
```

Variable - O variabilă în C# poate să conțină fie o valoare a unui tip elementar, fie o referință la un obiect. C# este „case sensitive”, deci face distincție între litere mari și mici.

Exemplul 3:

```
int Salut;  
int Azi_si_maine;  
char caracter;
```

Expresii și operatori

Definiție: Prin **expresie** se înțelege o secvență formată din **operatori** și **operandi**. Un **operator** este un simbol ce indică acțiunea care se efectuează, iar **operandul** este valoarea asupra căreia se execută operația.

Operatorii se împart în trei categorii:

- Unari: - acționează asupra unui singur operand
- Binari: - acționează între doi operanzi
- Ternari: - acționează asupra a trei operanzi; există un singur operator ternar și acesta este **?:**

În C# sunt definiți mai mulți operatori. În cazul în care într-o expresie nu intervin paranteze, operațiile se execută conform priorității operatorilor. În cazul în care sunt mai mulți operatori cu aceeași prioritate, evaluarea expresiei se realizează de la stânga la dreapta. În tabelul alăturat prioritatea descrește de la 0 la 13.

Tabelul de priorități:

Prioritate	Tip	Operatori	Asociativitate
0	Primar	() [] f() . x++ x-- new typeof sizeof checked unchecked ->	→
1	Unar	+ - ! ~ ++x --x (tip) true false & sizeof	→
2	Multiplicativ	* / %	→
3	Aditiv	+ -	→
4	De deplasare	<< >>	→
5	Relațional	< > <= >= is as	→
6	De egalitate	== !=	→
7	AND (SI) logic	&	→
8	XOR (SAU exclusiv) logic	^	→
9	OR (SAU) logic		→
10	AND (SI) condițional	&&	→
11	OR (SAU) condițional		→
12	Condițional(ternar)	?:	←
13	atribuire simplă atribuire compusă	= *= /= %= += -= ^= &= <<= >>= =	←

Exemplul 4: folosind operatorul ternar **?:**, să se decidă dacă un număr citit de la tastatură este pozitiv sau negativ.

Indicații:

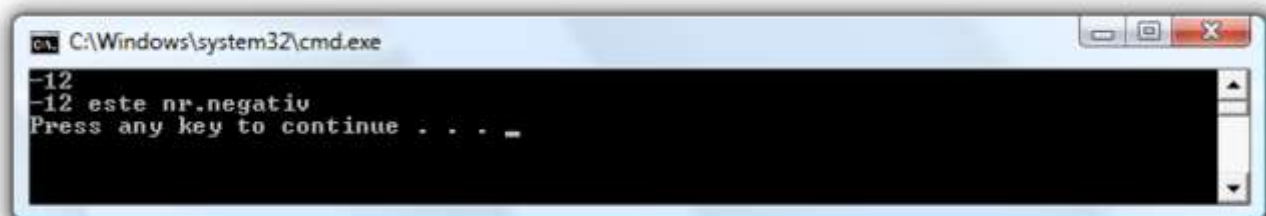
- Sintaxa acestui operator este: **(condiție) ? (expr_1): (expr_2)** cu semnificația se evaluează **condiție**, dacă ea este adevărată se execută **expr_1**, altfel **expr_2**
- **int.Parse** convertește un șir la **int**

```

using System;
using System.Collections.Generic;
using System.Text;
namespace OperatorConditional
{
    class Program
    {
        static void Main(string[] args)
        {
            int a;
            string rezultat;
            a = int.Parse(Console.ReadLine());
            Console.Write(a);
            rezultat = (a > 0) ? " este nr. pozitiv" : " este nr. negativ";
            Console.Write(rezultat);
            Console.ReadLine();
        }
    }
}

```

În urma rulării programului obținem:



```

C:\Windows\system32\cmd.exe
-12
-12 este nr.negativ
Press any key to continue . . . .

```

Exemplul 5: Folosind operatorul %, să se verifice dacă un număr este par sau impar. Observație:

Convert.ToInt32 convertește un șir la **Int32**

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace primul_proiect
{
    class Program
    {
        static void Main(string[] args)
        {
            int x;
            x = Convert.ToInt32(Console.ReadLine());
            if (x % 2 == 0) Console.WriteLine("este par");
            else System.Console.WriteLine("este impar");
        }
    }
}

```



```

C:\Windows\system32\cmd.exe
4
4 este numar par
Press any key to continue . . . .

```

Exemplul 6: Următorul program afișează la consolă tabelul de adevăr pentru operatorul logic &.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

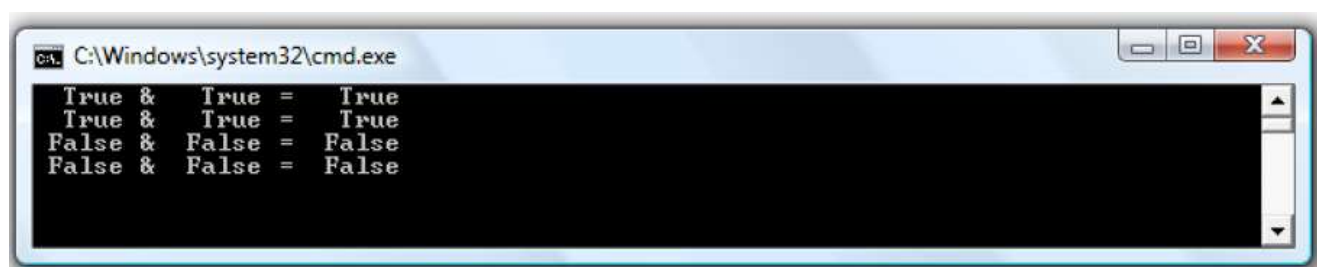
namespace Exemplul_6
{
    class Program
    {
        static void Main(string[] args)
        {
            bool v1, v2;
            v1 = true; v2 = true;
            Console.WriteLine("{0,6}" + " & " + "{0,6}" + " = " + "{0,6}",
                                v1, v2, v1 & v2);

            v1 = true; v2 = false;
            Console.WriteLine("{0,6}" + " & " + "{0,6}" + " = " + "{0,6}",
                                v1, v2, v1 & v2);

            v1 = false; v2 = true;
            Console.WriteLine("{0,6}" + " & " + "{0,6}" + " = " + "{0,6}",
                                v1, v2, v1 & v2);

            v1 = false; v2 = false;
            Console.WriteLine("{0,6}" + " & " + "{0,6}" + " = " + "{0,6}",
                                v1, v2, v1 & v2);

            Console.ReadKey();
        }
    }
}
```



Opțiuni de afișare

Pentru a avea control asupra modului de afișare a informației numerice, se poate folosi următoarea formă a lui `WriteLine()`:

```
WriteLine("sir", var1, var2, ..., varn);
```

unde „sir” este format din două elemente:

- caracterele afișabile obișnuite conținute în mesaje
- specificatorii de format ce au forma generală `{nr_var,width:fmt}` unde **nr_var** precizează numărul variabilei (parametrului) care trebuie afișată începând cu 0, **width** stabilește lățimea câmpului de afișare, iar **fmt** stabilește formatul

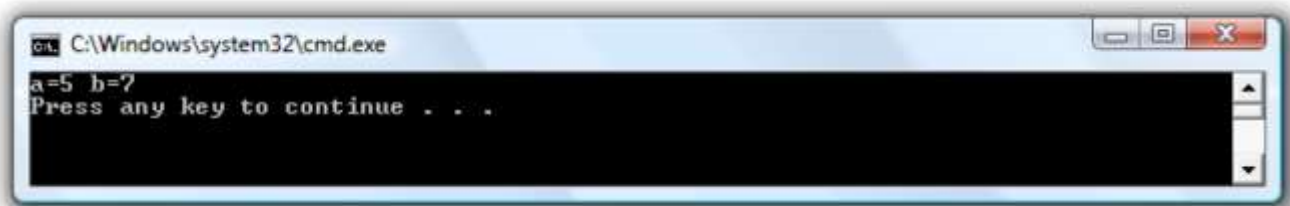
Exemplul 7:


```

using System;
using System.Collections.Generic;
using System.Text;

namespace Exemplul_7
{
    class Program
    {
        static void Main(string[] args)
        {
            int a, b, c = 5;
            a = c++;
            b = ++c;
            Console.WriteLine("a={0} b={1}", a,b);
        }
    }
}

```



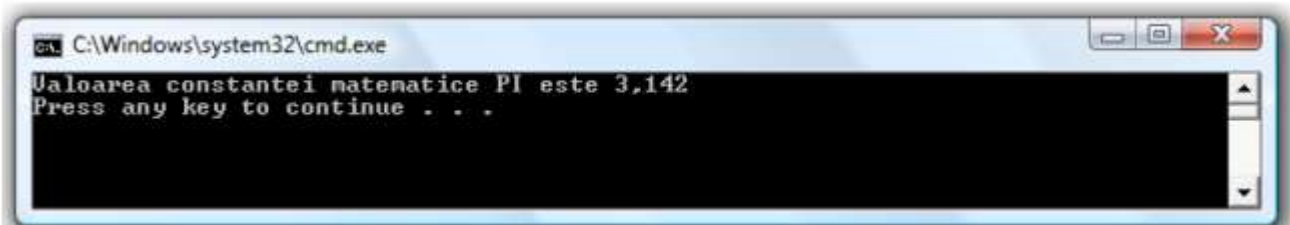
Exemplul 8: în acest exemplu, formatul de afișare ales `#.###` va produce afișarea cu trei zecimale a constantei PI

```

using System;
using System.Collections.Generic;
using System.Text;

namespace Exemplul_8
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Valoarea constantei matematice PI este
                               {0:#.###}", Math.PI);
        }
    }
}

```



Conversii

În C# există două tipuri de conversii numerice:

- implicite
- explicite.

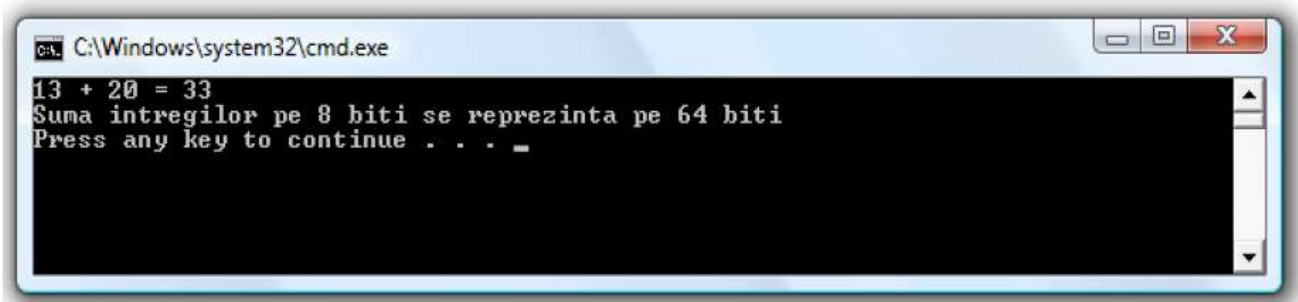
Conversia implicită se efectuează (automat) doar dacă nu este afectată valoarea convertită.

Exemplul 9: Exemplul următor realizează suma a două valori numerice fără semn cu reprezentare

pe 8 biți. Rezultatul va fi reținut pe 64 biți

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Exemplul_9
{
    class Program
    {
        static void Main(string[] args)
        {
            byte a = 13; // byte intreg fara semn pe 8 biți
            byte b = 20;
            long c;       // intreg cu semn pe 64 biți
            c = a + b;
            Console.WriteLine("{0} + {1} = {2}", a, b, c);
            Console.WriteLine("Suma intregilor pe 8 biți se reprezinta pe 64
biți");
        }
    }
}
```



Conversiile implicite

Regula după care se efectuează **conversiile implicite** este descrisă de tabelul următor:

din	în
sbyte	short, int, long, float, double, decimal
byte	short, ushort, int, uint, long, ulong, float, double, decimal
short	int, long, float, double, decimal
ushort	int, uint, long, ulong, float, double, decimal
int	long, float, double, decimal
uint	long, ulong, float, double, decimal
long	float, double, decimal
char	ushort, int, uint, long, ulong, float, double, decimal
float	double
ulong	float, double, decimal

Conversia explicită

Se realizează prin intermediul unei expresii cast (care va fi studiată mai târziu), atunci când nu există posibilitatea unei conversii implicite.

din	în
sbyte	byte, ushort, uint, ulong, char
byte	sbyte, char
short	sbyte, byte, ushort, uint, ulong, char
ushort	sbyte, byte, short, char
int	sbyte, byte, short, ushort, uint, ulong, char
uint	sbyte, byte, short, ushort, int, char
long	sbyte, byte, short, ushort, int, uint, ulong, char
ulong	sbyte, byte, short, ushort, int, uint, long, char
char	sbyte, byte, short
float	sbyte, byte, short, ushort, int, uint, long, ulong, char, decimal
double	sbyte, byte, short, ushort, int, uint, long, ulong, char, float, decimal
decimal	sbyte, byte, short, ushort, int, uint, long, ulong, char, float, double

Exemplul 10:

```
using System;
using System.Collections.Generic;
using System.Text;
namespace Exemplul_10
{
    class Program
    {
        static void Main(string[] args)
        {
            int a = 5;
            int b = 2;
            float c;
            c = (float)a / b; //operatorul cast
            Console.WriteLine("{0} / {1} = {2}", a, b, c);
            Console.WriteLine("Catul intregilor, reprezentat ca real datorita
operatorului cast\nde conversie explicita");
        }
    }
}
```

în urma rulării programului, se va obține:

```
C:\Windows\system32\cmd.exe
5 / 2 = 2,5
Catul intregilor, reprezentat ca real datorita operatorului cast
de conversie explicita
Press any key to continue . . . _
```

În cazul în care nu s-ar fi folosit operatorul **cast**, rezultatul - evident eronat - ar fi fost:

```
C:\Windows\system32\cmd.exe
5 / 2 = 2
Press any key to continue . . . _
```

- Des întâlnită este conversia din tipul numeric în șir de caractere și reciproc. Conversia din tipul numeric în șir de caractere se realizează cu metoda **ToString** a clasei **Object**

Exemplul 11:

```
int i = 13
string j = i.ToString();
```

Conversia din șir de caractere în număr se realizează cu ajutorul metodei **Parse** tot din clasa **Object**.

Exemplul 12:

```
string s = "13";
int n = int.Parse(s);
```

Exemplul 13: Exemplul de mai jos prezintă mai multe tipuri de conversii

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Exemplul_13
{
    class Program
    {
        static void Main(string[] args)
        {
            short srez, sv = 13;
            int iv = 123;
            long lrez;
            float frez, fv = 13.47F;
            double drez, dv = 87.86;
            string strrez, strv = "15";
            bool bv = false;

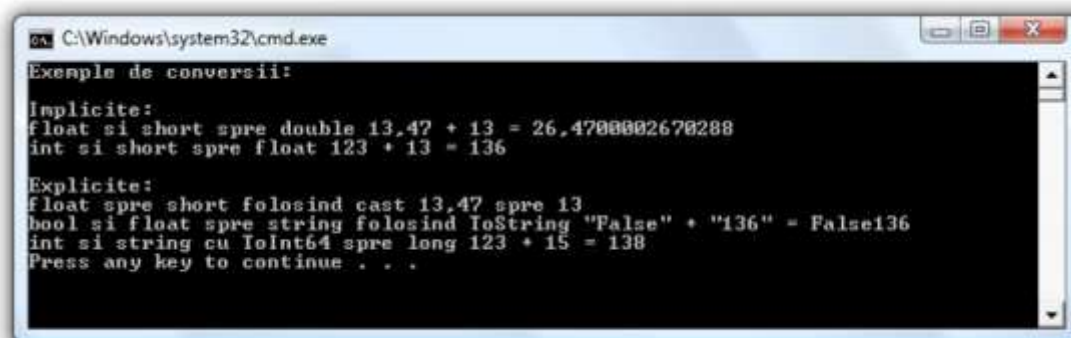
            Console.WriteLine("Exemple de conversii:\n");

            Console.WriteLine("Implicite:");
            drez = fv + sv;
            Console.WriteLine("float si short spre double {0} + {1} = {2}",
fv, sv, drez);
            frez = iv + sv;
            Console.WriteLine("int si short spre float {0} + {1} = {2}\n",
iv, sv, frez);

            Console.WriteLine("Explicite:");
            srez = (short)fv;
            Console.WriteLine("float spre short folosind cast {0} spre {1}",
fv, srez);

            strrez = Convert.ToString(bv) + Convert.ToString(frez);
            Console.WriteLine("bool si float spre string folosind ToString
\"{0}\" + \"{1}\" = {2}", bv, frez, strrez);

            lrez = iv + Convert.ToInt64(strv);
            Console.WriteLine("int si string cu ToInt64 spre long {0} + {1} =
{2}", iv, strv, lrez);
        }
    }
}
```



```
C:\Windows\system32\cmd.exe
Exemple de conversii:

Implicite:
float si short spre double 13,47 + 13 = 26,4700002670288
int si short spre float 123 + 13 = 136

Explicite:
float spre short folosind cast 13,47 spre 13
bool si float spre string folosind ToString "False" + "136" = False136
int si string cu ToInt64 spre long 123 + 15 = 138
Press any key to continue . . .
```


Conversii boxing și unboxing

Datorită faptului că în C# toate tipurile sunt derivate din clasa **Object** (**System.Object**), prin conversiile **boxing** (împachetare) și **unboxing** (despachetare) este permisă tratarea tipurilor valoare drept obiecte și reciproc. Prin conversia boxing a unui tip valoare, care se păstrează pe stivă, se produce ambalarea în interiorul unei instanțe de tip referință, care se păstrează în memoria heap, la clasa **Object**. Unboxing permite convertirea unui obiect în tipul valoare echivalent.

Exemplul 14:

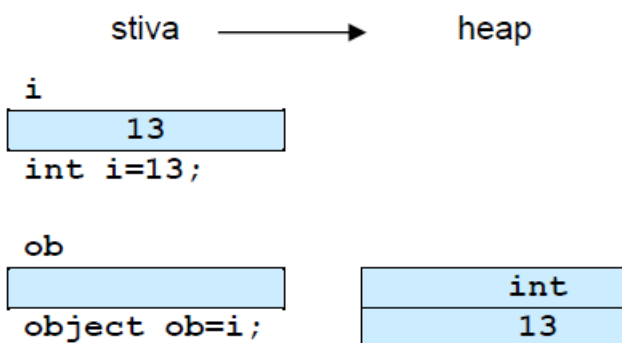
Prin boxing, variabila **i** este asignata unui obiect **ob**:

```
int i = 13;
object ob = (object)i; //boxing explicit
```

Sau

```
int i = 13;
object ob = i; //boxing implicit
```

În prima linie din exemplu se declară și se inițializează o variabilă de tip valoare, care va conține valoarea 13, valoare care va fi stocată pe stivă. Linia a doua creează o referință către un obiect alocat în heap, care va conține atât valoarea 13, cât și informația referitoare la tipul de dată conținut.



- Se poate determina tipul pentru care s-a făcut împachetarea folosind operatorul **is**:

Exemplul 15:

```
int i = 13;
object ob = i;
if (ob is int)
{
    Console.WriteLine("Impachetarea s-a facut pentru int");
}
```

Prin boxing se creează o copie a valorii care va fi conținută.

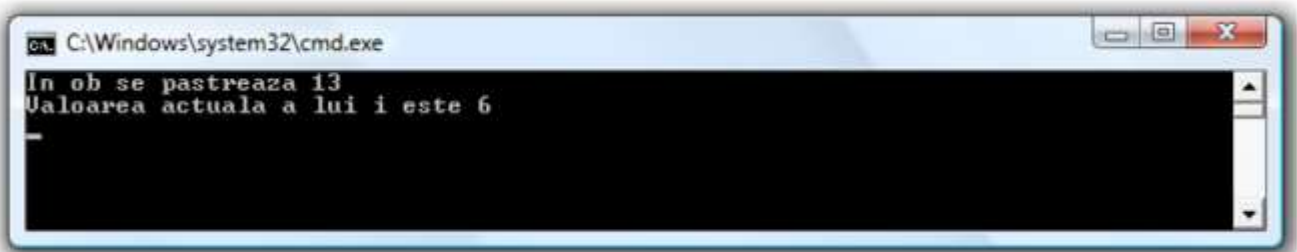
Exemplul 16:

```

using System;
using System.Collections.Generic;
using System.Text;
namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            int i = 13;
            object ob = i;
            i=6;
            Console.WriteLine("In ob se pastreaza {0}", ob);
            Console.WriteLine("Valoarea actuala a lui i este {0}", i);
            Console.ReadLine();
        }
    }
}

```

În urma rulării se obține:



```

C:\Windows\system32\cmd.exe
In ob se pastreaza 13
Valoarea actuala a lui i este 6

```

Exemplul 17: Prin conversia de tip unboxing, obiectul **ob** poate fi asignat variabilei întregi **i**:

```

int i = 13;
object ob = i; //boxing implicit
i = (int)ob;    //unboxing explicit

```

Conversii între numere și șiruri de caractere

Limbajul C# oferă posibilitatea efectuării de conversii între numere și șiruri de caractere.

Sintaxa pentru conversia număr în șir de caractere:

```
număr → șir    "\"" + număr
```

Pentru conversia inversă, adică din șir de caractere în număr, sintaxa este:

```

șir → int      int.Parse(șir)   sau Int32.Parse(șir)
șir → long     long.Parse(șir)  sau Int64.Parse(șir)
șir → double   double.Parse(șir) sau Double.Parse(șir)
șir → float    float.Parse(șir) sau Float.Parse(șir)

```

Observație: În cazul în care șirul de caractere nu reprezintă un număr valid, conversia acestui șir la număr va eșua.

Exemplul 18:

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Exemplul_18
{
    class Program
    {
        static void Main(string[] args)
        {
            string s;
            const int a = 13;
            const long b = 100000;
            const float c = 2.15F;
            double d = 3.1415;

            Console.WriteLine("CONVERSII\n");
            Console.WriteLine("TIP\tVAL. \tSTRING");
            Console.WriteLine("-----");
            s = "" + a;
            Console.WriteLine("int\t{0} \t{1}", a, s);
            s = "" + b;
            Console.WriteLine("long\t{0} \t{1}", b, s);
            s = "" + c;
            Console.WriteLine("float\t{0} \t{1}", c, s);
            s = "" + d;
            Console.WriteLine("double\t{0} \t{1}", d, s);
            Console.WriteLine("\nSTRING\tVAL \tTIP");
            Console.WriteLine("-----");
            int al;
            al = int.Parse("13");

```

```

            Console.WriteLine("{0}\t{1}\tint", "13", al);
            long b2;
            b2 = long.Parse("1000");
            Console.WriteLine("{0}\t{1} \tlong", "1000", b2);
            float c2;
            c2 = float.Parse("2,15");
            Console.WriteLine("{0}\t{1} \tfloat", "2,15", c2);
            double d2;
            d2 = double.Parse("3.1415",
System.Globalization.CultureInfo.InvariantCulture);

            Console.WriteLine("{0}\t{1}\tdouble", "3.1415", d2);
            Console.ReadKey();
        }
    }
}

```

```

C:\Windows\system32\cmd.exe
CONVERSII
TIP      VAL.      STRING
int       13        13
long      100000    100000
float     2.15      2.15
double    3.1415    3.1415
STRING   VAL      TIP
13        13        int
1000      1000      long
2.15      2.15      float
3.1415    3.1415    double

```

Tipuri de date

În C# există două categorii de tipuri de date:

-tipuri valoare

- tipul simplu predefinit: **byte, char, int, float** etc.
- tipul enumerare – **enum**
- tipul structură - **struct**

- tipuri referință

- tipul clasă – **class**
- tipul interfață – **interface**
- tipul delegat – **delegate**
- tipul tablou - **array**

- **Observație:** Toate tipurile de date sunt derivate din tipul **System.Object**

- Toate **tipurile valoare** sunt derivate din clasa **System.ValueType**, derivată la rândul ei din clasa **Object** (alias pentru **System.Object**).

- Pentru **tipurile valoare**, declararea unei variabile implică și alocarea de spațiu. Dacă inițial, variabilele conțin valoarea implicită specifică tipului, la atribuire, se face o copie a datelor în variabila destinație care nu mai este legată de variabila inițială. Acest proces se numește transmitere prin valoare, sau **value semantics**.

Exemplul 19:

```
using System;
using System.Collections.Generic;
using System.Text;

namespace ExempluTipuriValoare
{
    public struct Intreg
    {
        public int v;
    }
    class Program
    {
        static void Main(string[] args)
        {
            Intreg sa = new Intreg();
            sa.v = 13;
            Intreg sb = sa;
            // se initializeaza prin copiere variabila sb
            Console.WriteLine("sa.v este {0}.", sa.v);
            Console.WriteLine("sb.v este {0} prin initializare.", sb.v);
            sa.v = 10;
            Console.WriteLine("sa.v este {0}.", sa.v);
            Console.WriteLine("sb.v este {0}.", sb.v);
            Console.ReadLine();
        }
    }
}
```



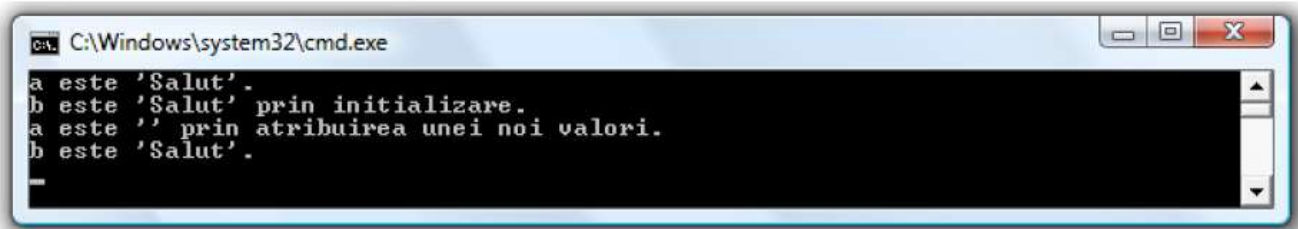
- Spre deosebire de tipurile valoare, pentru **tipurile referință**, declararea unei variabile nu implică automat alocarea de spațiu: inițial, referințele sunt **null** și trebuie alocată explicit memorie pentru obiectele propriu-zise. În plus, la atribuire, este copiată referința în variabila destinație, dar obiectul spre care indică rămâne același (**aliasing**). Aceste reguli poartă denumirea de **reference semantics**.

Exemplul 20: Pentru exemplificarea celor de mai sus, pentru tipurile referință, vom folosi clasa

StringBuilder.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ExempluTipuriReferinta
{
    class Program
    {
        static void Main(string[] args)
        {
            StringBuilder a = new StringBuilder();
            StringBuilder b = a;
            a.Append("Salut");
            Console.WriteLine("a este '{0}'.", a);
            Console.WriteLine("b este '{0}' prin initializare.", b);
            a = null;
            Console.WriteLine("a este '{0}' prin atribuirea unei noi
valori.", a);
            Console.WriteLine("b este '{0}'.", b);
            Console.ReadLine();
        }
    }
}
```



```
cmd. C:\Windows\system32\cmd.exe
a este 'Salut'.
b este 'Salut' prin initializare.
a este '' prin atribuirea unei noi valori.
b este 'Salut'.
```

Tipul valoare

Tipuri predefinite

Limbajul C# conține un set de **tipuri predefinite** (**int**, **bool** etc.) și permite definirea unor tipuri proprii (**enum**, **struct**, **class** etc.).

Tipuri simple predefinite

Tip	Descriere	Alias pentru tipul struct din spațiul de nume System
<code>object</code>	rădăcina oricărui tip	
<code>string</code>	secvență de caractere Unicode	<code>System.String</code>
<code>sbyte</code>	tip întreg cu semn, pe 8 biți	<code>System.Sbyte</code>
<code>short</code>	tip întreg cu semn, pe 16 biți	<code>System.Int16</code>
<code>int</code>	tip întreg cu semn pe, 32 biți	<code>System.Int32</code>
<code>long</code>	tip întreg cu semn, pe 64 de biți	<code>System.Int64</code>
<code>byte</code>	tip întreg fără semn, pe 8 biți	<code>System.Byte</code>
<code>ushort</code>	tip întreg fără semn, pe 16 biți	<code>System.UInt16</code>
<code>uint</code>	tip întreg fără semn, pe 32 biți	<code>System.UInt32</code>
<code>ulong</code>	tip întreg fără semn, pe 64 biți	<code>System.UInt64</code>
<code>float</code>	tip cu virgulă mobilă, simplă precizie, pe 32 biți (8 pentru exponent, 24 pentru mantisă)	<code>System.Single</code>
<code>double</code>	tip cu virgulă mobilă, dublă precizie, pe 64 biți (11 pentru exponent, 53 pentru mantisă)	<code>System.Double</code>
<code>bool</code>	tip boolean	<code>System.Boolean</code>
<code>char</code>	tip caracter din setul Unicode, pe 16 biți	<code>System.Char</code>
<code>decimal</code>	tip zecimal, pe 128 biți (96 pentru mantisă), 28 de cifre semnificative	<code>System.Decimal</code>

Domeniul de valori pentru tipurile numerice:

Tip	Domeniul de valori
<code>sbyte</code>	-128; 127
<code>short</code>	-32768; 32767
<code>int</code>	-2147483648; 2147483647
<code>long</code>	-9223372036854775808; 9223372036854775807
<code>byte</code>	0; 255
<code>ushort</code>	0; 65535
<code>uint</code>	0; 4294967295
<code>ulong</code>	0; 18446744073709551615
<code>float</code>	-3.402823E+38; 3.402823E+38
<code>double</code>	-1.79769313486232E+308; 1.79769313486232E+308
<code>decimal</code>	-79228162514264337593543950335; 79228162514264337593543950335

O valoare se asignează după următoarele reguli:

Sufix	Tip
nu are	<code>int</code> , <code>uint</code> , <code>long</code> , <code>ulong</code>
<code>u</code> , <code>U</code>	<code>uint</code> , <code>ulong</code>
<code>L</code> , <code>l</code>	<code>long</code> , <code>ulong</code>
<code>ul</code> , <code>lu</code> , <code>Ul</code> , <code>lU</code> , <code>UL</code> , <code>LU</code> , <code>Lu</code>	<code>ulong</code>

Exemplul 21:

```
string s = "Salut!";
long a = 10;
long b = 13L;
ulong c = 12;
ulong d = 15U;
ulong e = 16L;
ulong f = 17UL;

float g = 1.234F;
double h = 1.234;
double i = 1.234D;
bool cond1 = true;
bool cond2 = false;
decimal j = 1.234M;
```

Tipul enumerare

Tipul enumerare, asemănător cu cel din C++, se definește de către utilizator. Acest tip permite utilizarea numelor care, sunt asociate unor valori numerice.

- Enumerările nu pot fi declarate abstracte și nu pot fi derivate. Orice **enum** este derivat automat din clasa **System.Enum**, derivată din **System.ValueType**.

În cazul în care nu se specifică tipul enumerării, acesta este considerat implicit **int**.

Specificarea tipului se face după numele enumerării:

```
[attribute][modificatori]enum NumeEnumerare [: Tip]
{
    lista
}
```

În ceea ce urmează, vom considera **enum** fără elementele opționale.

Folosirea tipului enumerare impune următoarele observații:

- în mod implicit, valoarea primului membru al enumerării este 0, iar fiecare variabilă care urmează are valoarea (implicită) mai mare cu o unitate decât precedentă.
- valorile folosite pentru inițializări trebuie să facă parte din domeniul de valori al tipului **enum**
- nu se admit referințe circulare

```
enum ValoriCirculare
{
    a = b,
    b
}
```

În acest exemplu, **a** depinde explicit de **b**, iar **b** depinde de **a** implicit

Asemănător celor cunoscute din C++, tipul structură poate să conțină declarații de constante, câmpuri, metode, proprietăți, indexatori, operatori, constructori sau tipuri imbricate.

Exemplul 22:

```
using System;
namespace tipulEnum
{
    class Program
    {
        enum lunaAnului
        {
            Ianuarie = 1,
            Februarie,
            Martie,
            Aprilie,
            Mai,
            Iunie,
            Iulie,
            August,
            Septembrie,
            Octombrie,
            Noiembrie,
            Decembrie
        }

        static void Main(string[] args)
        {
            Console.WriteLine("Luna Mai este a: {0}", (int)lunaAnului.Mai + " luna din an.");
            Console.ReadLine();
        }
    }
}
```



Tipuri nulabile

Tipurile nulabile, **nullable**, sunt tipuri valoare pentru care se pot memora valori posibile din aria tipurilor de bază, eventual și valoarea **null**.

Am văzut mai sus că pentru tipurile valoare, la declararea unei variabile, aceasta conține valoarea implicită a tipului. Sunt cazuri în care se dorește ca, la declarare, valoarea implicită a variabilei să fie nedefinită.

În C# există o astfel de posibilitate, folosind structura **System.Nullable<T>**.

Concret, o declarație de forma:

```
System.Nullable<T> var;
```

este echivalentă cu

```
T? var;
```

unde **T** este un tip valoare.

Aceste tipuri nulabile conțin două proprietăți:

- proprietate **HasValue**, care indică dacă valoarea internă este diferită sau nu de **null**
- proprietatea **Value**, care va conține valoarea propriu zisă.
- Legat de această noțiune, s-a mai introdus operatorul binar **??**

```
a ?? b
```

cu semnificația: dacă **a** este **null** b este evaluat și constituie rezultatul expresiei, altfel rezultatul este a.