Programarea Aplicațiilor Windows – curs 1

Conf. dr. Cristian CIUREA
Departamentul de Informatică și Cibernetică Economică
Academia de Studii Economice București
cristian.ciurea@ie.ase.ro

Agenda

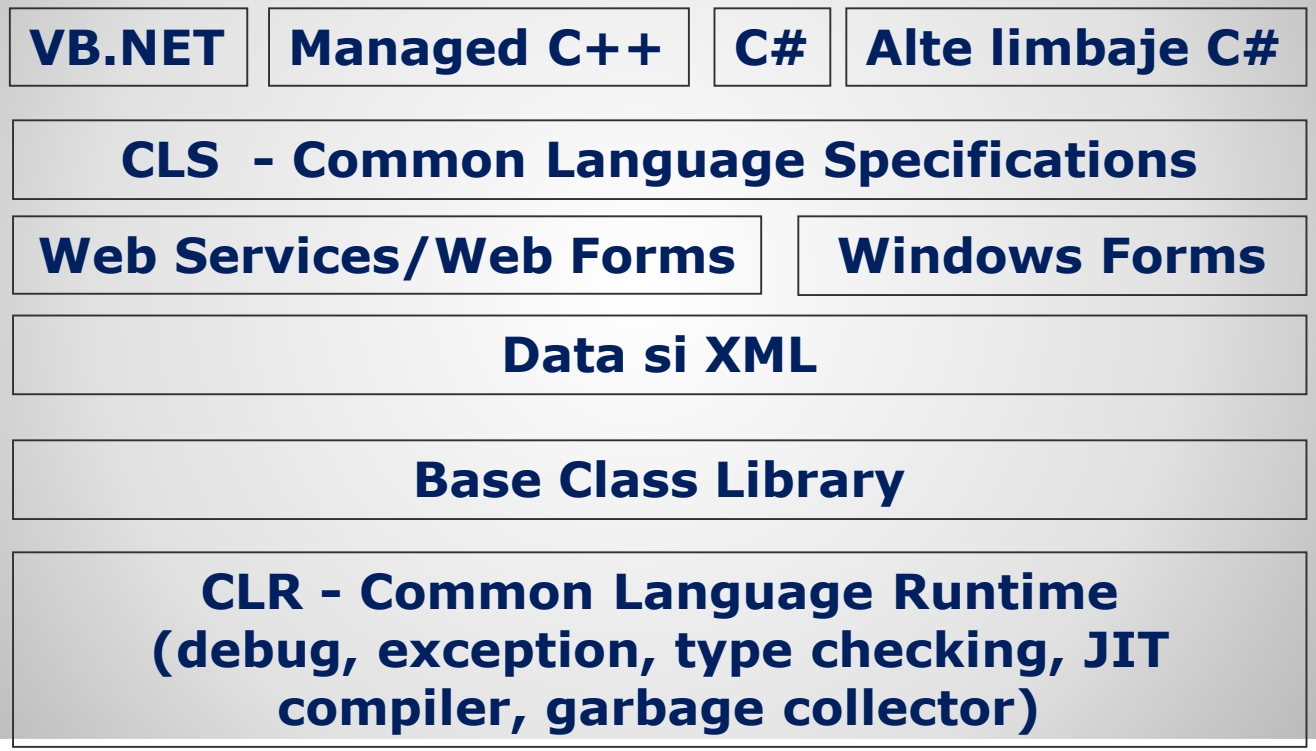
1. Structură evaluare
2. Platforma .NET
3. Diferențe C++ vs. C#
4. Tipuri de date în C#
5. Boxing / unboxing
6. Transferul parametrilor
7. Masive în C#
8. Colecții în C#

Evaluare

- **seminar = 40%**
  - lucrare calculator = 30%
  - proiect (obligatoriu) = 10%
- **curs = 60%**
  - examen la calculator = 60%

Detalii la http://acs.ase.ro/paw

Platforma .NET



Platforma .NET

Sursa
C# / J# / Managed C++, VB .NET

Compilare – csc.exe, vbc.exe, cl.exe

IL – Interpreted Language

Procedura de asamblare – csc.exe, vbc.exe, cl.exe

PE – Portable Executable
(EXE sau DLL)
Assembly

Interpretat de CLR

Mixed Programming

C#

```
namespace Math{
    public class BasicOp{
        public static double Add(double op1, double
      op2){
            return op1 + op2;
        }
        public static double Multiply(double op1,
      double op2){
            return op1 * op2;
        }
    }
}
```

csc /t:library math.cs

↓

math.dll

VB

```
Imports Math
Imports System

Module MathTest
    Sub Main()
        Dim vb1 As Double = 20
        Dim vb2 As Double = 30

        Console.WriteLine("Suma celor 2 variabile este {0}, iar
       produsul este {1}.", BasicOp.Add(vb1, vb2),
       BasicOp.Multiply(vb1, vb2))
    End Sub
End Module
```

vbc /r:math.dll mathtest.vb

↓

mathtest.exe

Diferențe C++ vs. C#

1. În C++ se definesc metode și variabile la nivel global, iar în C# se definesc metode și variabile doar în clase și structuri.

2. În C++ se utilizează referințe și pointeri, în C# se utilizează referințe, iar pointerii se utilizează doar în modul de lucru "unsafe".

Diferențe C++ vs. C#

```
unsafe
{
      int* ptr1, ptr2;
      ptr1 = &var1;
      ptr2 = ptr1;
      *ptr2 = 20;
}
```

csc.exe C:\pointeri.cs /unsafe

Diferențe C++ vs. C#

3. În C++ tipul șir de caractere nu este implementat ca tip fundamental. În C# tipul șir de caractere este introdus prin clasa **String**, care este derivată din clasa *Object* și are următoarele caracteristici:

- este o clasă **sealed**, adică din ea nu mai poate deriva o altă clasă;

- este **immutable**, adică un obiect odată creat nu mai poate fi modificat.

Diferențe C++ vs. C#

4. În C++ este permisă moștenirea multiplă (clasa Pegas era derivată din clasa Pasăre și clasa Cal). În C# nu este permisă moștenirea multiplă la nivel de clasă concretă, dar este permisă derivarea dintr-o clasă și mai multe interfețe.

```
class Urs: Animal, ICloneable, IComparable //clasa Urs este derivata din clasa Animal si interfetele ICloneable si IComparable
```

C++ vs. C#

- **obiecte gestionate prin valoare și referințe;**
- o clasă poate conține atribute dinamice gestionate de pointeri;
- destructorul are rol de a dezaloca spațiul ocupat și pentru a evita memory leak-uri;
- necesitate definire operator = și constructor copiere pentru a evita copierea implicită;
- string gestionat prin char*;

- **obiecte gestionate doar prin referințe;**
- pointerii din C++ definiți cu * sunt considerați *unsafe* si sunt indicați doar pentru lucru cu COM-uri;
- dezalocarea memoriei se face de către GC din CLR;
- operatorul = nu se mai supraincarca;
- operatorul = implicit realizeaza *shallow copy*;
- constructorul de copiere necesar pentru a face *deep copy*;
- un nou tip valoric – string;

Tipuri de date în C#

a) **tipuri valorice**, care sunt manipulate direct prin numele lor; tipurile valorice includ tipurile de bază și cele introduse prin "struct" și "enum".

b) **tipuri referențiale**, care sunt manipulate prin referințe; tipurile referențiale includ tipurile introduse prin "class", "interface" și "delegate".

Tipuri de date în C#

- **struct** example:

```
struct Student
{
      public int cod;
      private string nume;
      public bool integralist;
};
```

- **enum** example:

```
enum Importance
{
      None, Regular, Important, Critical
};
```

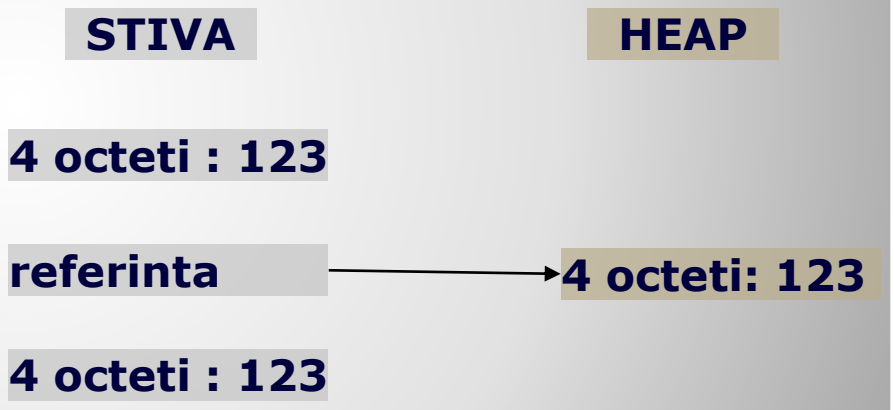
Boxing / unboxing

Trecerea din tip valoric în tip referențial și invers se face prin împachetare (boxing) și despachetare (unboxing).

```
int i = 123;
object o = i; //boxing
int j = (int)o; //unboxing
```

Boxing / unboxing

```
class BoxUnbox
{
        static void Main()
        {
            int i = 123;
            object obj= i;
            int j = (int)obj;
        }
}
```



Pointeri/Referințe – Transfer parametrii

- subprogramale pot primi parametrii prin valoare sau prin referință;
- prin valoare => copierea valorii parametrului pe stiva funcției;
- prin referință => copierea adresei parametrului pe stiva funcției;
- transferul parametrilor prin pointeri este considerat unsafe de către compilator.

Masive în C#

În C#, masivele sunt tipuri referenţiale derivate din clasa abstractă **System.Array**.

Elementele masivelor pot fi de orice tip suportat (tipuri referenţiale, tipuri valorice, alte masive) şi sunt accesate prin intermediul indicilor.

Dimensiunea masivelor este stabilită la crearea acestora (la rulare) şi nu poate fi modificată pe parcurs. Limbajul suportă atât masive unidimensionale, cât şi multidimensionale.

Masive în C#

SINTAXA: `tip_date [ ] nume_masiv;`

- reprezintă colecții indexate de obiecte;
- diferit de masivul din C/C++;
- un obiect derivat din clasa Array;
- moștenește o serie de metode:
  - `BinarySearch()`
  - `Clear()`
  - `CopyTo()`
  - `Sort()`
  - `Clone()`
- moștenește o serie de proprietăți:
  - `Length`

Masive în C#

Vectorii de tipuri valorice – numele vectorului este o referință, iar la capătul referinței sunt valori.

```
int [] v;
v=new int[4];
v=new int[4] {1,2,3,4};
v={1,2,3,4};
```

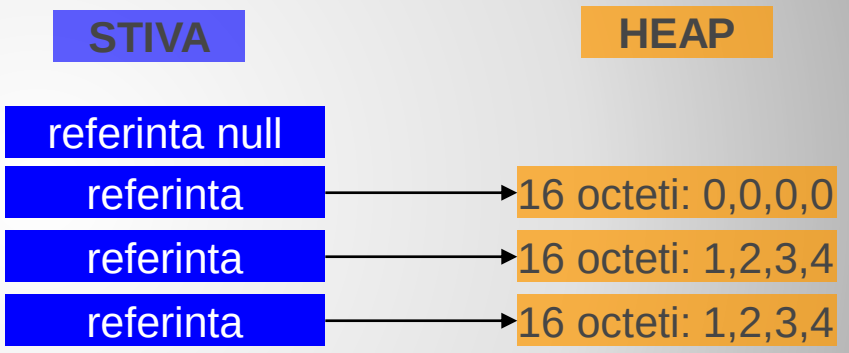
Masive în C#

Vectorii de obiecte – numele vectorului este o referință, iar la capătul referinței sunt alte referințe la zone de memorie statică în care au fost alocate și instanțiate obiecte.

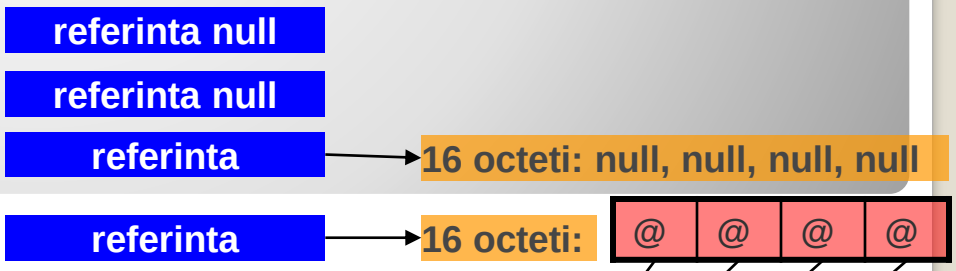
```
Persoana [] p;
p=new Persoana[4];
p=new Persoana[4] {p1,p2,p3,p4};
```

Masive în C#

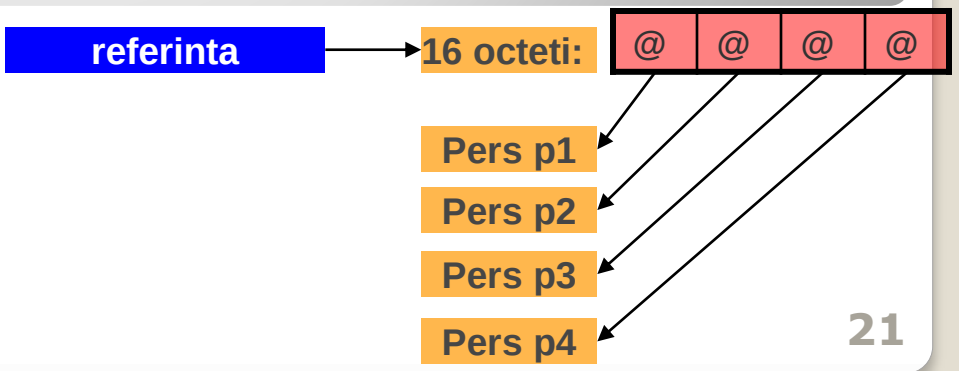
```
int [ ] vect;
vect = new int[4];
int [ ] vect2 = {1,2,3,4};
int [ ] vect3 = new int[4] {1,2,3,4};
```




```
Pers p1;
Pers [ ] vPers;
vPers = new Pers[4];
```




```
vPers = new Pers[4] {p1,p2,p3,p4};
```



Masive în C#

Masivele bidimensionale:

```
int [,] mb1 = new int[2,3];
int [,] mb2 = {{1,2},{3,4},{5,6}};
```

La declararea unui masiv bidimensional, dimensiunile se specifică la alocarea cu operatorul "new" sau se deduc din lista de inițializatori.

Masive în C#

Masiv în scară (jagged sau în zig-zag)

```
int [][] ms = new int[3][];
```

Masivul în scară necesită la alocare doar precizarea numărului de linii, efectuând alocări doar pentru liniile respective. Ulterior se fac alocări pentru fiecare linie în parte, acestea putând avea dimensiuni diferite.

Colecții în C#

Vectorii au dezavantajul că au dimensiune fixă; chiar dacă dimensiunea se specifică la execuție și nu la compilare, ea tot fixă rămâne.

```
Student[] vs = new Student[3] {s1, s2, s3};
```

Colecțiile sunt redimensionabile dinamic, ele alocându-se element cu element.

```
ArrayList lista = new ArrayList();
```

Colecții în C#

Elementele unui vector sunt de un anumit tip, în timp ce elementele unei colecții sunt de tip generic "object"; rezultă că o colecție poate conține elemente de orice tip, cu condiția ca toate elementele colecției să fie de același tip.

```
lista.Add(s1);
lista.Add(s2);
lista.Add(s3);
lista.Add(s4);
```

Colecții în C#

Cele mai utilizate colecții sunt: ArrayList (lista), SortedList (map), Queue (coada) și Stack (stiva).

Atât vectorii, cât și colecțiile pot fi parcurse cu "foreach", cât și cu "for", dimensiunea vectorului fiind dată de proprietatea **Length**, iar a colecției de proprietatea **Count**.

Colecții în C#

```
for (int i = 0; i < v.Length; i++)
{
     Console.WriteLine("Lungimea liniei {0}
     este {1}", i, v[i].Length);
}

foreach (Animal a in z1.AnimalList)
{
     Console.WriteLine(a.Name.ToString());
}
```

Bibliografie

[1] I. Smeureanu, M. Dârdală, A. Reveiu – *Visual C# .NET*, Editura CISON, București, 2004.

[2] C. Petzold – *Programming Microsoft Windows with C#*, Microsoft Press, 2002.

[3] L. O'Brien, B. Eckel – *Thinking in C#*, Prentice Hall.

[4] J. Richter – *Applied Microsoft .NET Framework Programming*, Microsoft Press, 2002.

[5] http://acs.ase.ro/paw