

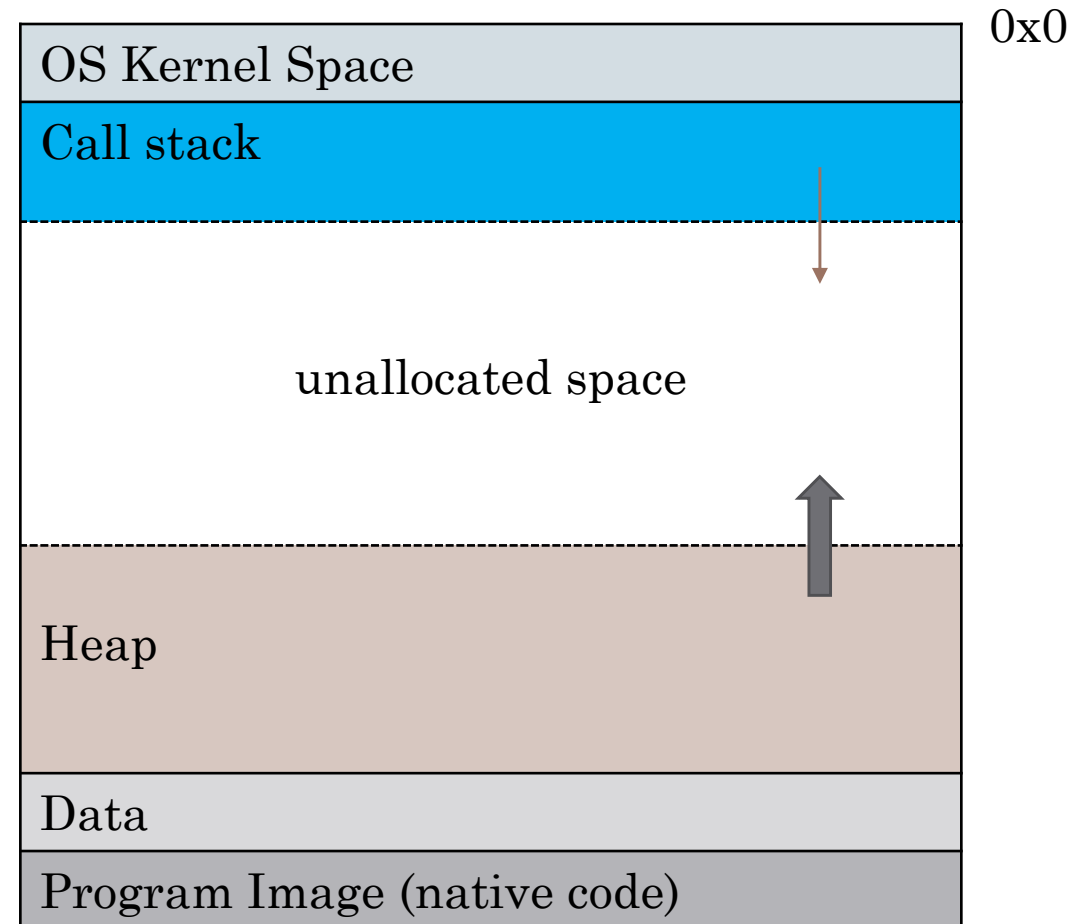
C# .NET

Laborator 3

Managementul memoriei

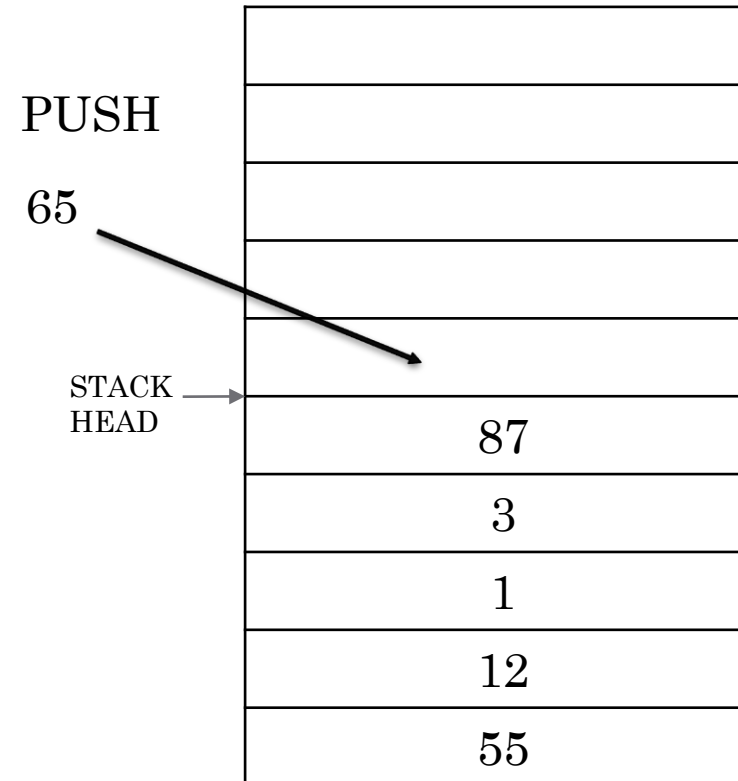
Managementul memoriei

- Kernel Space – zona rezervata memoriei sistemului
- Program Image – codul programului in format nativ
- Data – DLL-uri, alte informatii si date de care programul are nevoie pt. a rula
- Stack – stiva de program
 - Default :1MB
 - Poate creste pana la max 1GB (legenda spune ca)
- Heap – memoria dinamica
 - 32b OS – 1.5GB
 - 64b OS - ∞



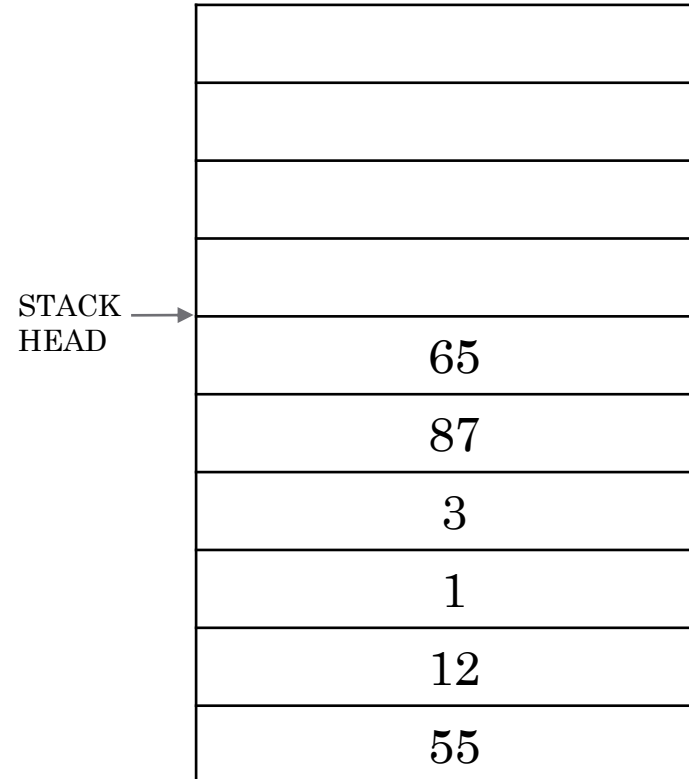
Stiva

- Structura de memorie
- FILO / LIFO - last in first out
 - Push adauga un element in stiva
 - Pop- extrage un element din stiva
- Top of the stack
 - Adaugarea intotdeauna “on top”
 - Extragerea intotdeauna “from top”



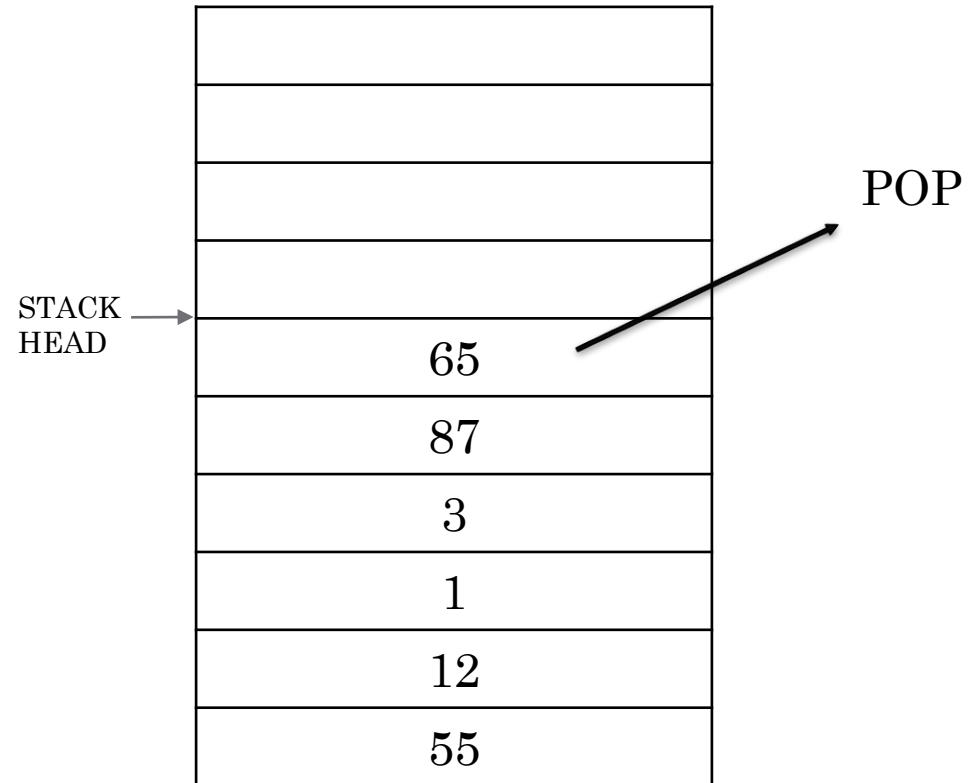
Stiva

- Structura de memorie
- FILO / LIFO - last in first out
 - Push adauga un element in stiva
 - Pop- extrage un element din stiva
- Top of the stack
 - Adaugarea intotdeauna “on top”
 - Extragerea intotdeauna “from top”



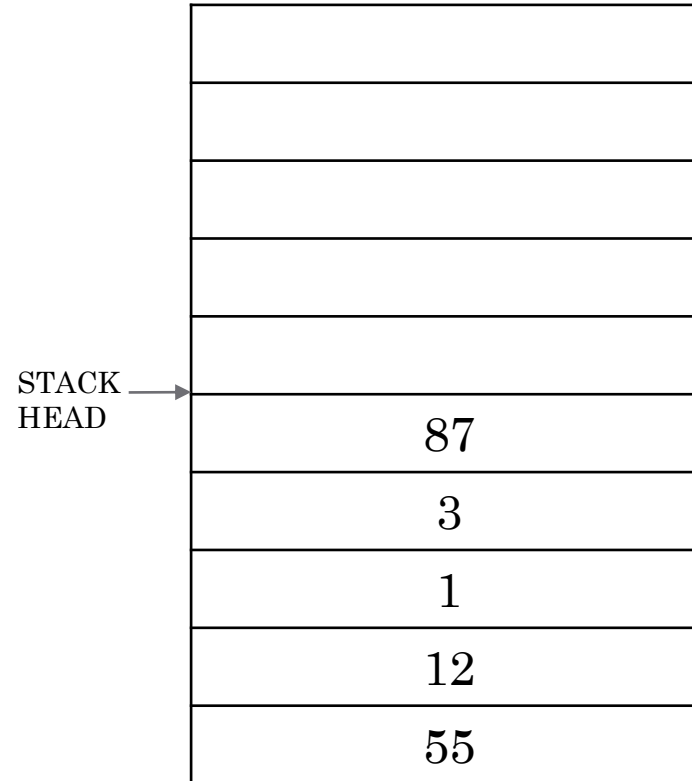
Stiva

- Structura de memorie
- FILO / LIFO - last in first out
 - Push adauga un element in stiva
 - Pop- extrage un element din stiva
- Top of the stack
 - Adaugarea intotdeauna “on top”
 - Extragerea intotdeauna “from top”



Stiva

- Structura de memorie
- FILO / LIFO - last in first out
 - Push adauga un element in stiva
 - Pop- extrage un element din stiva
- Top of the stack
 - Adaugarea intotdeauna “on top”
 - Extragerea intotdeauna “from top”



Heap

- memoria dinamica
- 32b OS – 1.5GB
- 64b OS - ∞
- Limbaje unmanaged
 - Memory leaks
- Limbaje managed
 - Garbage collector
- Obiecte de dimensiuni mari
 - obiecte grafice, buffersm Vectori, matrici, siruri de caractere etc.
- Obiecte managed



Heap

```
int integ = 42;  
int[] vectorIntegi = new int[] { 1, 2, 3 };
```

Stack

[illegible]

Heap

```
int integ = 42;
int[] vectorIntegri = new int[]{ 1, 2, 3 };
```

Stack
42

[illegible]

Heap

```
int integ = 42;
int[] vectorIntegri = new int[]{ 1, 2, 3 };
```

Stack
42

[illegible]

Heap

```
int integ = 42;
int[] vectorIntegri = new int[]{ 1, 2, 3 };
```

Stack
42
0xC0FFEE

← REFERENCE

[illegible]

Heap

```
int intreg = 42;  
int[] vectorIntregi = new int[] { 1, 2, 3 };
```

Stack
42
0xC0FFEE

Heap

1
2
3



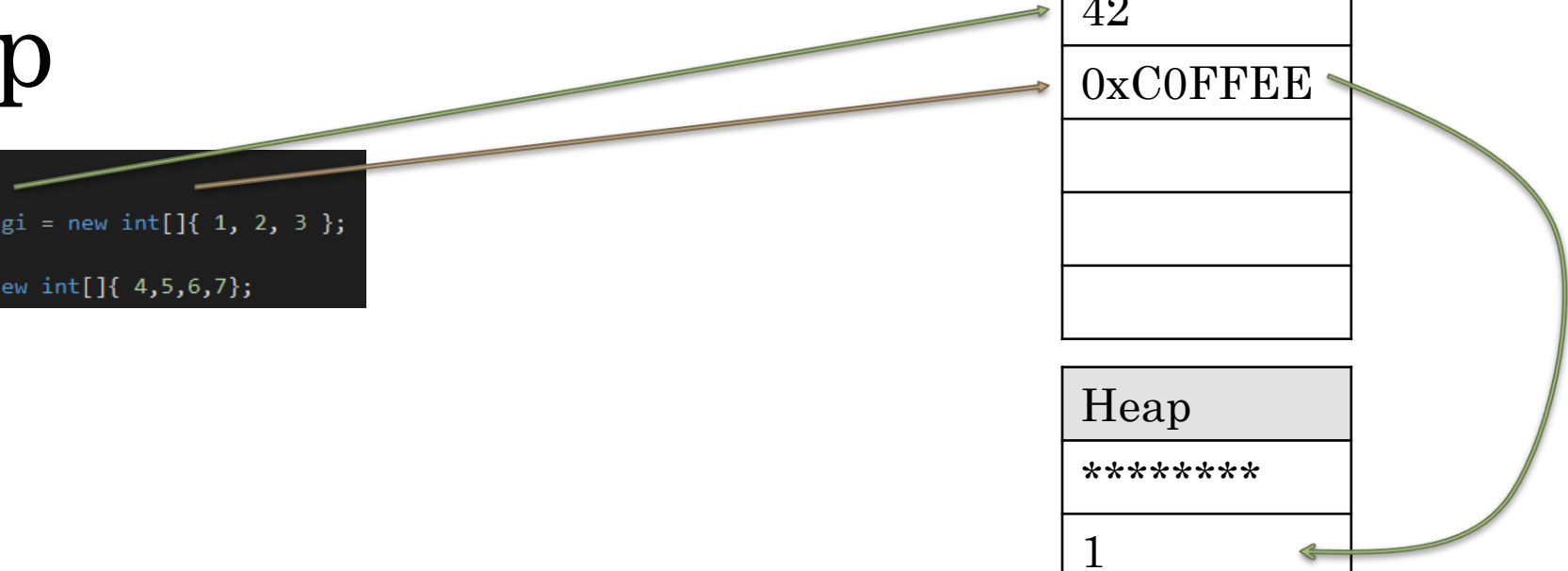
Heap

```
int integ = 42;  
int[] vectorIntegi = new int[]{ 1, 2, 3 };  
vectorIntegi = new int[]{ 4,5,6,7};
```

Stack
42
0xC0FFEE

Heap

1
2
3



Heap

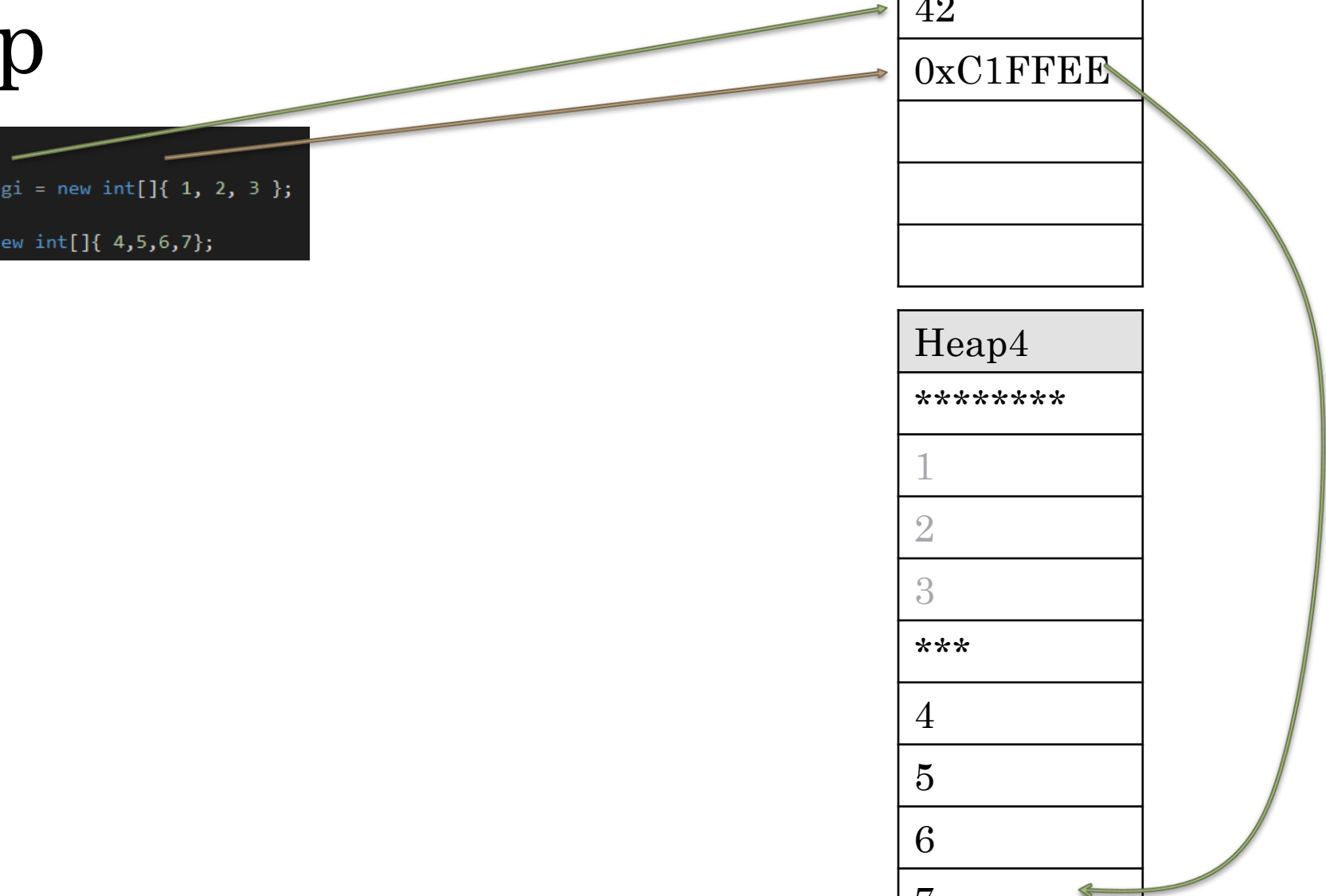
```
int intreg = 42;  
int[] vectorIntregi = new int[]{ 1, 2, 3 };  
vectorIntregi = new int[]{ 4,5,6,7};
```

Stack
42
0xC1FFEE

Heap4

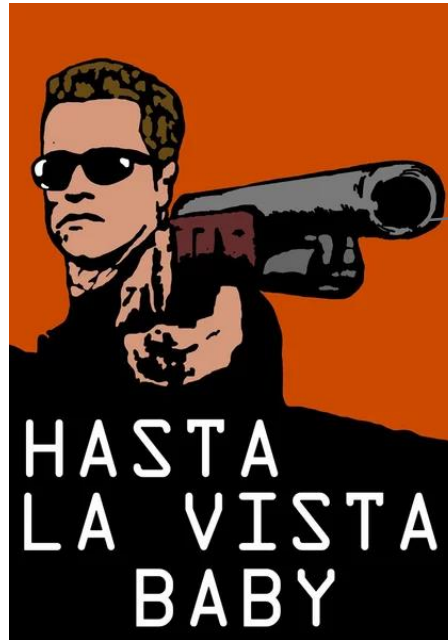
1
2
3

4
5
6
7



Heap

```
int integ = 42;  
int[] vectorIntregi = new int[]{ 1, 2, 3 };  
vectorIntregi = new int[]{ 4,5,6,7};
```



Garabage Collector

Stack
42
0xC1FFEE

Heap4

1
2
3

4
5
6
7

Heap

```
int intreg = 42;  
int[] vectorIntregi = new int[]{ 1, 2, 3 };  
vectorIntregi = new int[]{ 4,5,6,7};
```



Garabage Collector

Stack
42
0xC1FFEE
Heap4

4
5
6
7

Value type vs reference type

- Value type
 - Alocate pe stiva
 - Tipuri simple, predefinite
 - Int
 - Double
 - Struct
 - Bool
 - Char
 - Transmise prin valoare
 - Exceptie out (bad practice)
 - Dealocarea memoriei
 - Automata
- Reference type
 - Alocate in Heap
 - Variabila e pe stiva
 - string
 - Vectori
 - Obiecte
 - Etc...
 - Transmise prin referinta
 - Exceptie string
 - Dealocarea memoriei
 - Manuala
 - Garbage collector

Value type vs reference type

- Value type
 - Alocate pe stiva
 - Tipuri simple, predefinite
 - Int
 - Double
 - Struct
 - Bool
 - Char
 - Transmise prin valoare
 - Exceptie out (bad practice)
 - Dealocarea memoriei
 - Automata
- Reference type
 - Alocate in Heap
 - Variabila e pe stiva
 - string
 - Vectori
 - Obiecte
 - Etc...
 - Transmise prin referinta
 - Exceptie string
 - Dealocarea memoriei
 - Manuala
 - Garbage collector



Stiva apelurilor

- Aka Call stack
- Metodele sunt adaugate in ordinea executiei
 - Main, Factorial
- Este alocat spatiu de memorie in stiva pentru fiecare variabila in parte pentru fiecare metoda in parte
 - Cele doua n-uri sunt zone de memorie diferite
- Este alocat spatiu de memorie in stiva pentru fiecare parametru in parte pentru fiecare apel in parte!
- La finalul executiei unei functii
 - Functia e "Extrasă" din stiva
 - Zona de memorie aferenta se elibereaza
 - Executia continua in functia apelanta
- Stack overflow

```
0 references
static void Main(string[] args)
{
    Console.WriteLine("introduceti n");
    int n = int.Parse(Console.ReadLine());

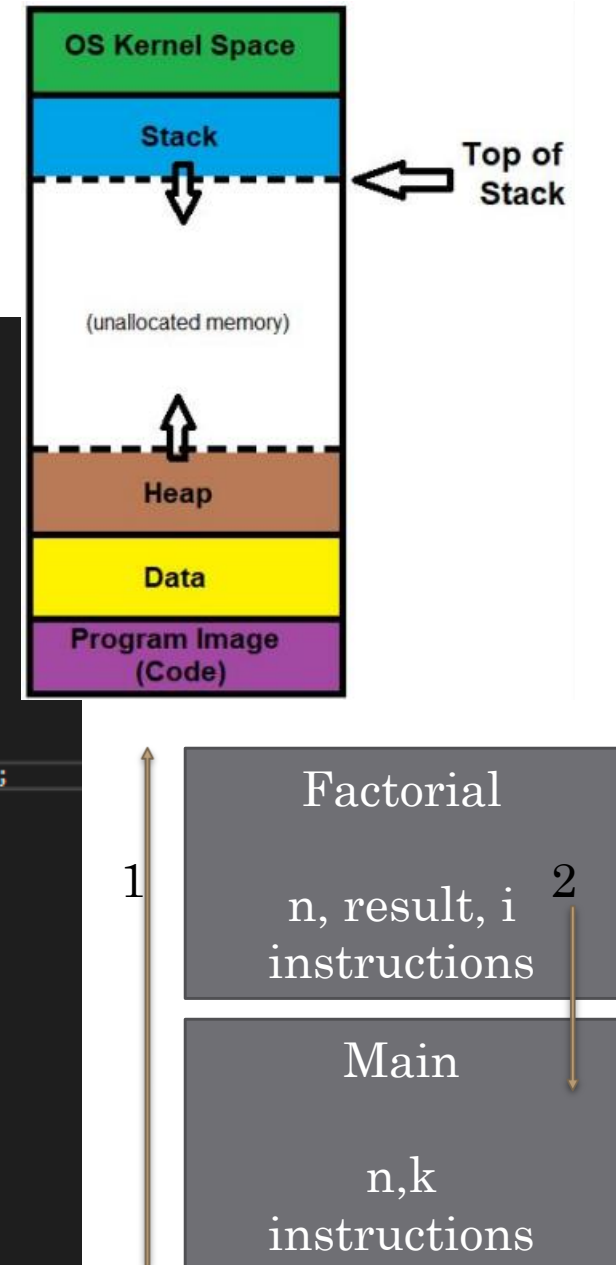
    Console.WriteLine("introduceti k");
    int k = int.Parse(Console.ReadLine());

    if (n <= 0 || k <= 0 || n <= k)
    {
        Console.WriteLine("numere invalide ");
        return;
    }

    double aranjamente = (double)Factorial(n) / Factorial(n - k);
    double combinari = aranjamente / Factorial(k);

    Console.WriteLine("aranjamente " + aranjamente);
    Console.WriteLine("combinari " + combinari);
}

3 references
static int Factorial(int n)
{
    int result = 1;
    for (var i = 1; i <= n; i++)
    {
        result *= i;
    }
    return result;
}
```



Call stack

```
static void Main(string[] args)
{
    int n = CitesteIntreg();
    int k = CitesteIntreg();

    int combinari = Combinari(n, k);
}

1 reference
private static int Combinari(int n, int k)
{
    return Fact(n) / (Fact(k) * Fact(n - k));
}

3 references
private static int Fact(int n)
{
    int i;
    int result;
    for (i = 1, result = 1; i <= n; i++)
    {
        result *= i;
    }
    return result;
}
```


Call stack

```
static void Main(string[] args)
{
    int n = CitesteIntreg();
    int k = CitesteIntreg();

    int combinari = Combinari(n, k);
}

1 reference
private static int Combinari(int n, int k)
{
    return Fact(n) / (Fact(k) * Fact(n - k));
}

3 references
private static int Fact(int n)
{
    int i;
    int result;
    for (i = 1, result = 1; i <= n; i++)
    {
        result *= i;
    }
    return result;
}
```

STACK
HEAD →

Main{ int n, int k, int combinari }

Call stack

```
static void Main(string[] args)
{
    int n = CitesteIntreg();
    int k = CitesteIntreg();

    int combinari = Combinari(n, k);
}

1 reference
private static int Combinari(int n, int k)
{
    return Fact(n) / (Fact(k) * Fact(n - k));
}

3 references
private static int Fact(int n)
{
    int i;
    int result;
    for (i = 1, result = 1; i <= n; i++)
    {
        result *= i;
    }
    return result;
}
```

STACK
HEAD →

CitesteIntreg{ int __returnValue; }
Main{ int n, int k, int combinari }

Call stack

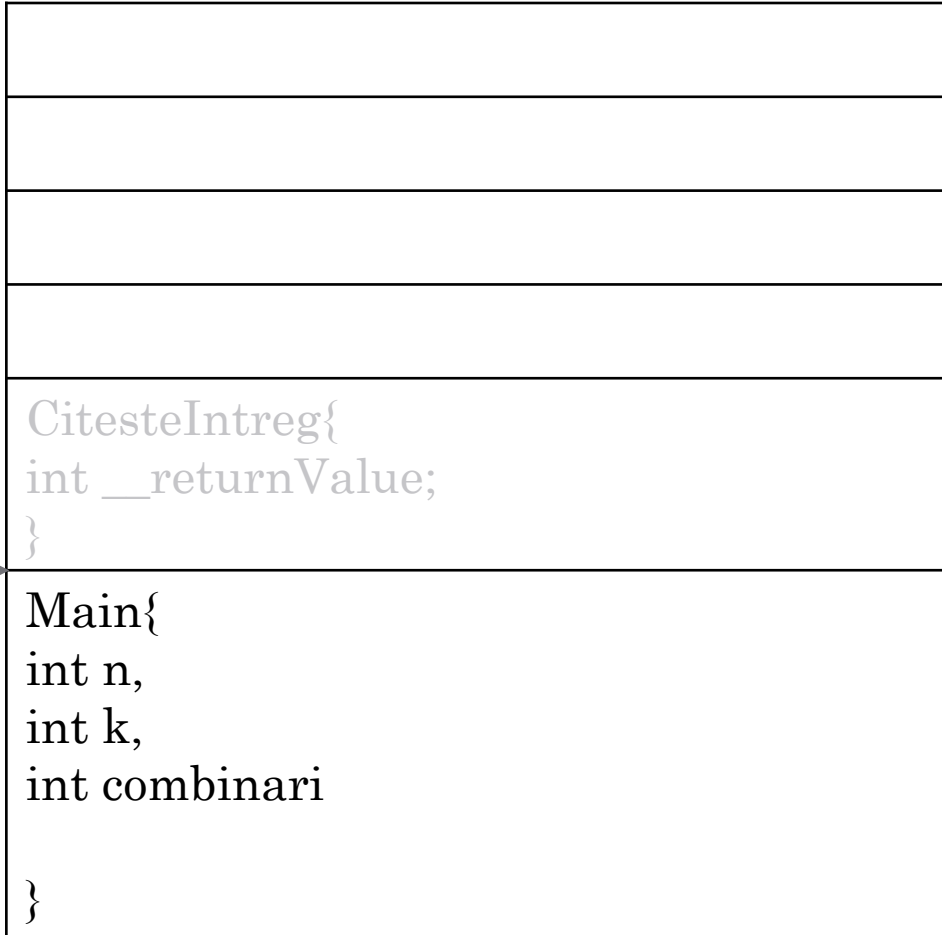
```
static void Main(string[] args)
{
    int n = CitesteIntreg();
    int k = CitesteIntreg();

    int combinari = Combinari(n, k);
}

1 reference
private static int Combinari(int n, int k)
{
    return Fact(n) / (Fact(k) * Fact(n - k));
}

3 references
private static int Fact(int n)
{
    int i;
    int result;
    for (i = 1, result = 1; i <= n; i++)
    {
        result *= i;
    }
    return result;
}
```

STACK
HEAD →



Call stack

```
static void Main(string[] args)
{
    int n = CitesteIntreg();
    int k = CitesteIntreg();

    int combinari = Combinari(n, k);
}

1 reference
private static int Combinari(int n, int k)
{
    return Fact(n) / (Fact(k) * Fact(n - k));
}

3 references
private static int Fact(int n)
{
    int i;
    int result;
    for (i = 1, result = 1; i <= n; i++)
    {
        result *= i;
    }
    return result;
}
```

STACK
HEAD →

CitesteIntreg{ int __returnValue; }
Main{ int n, int k, int combinari }

Call stack

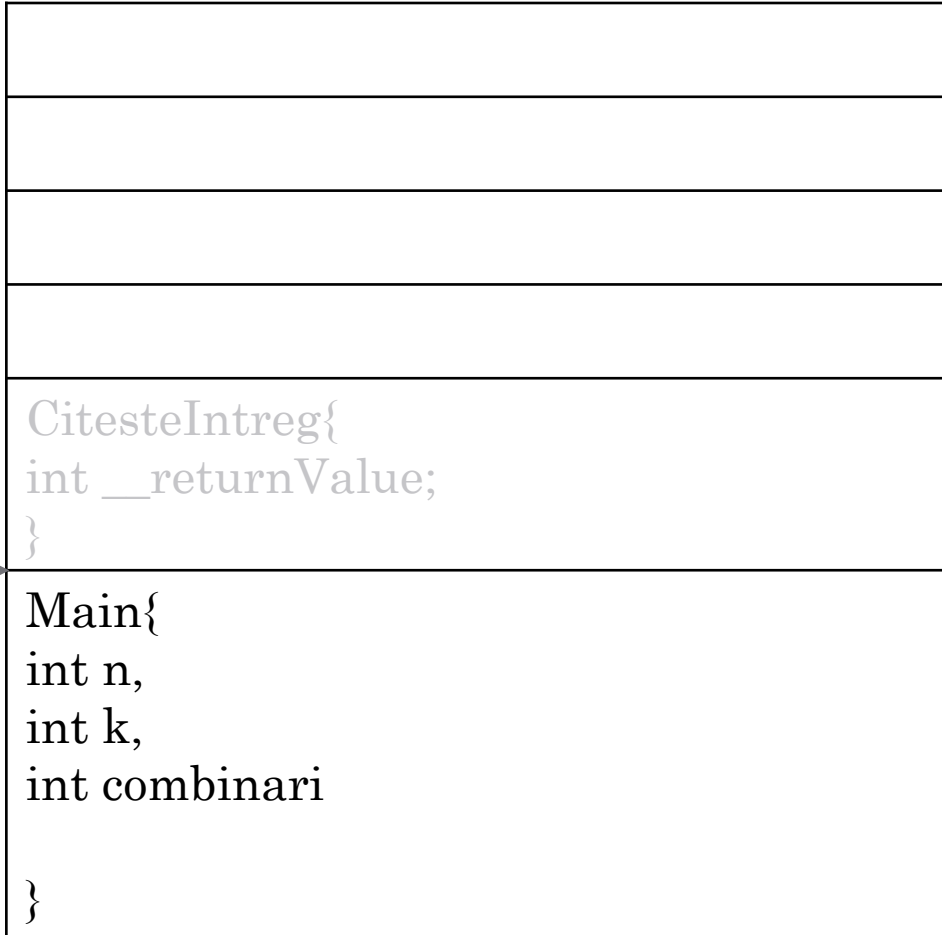
```
static void Main(string[] args)
{
    int n = CitesteIntreg();
    int k = CitesteIntreg();

    int combinari = Combinari(n, k);
}

1 reference
private static int Combinari(int n, int k)
{
    return Fact(n) / (Fact(k) * Fact(n - k));
}

3 references
private static int Fact(int n)
{
    int i;
    int result;
    for (i = 1, result = 1; i <= n; i++)
    {
        result *= i;
    }
    return result;
}
```

STACK
HEAD →



Call stack

```
static void Main(string[] args)
{
    int n = CitesteIntreg();
    int k = CitesteIntreg();

    int combinari = Combinari(n, k);
}

1 reference
private static int Combinari(int n, int k)
{
    return Fact(n) / (Fact(k) * Fact(n - k));
}

3 references
private static int Fact(int n)
{
    int i;
    int result;
    for (i = 1, result = 1; i <= n; i++)
    {
        result *= i;
    }
    return result;
}
```

STACK
HEAD →

Combinari{ int n, int k }
Main{ int n, int k, int combinari }

Call stack

```
static void Main(string[] args)
{
    int n = CitesteIntreg();
    int k = CitesteIntreg();

    int combinari = Combinari(n, k);
}

1 reference
private static int Combinari(int n, int k)
{
    return Fact(n) / (Fact(k) * Fact(n - k));
}

3 references
private static int Fact(int n)
{
    int i;
    int result;
    for (i = 1, result = 1; i <= n; i++)
    {
        result *= i;
    }
    return result;
}
```

STACK
HEAD →

Fact{ int n }
Combinari{ int n, int k }
Main{ int n, int k, int combinari }

Call stack

```
static void Main(string[] args)
{
    int n = CitesteIntreg();
    int k = CitesteIntreg();

    int combinari = Combinari(n, k);
}

1 reference
private static int Combinari(int n, int k)
{
    return Fact(n) / (Fact(k) * Fact(n - k));
}

3 references
private static int Fact(int n)
{
    int i;
    int result;
    for (i = 1, result = 1; i <= n; i++)
    {
        result *= i;
    }
    return result;
}
```

STACK
HEAD →

Fact{ int n }
Combinari{ int n, int k }
Main{ int n, int k, int combinari }

Call stack

```
static void Main(string[] args)
{
    int n = CitesteIntreg();
    int k = CitesteIntreg();

    int combinari = Combinari(n, k);
}

1 reference
private static int Combinari(int n, int k)
{
    return Fact(n) / (Fact(k) * Fact(n - k));
}

3 references
private static int Fact(int n)
{
    int i;
    int result;
    for (i = 1, result = 1; i <= n; i++)
    {
        result *= i;
    }
    return result;
}
```

STACK
HEAD →

Fact{ int n }
Combinari{ int n, int k }
Main{ int n, int k, int combinari }

Call stack

```
static void Main(string[] args)
{
    int n = CitesteIntreg();
    int k = CitesteIntreg();

    int combinari = Combinari(n, k);
}

1 reference
private static int Combinari(int n, int k)
{
    return Fact(n) / (Fact(k) * Fact(n - k));
}

3 references
private static int Fact(int n)
{
    int i;
    int result;
    for (i = 1, result = 1; i <= n; i++)
    {
        result *= i;
    }
    return result;
}
```

STACK
HEAD →

Fact{ int n }
Combinari{ int n, int k }
Main{ int n, int k, int combinari }

Call stack

```
static void Main(string[] args)
{
    int n = CitesteIntreg();
    int k = CitesteIntreg();

    int combinari = Combinari(n, k);
}

1 reference
private static int Combinari(int n, int k)
{
    return Fact(n) / (Fact(k) * Fact(n - k));
}

3 references
private static int Fact(int n)
{
    int i;
    int result;
    for (i = 1, result = 1; i <= n; i++)
    {
        result *= i;
    }
    return result;
}
```

STACK
HEAD →

Fact{ int n }
Combinari{ int n, int k }
Main{ int n, int k, int combinari }

Call stack

```
static void Main(string[] args)
{
    int n = CitesteIntreg();
    int k = CitesteIntreg();

    int combinari = Combinari(n, k);
}

1 reference
private static int Combinari(int n, int k)
{
    return Fact(n) / (Fact(k) * Fact(n - k));
}

3 references
private static int Fact(int n)
{
    int i;
    int result;
    for (i = 1, result = 1; i <= n; i++)
    {
        result *= i;
    }
    return result;
}
```

STACK
HEAD →

Fact{ int n }
Combinari{ int n, int k }
Main{ int n, int k, int combinari }

Call stack

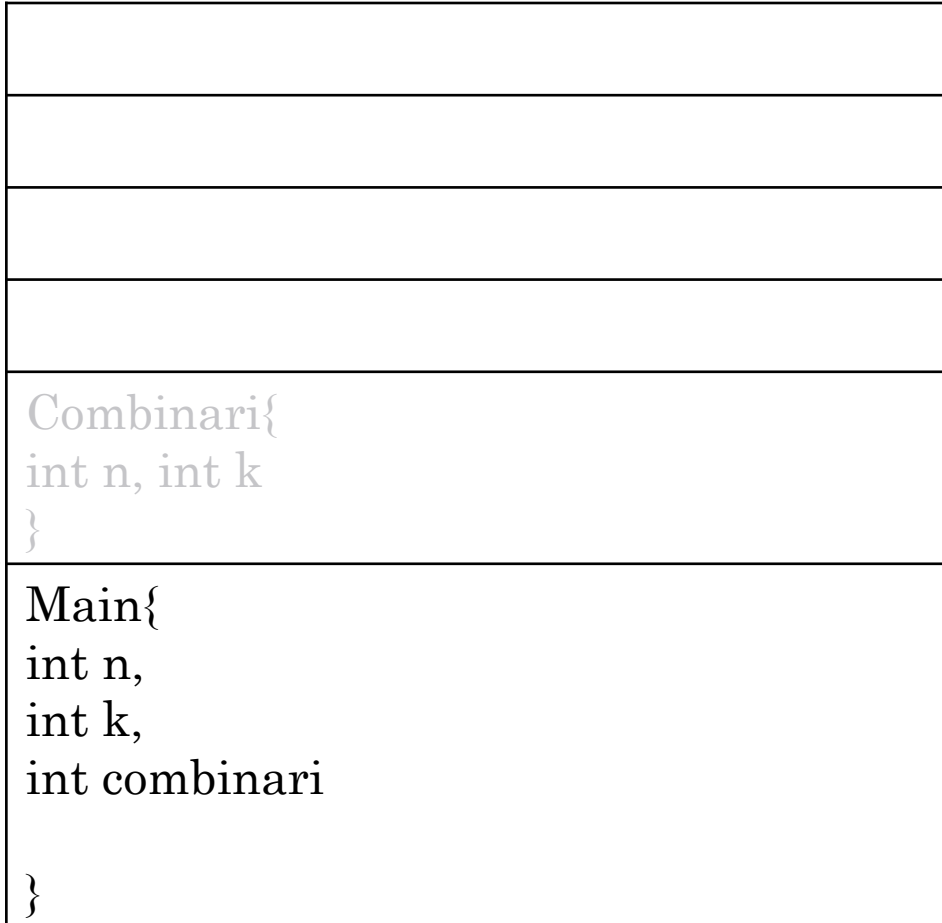
```
static void Main(string[] args)
{
    int n = CitesteIntreg();
    int k = CitesteIntreg();

    int combinari = Combinari(n, k);
}

1 reference
private static int Combinari(int n, int k)
{
    return Fact(n) / (Fact(k) * Fact(n - k));
}

3 references
private static int Fact(int n)
{
    int i;
    int result;
    for (i = 1, result = 1; i <= n; i++)
    {
        result *= i;
    }
    return result;
}
```

STACK
HEAD →



Call stack

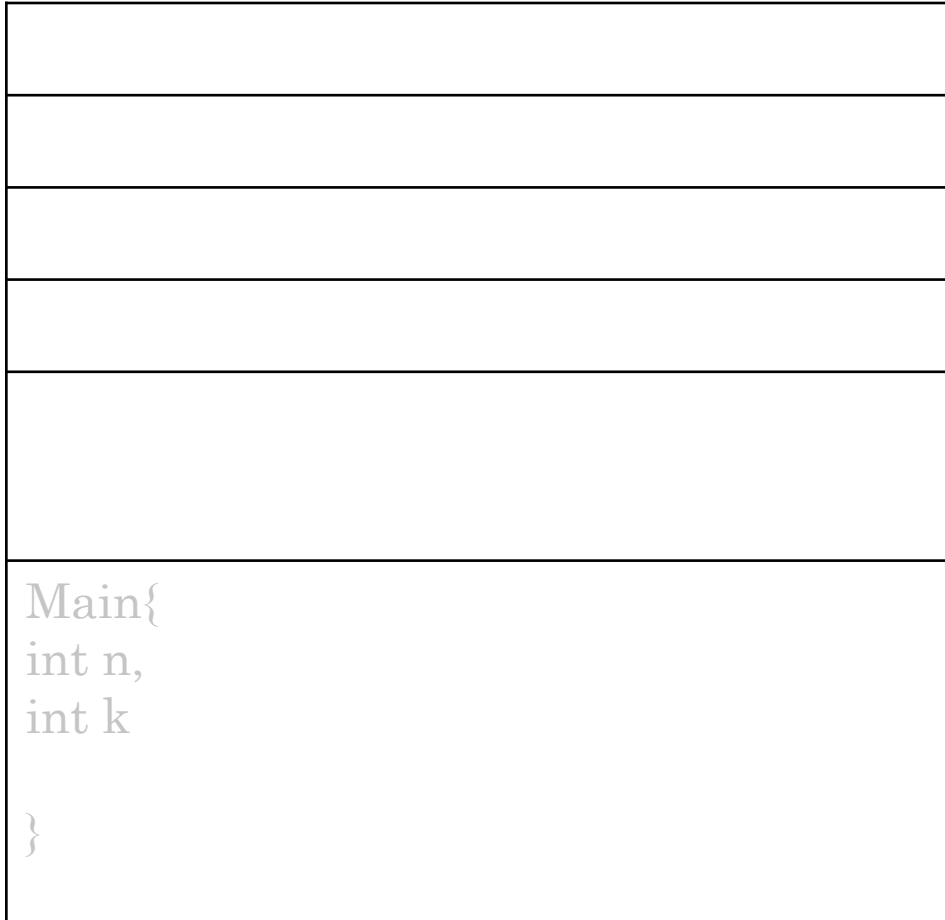
```
static void Main(string[] args)
{
    int n = CitesteIntreg();
    int k = CitesteIntreg();

    int combinari = Combinari(n, k);
}

1 reference
private static int Combinari(int n, int k)
{
    return Fact(n) / (Fact(k) * Fact(n - k));
}

3 references
private static int Fact(int n)
{
    int i;
    int result;
    for (i = 1, result = 1; i <= n; i++)
    {
        result *= i;
    }
    return result;
}
```

STACK
HEAD →



Call stack

```
static void Main(string[] args)
{
    int n = CitesteIntreg();
    int k = CitesteIntreg();

    int combinari = Combinari(n, k);
}

1 reference
private static int Combinari(int n, int k)
{
    return Fact(n) / (Fact(k) * Fact(n - k));
}

3 references
private static int Fact(int n)
{
    int i;
    int result;
    for (i = 1, result = 1; i <= n; i++)
    {
        result *= i;
    }
    return result;
}
```


Vectori (array)

Vectori

- Un vector: un tip de date capabil sa reprezinte un sir de valori.

- Exemple:

- Vector de intregi:

1	14	3	5	0	0
---	----	---	---	---	---

- Vector bool

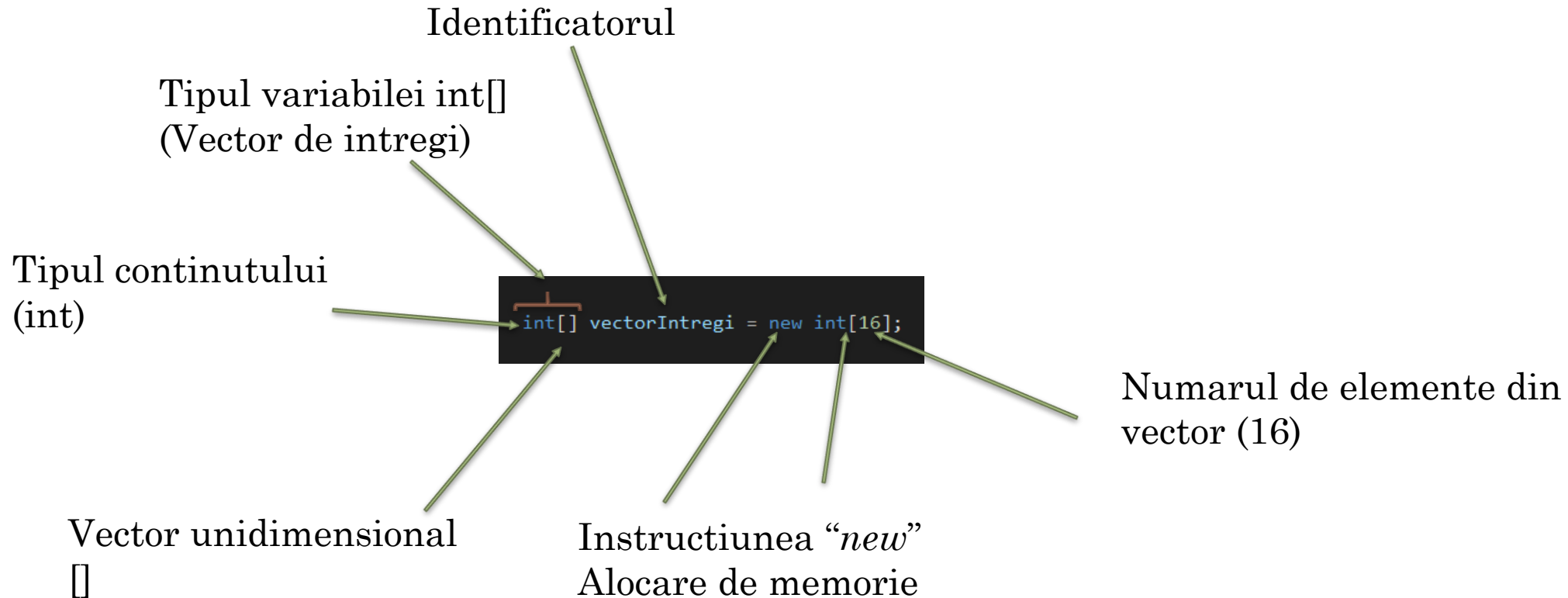
true	false	false	true	true	false
------	-------	-------	------	------	-------

- Vector double

1.5	5.333	9	14	0	53
-----	-------	---	----	---	----

- Elementele unui vector au același tip (nu putem amesteca elemente de tipuri diferite)
 - Valori default – false, 0, null... etc

Declararea si intializarea vectorilor



Declararea si intializarea vectorilor

- Declararea
 - Specificarea tipului si a identicatorului
- Initializarea unui vector: prin specificarea dimensiunii (lungimii) sale.
- Alocarea memoriei
 - La initializare

```
int[] vectorIntregi;
```

```
vectorIntregi = new int[16];
```

```
int[] vectorIntregi = new int[] { 1, 14, 3, 5, 0, 0 };
```


Valorile array-ului

• Int[]v	1	14	3	5	0	0
• Indeksi	0	1	2	3	4	5
• Valori	v[0]	v[1]	v[2]	v[3]	v[4]	v[5]

- **Indeksi încep de la 0 și merg până la vector.Length-1**
 - vector_name.Length = lungimea vectorului
- Fiecare element se găsește pe o anumită poziție (index). Indecșii se numerează începând de la 0
- Vectorii au dimensiune fixă (nu putem altera dinamic dimensiunea vectorului decât creând un vector nou)

Vectori

- Citire
- Schibmare valoare

```
// declarare, initializare  
int[] vectorIntregi = new int[16];  
  
// citire valoare din vector  
int pozitia6 = vectorIntregi[6];  
  
// atribuire/modificare valoare in vector  
vectorIntregi[6] = 55;
```

Heap

```
int intreg = 42;  
int[] vectorIntregi = new int[]{ 1, 2, 3 };
```

Stack
42
0xC0FFEE

Heap

1
2
3



Heap

```
int integ = 42;  
int[] vectorIntegi = new int[]{ 1, 2, 3 };  
vectorIntegi = new int[]{ 4,5,6,7};
```

Stack
42
0xC0FFEE

Heap

1
2
3



Heap

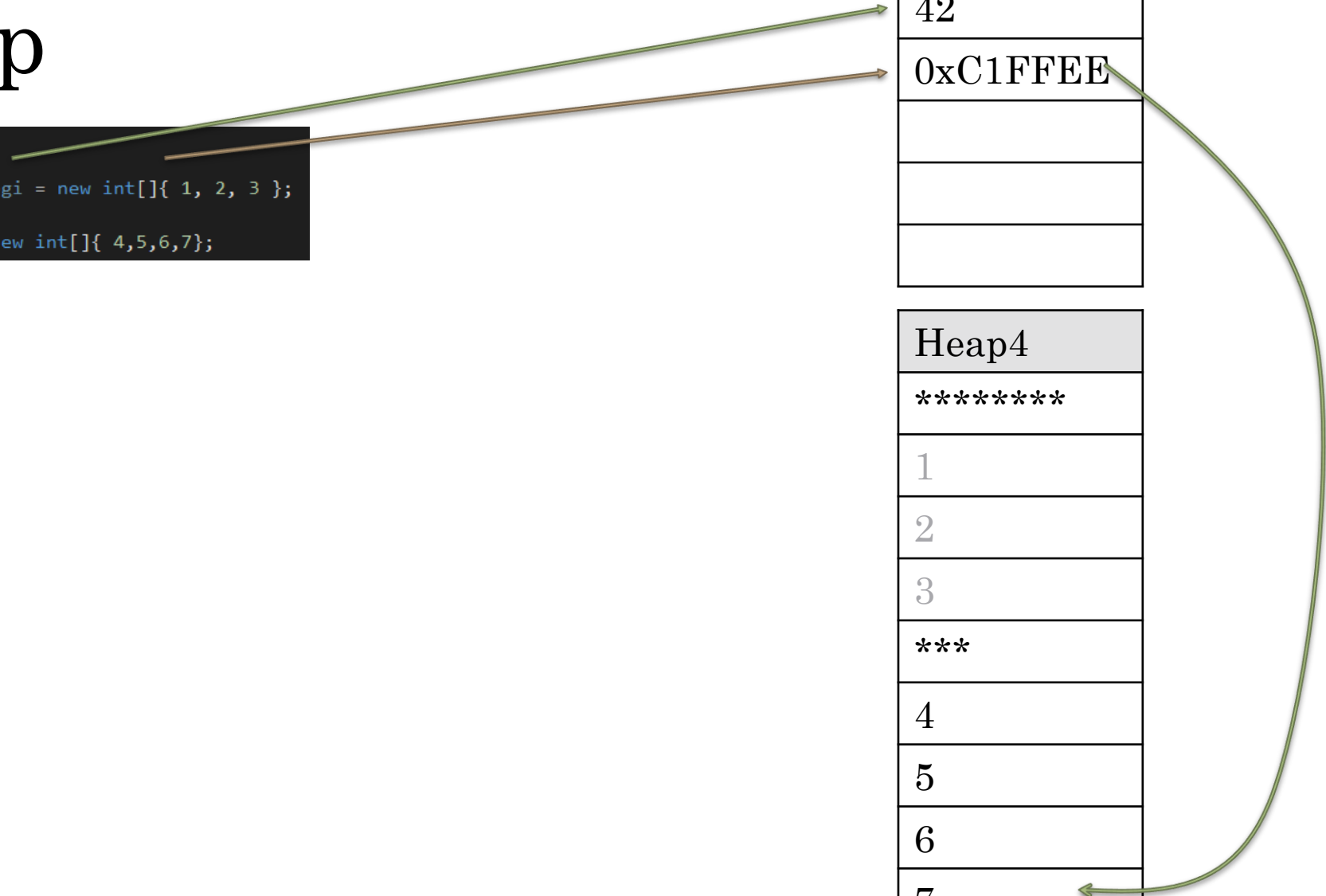
```
int integ = 42;  
int[] vectorIntegi = new int[]{ 1, 2, 3 };  
vectorIntegi = new int[]{ 4,5,6,7};
```

Stack
42
0xC1FFEE

Heap4

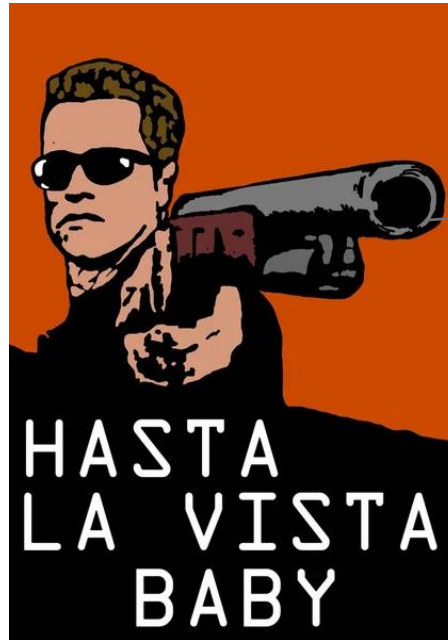
1
2
3

4
5
6
7



Heap

```
int integ = 42;  
int[] vectorIntregi = new int[]{ 1, 2, 3 };  
vectorIntregi = new int[]{ 4,5,6,7};
```



Garabage Collector

Stack
42
0xC1FFEE

Heap4

1
2
3

4
5
6
7

Heap

```
int intreg = 42;  
int[] vectorIntregi = new int[]{ 1, 2, 3 };  
vectorIntregi = new int[]{ 4,5,6,7};
```



Garabage Collector

Stack
42
0xC1FFEE
Heap4

4
5
6
7

Vectori – considerente

- Utilizari ale vectorilor:
 - Lucrul cu colecții de date de același tip
 - Calcule matematice
- Avantaje ale vectorilor:
 - Ocupă o zona continua de memorie – realocarea spatiului pt. structurile dinamice este costisitoare si fragmenteaza memoria
 - Adresare usoara/rapida dupa index – $O(1)$
 - Iterarea ușoară și performantă
- Dezavantaje ale vectorilor:
 - Nu se preteaza la utilizare atunci cand avem nevoie sa adaugam / eliminam dinamic elemente din colectie
 - Risipa de memorie in cazul colectiilor care contin multe elemente fara continut
 - Dimensiune fixa.

Vectors - exercises

- Write a program that will reverse the elements of a vector
 - The length of the vector will be read from the keyboard

var

var

- Declararea implicita a tipurilor
- In functie de valoarea initializarii
- Syntactic sugar
- Scope: block-level
- Tipul – cel mai precis

```
int → var intreg = 3;
float → var tipFloat = 3.0f;
double → var tipDouble = 3.0d;
decimal → var tipDecimal = 3.0m;
char → var caracter = 'a';
string → var sirDeCaractere = "a";
int[] → var vectorIntregi = new int[] { 1, 2, 3 };
Student → var student = new Student();
StudentLiceu → var studentLiceu = new StudentLiceu();

class Student
{
}
1 reference
class StudentLiceu : Student
{
}
```

Reprezentarea datelor

In heap/ stack

- Int
- Vector of int
- Vector of vector of int

Intreg

```
public static void Main()
{
    int intreg = 3;

    int[] vectoriIntregi = { 1, 2, 3 };

    int[][] jaggedArray = new int[][]{
        new int[] { 1, 3, 5, 7, 9 },
        new int[] { 0, 2, 4, 6 },
        new int[] { 11, 22 }
    };
}
```

[illegible][illegible]

Intreg

```
public static void Main()
{
    int intreg = 3;

    int[] vectorIntregi = { 1, 2, 3 };

    int[][] jaggedArray = new int[][]{
        new int[] { 1, 3, 5, 7, 9 },
        new int[] { 0, 2, 4, 6 },
        new int[] { 11, 22 }
    };
}
```

[illegible][illegible]

VectorIntregi

```
public static void Main()
{
    int intreg = 3;

    int[] vectorIntregi = { 1, 2, 3 };

    int[][] jaggedArray = new int[][]{
        new int[] { 1, 3, 5, 7, 9 },
        new int[] { 0, 2, 4, 6 },
        new int[] { 11, 22 }
    };
}
```

[illegible][illegible]

VectorIntregi

```
public static void Main()
{
    int intreg = 3;

    int[] vectorIntregi = { 1, 2, 3 };

    int[][] jaggedArray = new int[][]{
        new int[] { 1, 3, 5, 7, 9 },
        new int[] { 0, 2, 4, 6 },
        new int[] { 11, 22 }
    };
}
```

Stack
3
0x12CCEE

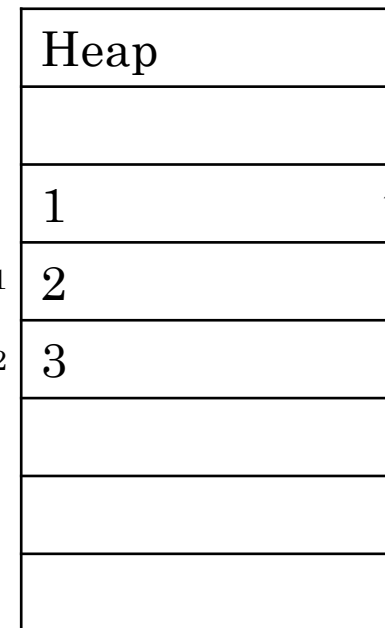
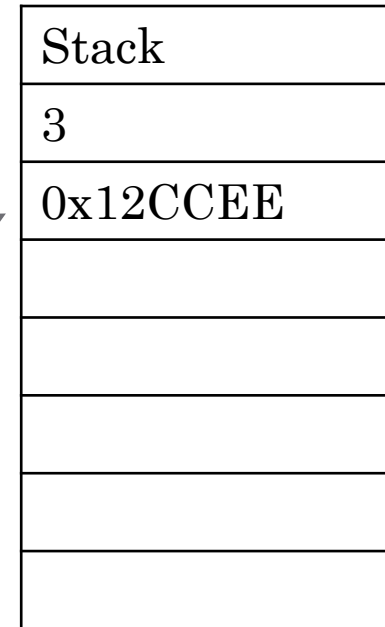
[illegible]

VectorIntregi

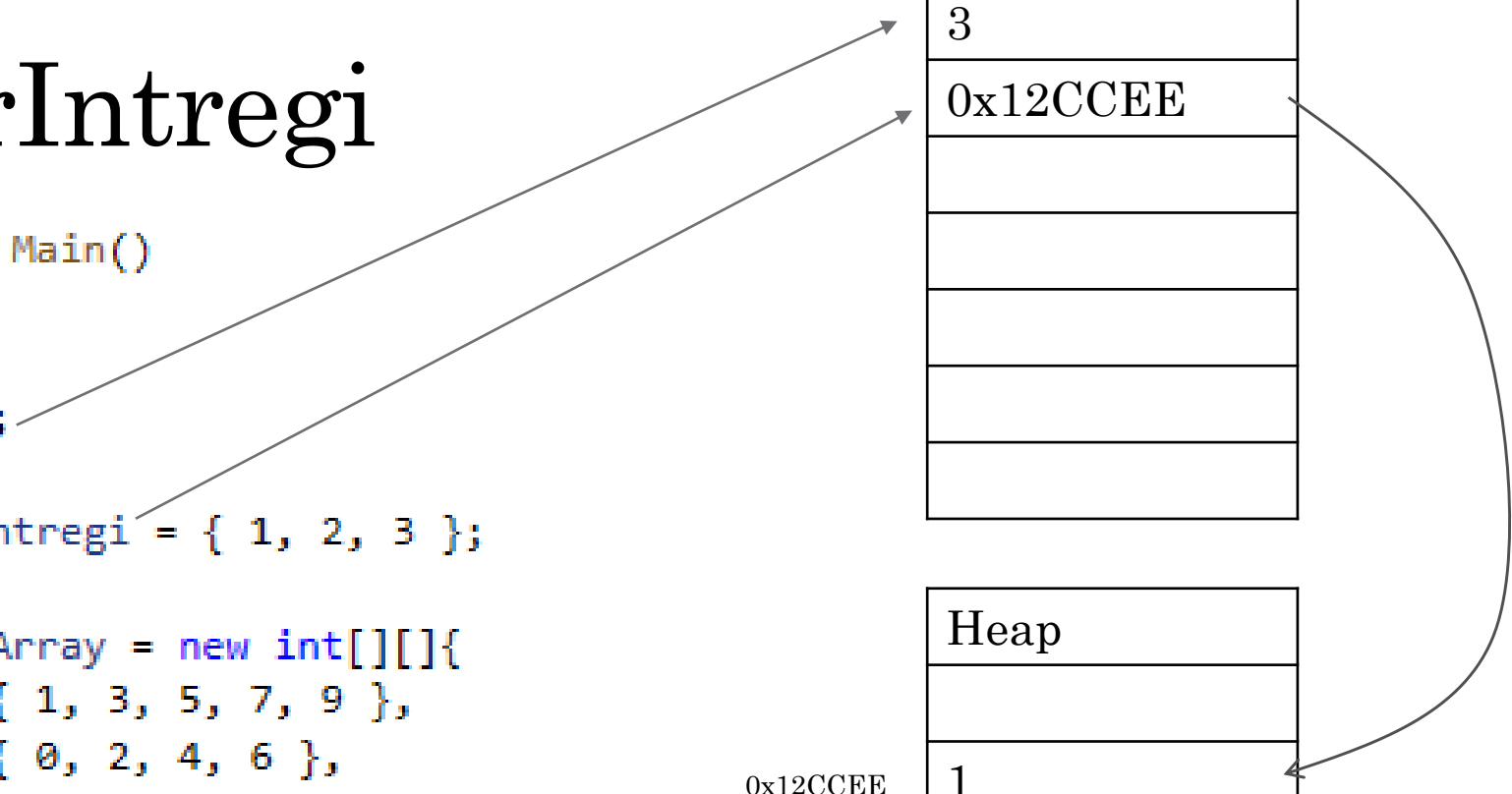
```
public static void Main()
{
    int intreg = 3;

    int[] vectoriIntregi = { 1, 2, 3 };

    int[][] jaggedArray = new int[][]{
        new int[] { 1, 3, 5, 7, 9 },
        new int[] { 0, 2, 4, 6 },
        new int[] { 11, 22 }
    };
}
```



0x12CCEE
0x12CCEE+1
0x12CCEE+2



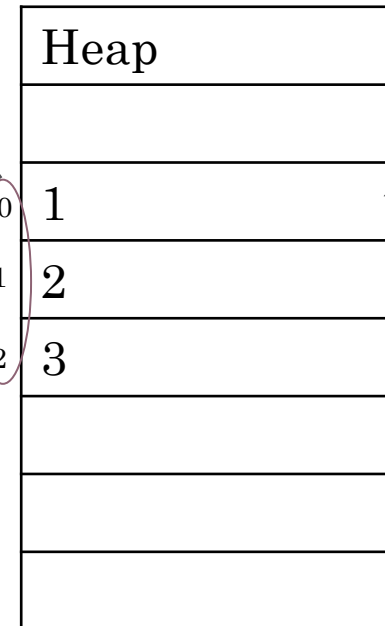
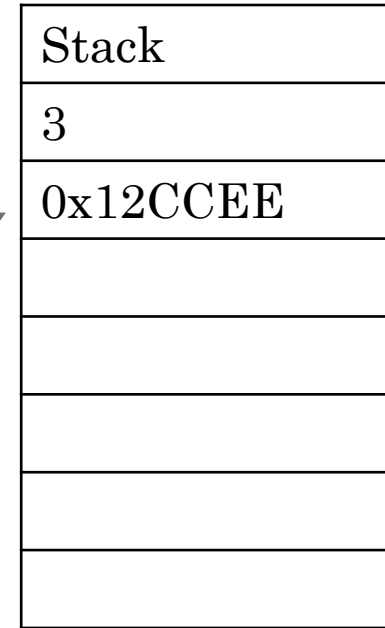
VectorIntregi

```
public static void Main()
{
    int intreg = 3;

    int[] vectorIntregi = { 1, 2, 3 };

    int[][] jaggedArray = new int[][]{
        new int[] { 1, 3, 5, 7, 9 },
        new int[] { 0, 2, 4, 6 },
        new int[] { 11, 22 }
    };
}
```

vectorIntregi[0]
vectorIntregi[1]
vectorIntregi[2]



0x12CCEE+0

0x12CCEE+1

0x12CCEE+2

1

2

3

Funcții recursive

Introducere

Funcții care se apelează pe ele însele

Functii recursive

- Conditia de iesire
 - La inceput
 - Cat mai cuprinzatoare
 - Evita bucla infinita, “stack overflow”
- Poate returna o valoare
- Ordinea apelului urmatoarei iteratii impacteaza masiv rezultatul

```
0 references
static void Main(string[] args)
{
    Console.WriteLine("introduceti n");
    int n = int.Parse(Console.ReadLine());

    AfisareNumere0laN(n);
}

2 references
static void AfisareNumere0laN( int n)
{
    if (n <= 0)
    {
        return;
    }
    AfisareNumere0laN(n - 1);
    Console.WriteLine(n);
}
```


Funcții recursive – recursivity

- Funcții care se cheama pe ele insele
- **Forma generala**
 - Conditie de iesire
 - Continutul functiei
- Conditia de iesire
 - **La inceput**
 - Cat mai cuprinzatoare
 - Evita bucla apelurilor infinite,
„**stack overflow**”
- Poate returna o valoare
- Ordinea apelului urmatoarei iteratii
impacteaza masiv rezultatul

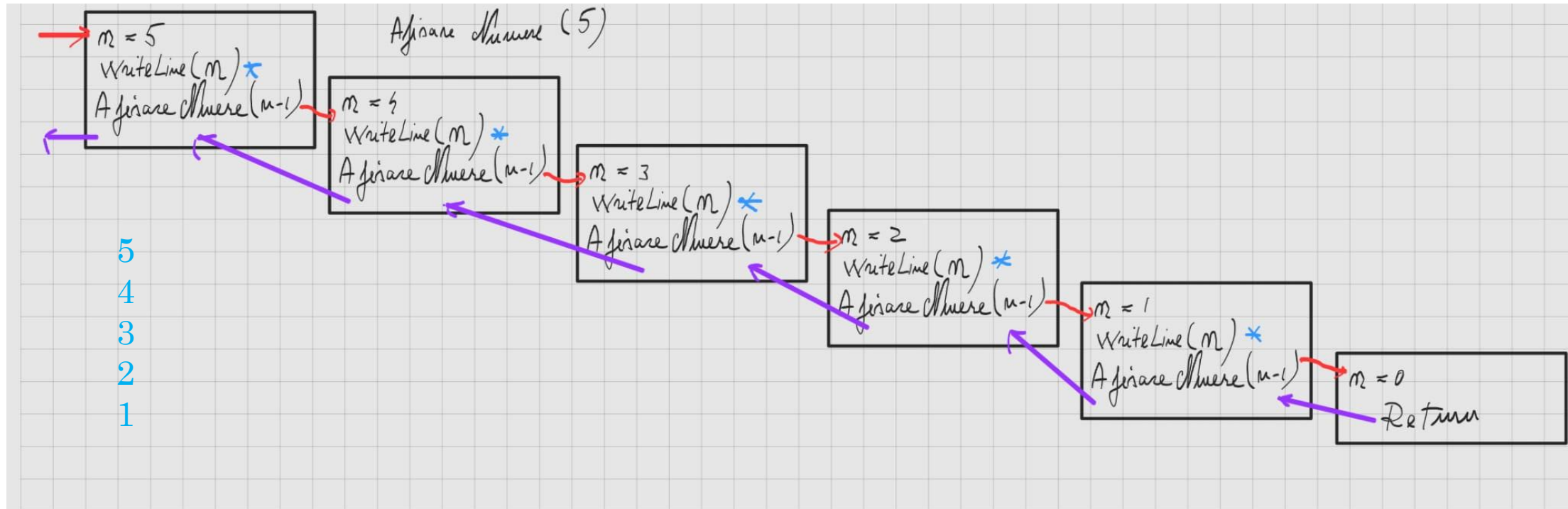
```
0 references
static void Main()
{
    AfisareNumere0N(5);
}

2 references
static void AfisareNumere0N(int n)
{
    if (n <= 0)
    {
        return;
    }

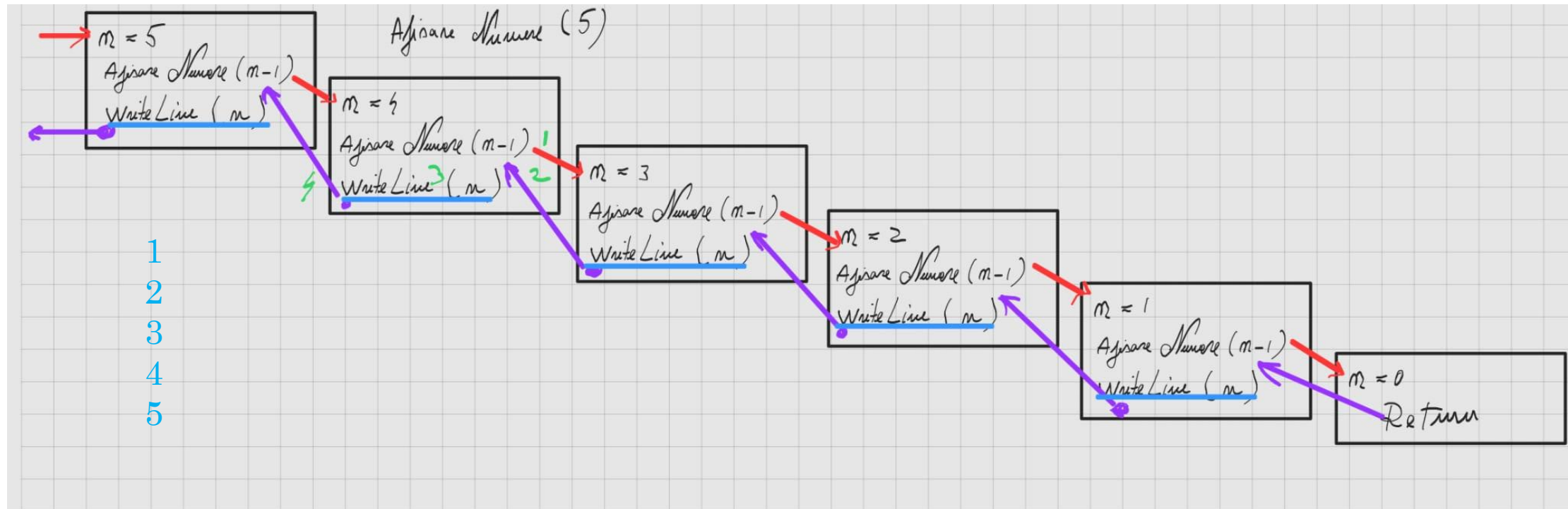
    Console.WriteLine(n);
    AfisareNumere0N(n - 1);
}
```



Funcții recursive – recursivity



Funcții recursive – recursivity



Funcții recursive – exercitii

- Scrieti o functie care va calcula factorialul unui numar

MATRIX

Multidimensional arrays



Vettori multidimensionali

- Vettori multidimensionari, matrici
 - Matrix

$$\begin{bmatrix} 10 & 4 & 2 \\ 12 & 6 & 9 \end{bmatrix}$$

	Column 0	Column 1	Column 2
Row 0	[0,0]	[0,1]	[0,2]
Row 1	[1,0]	[1,1]	[1,2]

Vettori multidimensionali

- Vettori multidimensionari
 - Matrix

$$\begin{bmatrix} 10 & 4 & 2 \\ 12 & 6 & 9 \end{bmatrix}$$

```
// this will be a matrix of 2 rows X 3 cols
int[,] matrix = new int[2, 3];

// first row
matrix[0, 0] = 10;
matrix[0, 1] = 4;
matrix[0, 2] = 2;
// second row
matrix[1, 0] = 12;
matrix[1, 1] = 6;
matrix[1, 2] = 9;
```

Vettori multidimensionali

```
// alternatively
matrix = new int[2, 3]
{
    {10, 4, 2},
    {12, 6, 9}
};

// or even
int[,] matrixSimple =
{
    {10, 4, 2},
    {12, 6, 9}
};
```

Vectori multidimensionali

```
int[, ,] matrix3D = {  
    { { 1,2,3 }, { 4,5,6 } },  
    { { 7,8,9 }, { 10,11,12 } },  
    { { 13,14,15 }, { 16,17,18 } }  
};
```

```
var pos = matrix3D[0, 1, 2];  
Console.WriteLine(pos);
```

```
int[,,,] matrix4D = new int[4, 2, 3, 4];
```

- `matrix.Length` – lungimea TOTALA a matricii
- `matrix.Rank` – numarul de dimensiuni a matricii
- `matrix.GetLength(rankNo)` – numarul de elemente dintr-o anumita dimensiune a matricii

Exercitiu

- Cititi de la tastatura continutul unei matrici de intregi cu 2 dimensiuni avand lungimile m , n . Lungimile celor doua dimensiuni ale matricii, m si n , vor fi citite de la tastaura.
 - Scrieti o functie care va afisa continutul unei astfel de matrici si apelati-o.
 - Scrieti o functie care va aduna doua matrici, apelati-o si afisati rezultatul.

ref

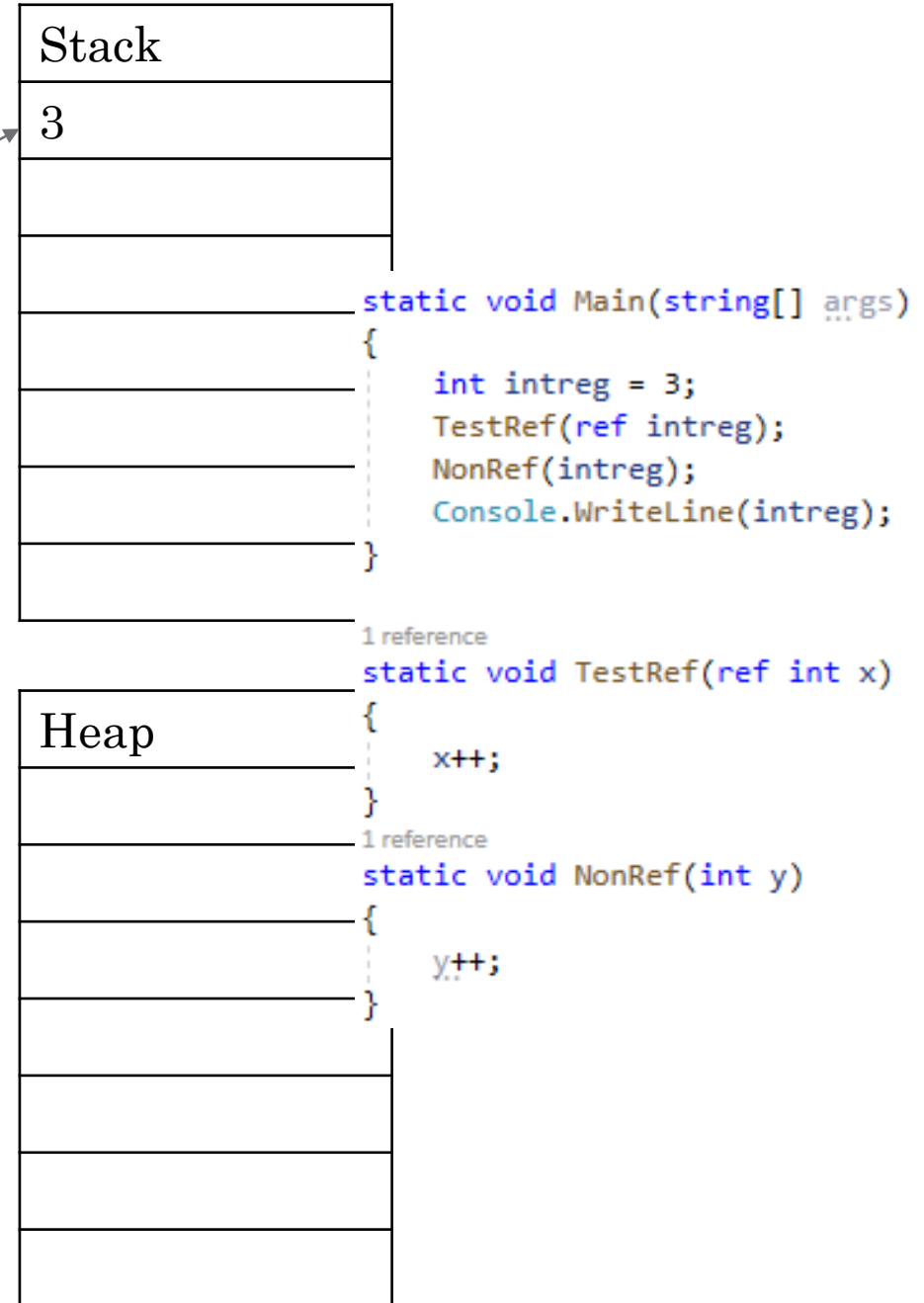
Asa nu

Intreg

```
public static void Main()
{
    int intreg = 3;

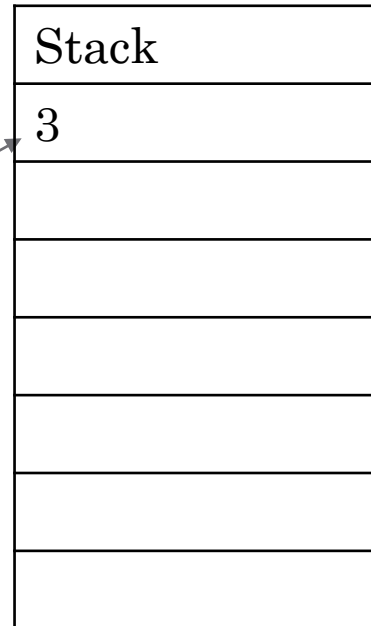
    int[] vectoriIntregi = { 1, 2, 3 };

    int[][] jaggedArray = new int[][]{
        new int[] { 1, 3, 5, 7, 9 },
        new int[] { 0, 2, 4, 6 },
        new int[] { 11, 22 }
    };
}
```



Intreg

```
public static void Main()  
{  
    int intreg = 3;  
  
    int[] vectorIntregi = { 1, 2, 3 };  
  
    int[][] jaggedArray = new int[][]{  
        new int[] { 1, 3, 5, 7, 9 },  
        new int[] { 0, 2, 4, 6 },  
        new int[] { 11, 22 }  
    };  
}
```



0xC0FFEE

```
static void Main(string[] args)  
{  
    int intreg = 3;  
    TestRef(ref intreg);  
    NonRef(intreg);  
    Console.WriteLine(intreg);  
}
```

1 reference

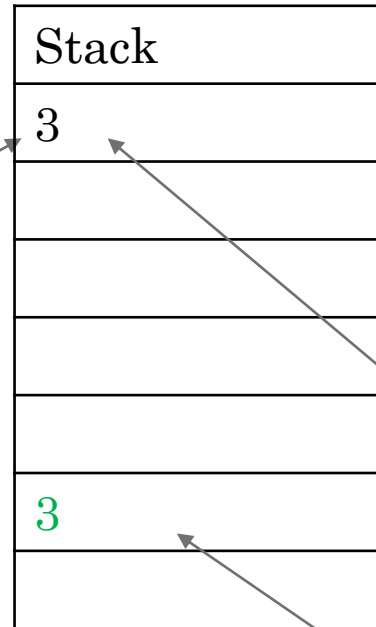
```
static void TestRef(ref int x)  
{  
    x++;  
}
```

1 reference

```
static void NonRef(int y)  
{  
    y++;  
}
```

Intreg

```
public static void Main()  
{  
    int intreg = 3;  
  
    int[] vectorIntregi = { 1, 2, 3 };  
  
    int[][] jaggedArray = new int[][]{  
        new int[] { 1, 3, 5, 7, 9 },  
        new int[] { 0, 2, 4, 6 },  
        new int[] { 11, 22 }  
    };  
}
```



0xC0FFEE

```
static void Main(string[] args)  
{  
    int intreg = 3;  
    TestRef(ref intreg);  
    NonRef(intreg);  
    Console.WriteLine(intreg);  
}
```

1 reference

```
static void TestRef(ref int x)  
{  
    x++;  
}
```

1 reference

```
static void NonRef(int y)  
{  
    y++;  
}
```

Tipul de date char

Char

- [Char – Microsoft link](#)
- Tipul de date char (caractere) – lucrul cu caractere (litere / simboluri)
- [Encoding: Unicode / UTF-16.](#)
 - ASCII (7bits)
- O valoare de tip char se reprezinta pe 2 bytes (16 biti).
- Tipul de date char este un alias pentru tipul “System.Char”.
- Tipul de date char este value type
- Intreg - Unsigned short, ushort
- Valorile – ghilimele simple -> 'a'

Char

- Initializare
 - Prin precizarea directa a valorii (literali)
 - Prin precizarea codului Unicode in format `'\uABCD'`, unde A, B, C, D sunt valorile in hexa ale codului Unicode
 - Prin precizarea valorii in hexa a codului Unicode, folosind un format de tipul `'\xABCD'`. Vezi si aceasta intrebare pe [StackOverflow](#)
 - Folosind o conversie catre char dintr-un tip numeric (valoarea numerica reprezentand codul Unicode)

Char - initialize

```
class Program
{
    0 references
    static void Main(string[] args)
    {
        char letterA = 'A';
        Console.WriteLine("The letter is:" + letterA);

        letterA = '\u0041';
        Console.WriteLine("The letter is:" + letterA);

        letterA = '\x0041';
        Console.WriteLine("The letter is:" + letterA);
    }
}
```

Char

- Tip numeric
- Tipul char este convertibil implicit catre urmatoarele tipuri numerice:
 - ushort, int, uint, long si ulong
 - float, double, decimal
- Conversia din tipuri numerice inspre char
 - Explicita (cast)

Char – caractere speciale

Apostrof	'	<code>char chr = '\'';</code>
Ghilimele	"	<code>char chr = '\"';</code>
Backslash	\	<code>char chr = '\\';</code>
Null (see wikipedia)		<code>char chr = '\0';</code>
Alert / Bell (see wikipedia)		<code>char chr = '\a';</code>
Backspace		<code>char chr = '\b';</code>
Form Feed		<code>char chr = '\f';</code>
New Line		<code>char chr = '\n';</code>
Carriage Return		<code>char chr = '\r';</code>
Horizontal tab		<code>char chr = '\t';</code>
Vertical tab		<code>char chr = '\v';</code>

Char - operatii

<code>char.IsDigit(char c)</code>	Test daca parametrul "c" este cifra
<code>char.IsLetter(char c)</code>	Test daca parametrul "c" este litera
<code>char.IsLetterOrDigit(char c)</code>	Test daca parametrul "c" este cifra sau litera
<code>char.IsWhiteSpace(char c)</code>	Test daca parametrul "c" este spatiu (sau tab/new-line, etc)
<code>char.IsPunctuation(char c)</code>	Test daca parametrul "c" este semn de punctuatie
etc ...	

```
public static void Main()
{
    for (var i = char.MinValue; i <= char.MaxValue; i++){
        Console.Write(i);
    }
}
```

Char - operatii

```
class Program
{
    0 references
    static void Main(string[] args)
    {
        char letter1 = 'A';
        char letter2 = 'B';

        // some non-sense operations
        // just to illustrate arithmetic operators
        int sumResult = letter1 + letter2;
        int multiplyResult = letter1 * letter2;
        int divResult = letter2 / letter1;
        int remainderResult = letter2 % letter1;
        // and some bitwise operators
        int shiftedResult = letter1 >> 2;
        int negatedResult = ~letter2;

        letter1++;
        // Outputs B
        Console.WriteLine("The letter is:" + letter1);

        // Outputs again A
        letter1--;
        Console.WriteLine("The letter is:" + letter1);
    }
}
```

- >, <, !=, ==, <=, >=, etc

Siruri de caractere

Tipul string

String

- Tipul de date string este un tip de date utilizat pentru lucrul cu siruri de
- caractere (litere / simboluri) Unicode / UTF-16.
- Tipul de date string este un alias pentru tipul “System.String”.
- Tipul de date string este un reference type (ce inseamna aceasta?)
- Se comporta ca un array – exemplu
 - Nu este un array, caracterele string-ului nu pot fi modificate

Initializare

```
class Program
{
    0 references
    static void Main(string[] args)
    {
        // caracterul backslash (\) folosit pentru
        // specificarea path-ului necesita escapare
        string path = "C:\\Windows\\System32\\drivers";

        Console.WriteLine(path);
    }
}
```

- Reference type – valoare default *null*
- Initializare string gol – *string.Empty*
- Pentru a evita aceasta problema (iritanta in special in cazul path-urilor), C# pune la dispozitie posibilitatea de a prefixa un string cu caracterul "@" (un astfel de string se numeste string verbatim)

String – imutabilitatea

- String-urile sunt imutabile, adica, odata initializate, ele nu mai pot fi modificate.
- Toate operatiunile efectuate asupra lor care “par” sa modifice un string, de fapt fac urmatoarele operatiuni:
 - Initializeaza un nou obiect string, avand ca valoare rezultatul operatiunii
 - Returneaza adresa noului obiect

Elementele string-ului

- Se comporta precum un vector de caractere
 - Caracterele accesibile prin intermediul indecsilor
- `string.Length`
 - Lungimea sirului de caractere
- Indecsii
 - De la *0* la *string.Length-1*

Exercitiu

- Scrieti o functie care va calcula numarul de caractere goale dintr-un string

String – verbatim

```
class Program
{
    0 references
    static void Main(string[] args)
    {
        // mult mai lizibil
        string path = @"C:\Windows\System32\drivers";

        Console.WriteLine(path);
    }
}
```

```
class Program
{
    0 references
    static void Main(string[] args)
    {
        string msg = @" acest gen de initializare
nu ar functiona
pentru un string non-verbatim";

        Console.WriteLine(msg);
    }
}
```

String

operatii

String – operatii

- Valoare default a string-ului si a orcarui reference type
 - null
 - Semnifica o adresa nespecificata
 - Produce erori in multe situatii
 - Conversie
 - Apeluri de metode ale obiectelor
 - Apeluri de functii
 - exemple
- Null/string empty test
 - `string.Empty` – valoare string gol, echivalenta “”, folosita la initializare.
 - Aceste valori speciale s-ar putea ca in majoritatea situatiilor sa necesite validare / tratare deosebita:
 - String-urile empty / whitespaces only probabil nu au un continut valid, lucru care poate duce la un comportament ciudat al aplicatiei (ex: probleme de afisare a informatiilor), sau chiar la erori (ex: conversii din string, sau alte tipuri de logica care se bazeaza pe faptul ca string-ul are un continut valid)
 - [Whitespace character - Wikipedia](#)

String – operatii

	null	Empty string	Whitespaces only
<code>== null</code>	True	False	false
<code>string.IsNullOrEmpty()</code>	True	True	True
<code>string.IsNullOrWhiteSpace()</code>	True	True	True

String – operatii

- Eliminarea whitespace-urilor de la inceputul, sau sfarsitul unui sir de caractere.
 - TrimStart: elimina caracterele whitespace de la inceputul unui string
 - TrimEnd: elimina caracterele whitespace de la finalul unui string
 - Trim: o combinatie a TrimStart si TrimEnd, eliminand whitespace-uri atat de la inceputul cat si de la sfarsitul unui string

Operatii cu string-uri

- Concatenare

```
string ana = "ana";  
string areMere = " are mere";  
  
string anaAreMere = ana + areMere;
```

- [string.Concat](#)
 - Click this!
- [string.Join](#)
 - Click this!

String – ToLower, ToUpper

- ToLower: transforma literele majuscule in litere mici, tinand cont de regulile specifice culturii curente a utilizatorului, sau o anumita cultura transmisa ca si parametru.
- ToLowerInvariant: transforma literele majuscule in litere mici, facand apel la regulile unei culturi standard (English, United States)
- ToUpper
- ToUpperInvariant

ToLower / ToUpper

```
class Program
{
    0 references
    static void Main(string[] args)
    {
        string message = "Mixed Lowercase aNd Uppercase letTTers";

        string lowercaseAll = message.ToLower();
        string uppercaseAll = message.ToUpper();

        Console.WriteLine(lowercaseAll);
        Console.WriteLine(uppercaseAll);
    }
}
```


String – interpolare

- Folosind concatenare

```
int numarulMerelor= 5;  
string anaAreMere = "ana are " + numarulMerelor + "mere";
```

- Asa nu!

- interpolare a string-urilor:
 - “decorarea” string-urilor cu valorile unor variabile
 - un sir de caractere care utilizeaza acest feature este prefixat cu semnul “\$” (analog cu modul in care string-urile verbatim sunt prefixate cu “@”)

```
int numarulMerelor= 5;  
string anaAreMere = $"ana are {numarulMerelor} mere";
```

String – interpolare

```
class Program
{
    0 references
    static void Main(string[] args)
    {
        int quantity = 10;
        double unitPrice = 20.5;
        string productName = "Product A";
        string unitOfMeasure = "buc";

        string message = $"Ati cumparat {quantity} {unitOfMeasure} din produsul \"{productName}\", pretul unitar fiind {unitPrice}";
        Console.WriteLine(message);
    }
}
```

String – fragmente

- Cautarea unor fragmente de text in cadrul string-urilor
 - `string.Contains`: testeaza daca un string contine textul specificat ca si parametru
 - `string.IndexOf`: intoarce primul index in cadrul string-ului unde se regaseste (sau unde incepe) textul specificat ca si parametru
 - `string.LastIndexOf`: intoarce ultimul index in cadrul string-ului unde se regaseste (sau unde incepe) textul specificat ca si parametru
 - `string.StartsWith`: testeaza daca un string incepe cu textul specificat ca si parametru
 - `string.EndsWith`: testeaza daca un string se termina cu textul specificat ca si parametru
- **Nu uitati**: toate aceste metode au nevoie sa faca verificari de egalitate pe string-uri, prin urmare nu uitati de parametrul `StringComparison`

String – substring

- extrage un fragment de text (sub-string) din cadrul unui string mai mare.
 - Functia [SubString](#).
- 2 moduri:
 - indexul de la care se incepe extragerea (copierea) si de la acea pozitie veti continua pana la finalul string-ului
 - indexul de la care incepeti extragerea si numarul de caractere (lungimea) pe care doriti sa le copiati

```
class Program
```

```
{
```

0 references

```
static void Main(string[] args)
```

```
{
```

```
    string message = "Hello world!";
```

```
    string part1 = message.Substring(6);
```

```
    Console.WriteLine("Part1=" + part1);
```

```
    string part2 = message.Substring(0, 5);
```

```
    Console.WriteLine("Part2=" + part2);
```

```
}
```

```
}
```

String – inlocuire valori

- inlocuirea un fragment de text (sub-string) din cadrul unui string, cu un alt fragment
 - `string.Replace`: permite specificarea string-ului care trebuie inlocuit si a string-ului cu care se inlocuieste
- eliminarea unui fragment de string
 - `string.Remove`: permite specificarea indexului de la care se incepe eliminarea continutului si eventual, numarul de pozitii eliminate
- **Nu uitati, String-ul este imutabil**

String – inlocuire valori

```
class Program
{
    0 references
    static void Main(string[] args)
    {
        var message = "Hello world, beautiful world!";

        var replacedMessage = message.Replace("world", "Test");

        Console.WriteLine(message);
        Console.WriteLine(replacedMessage);

        var removedMessage = message.Remove(11, 17);
        Console.WriteLine(message);
        Console.WriteLine(removedMessage);
    }
}
```

String – split

- Functia [Split](#)
 - “desparte” sirul de caractere bazandu-se pe un caracter
 - Returneaza o lista de siruri de caractere
 - Exemplu

```
var input = "ana are mere are ";
string[] rezultat = input.Split(' ');

foreach(string fragment in rezultat)
{
    Console.WriteLine(fragment);
}

/*
 * va tipari:
ana
are
mere
are

 */
```


Va multumesc!