C# .NET

Laborator 2

Recapitulare, Rezolvare exercitii

Instructiuni repetitive

while

Instructiunea while

```
while (condition)
{// do
    // Instructions
    // reevaluate condition
}
```

- ATAT TIMP cat CONDITIA este adevarata executa INSTRUCTIUNI
- Conditia – bool
- Intotdeauna conditia booleana de iesire trebuie modificata in interiorul buclei while
  - Caz contrar: bucla infinita
- Instructiunea break
  - Iasa automat din interiorul buclei
- Instructiunea continue
  - “sare peste” instructiunile urmatoare ruland urmatoarea iteratie a buclei

Instructiunea while - exercitii

- Ex 1: Scrieti un program care va afisa numerele de la 1 pana la *10*
- Ex 2: Scrieti un program care va afisa suma cifrelor unui numar *n*, n fiind citit de la tastatura.

Instructiuni repetitive

for

Instructiunea for

```
int index = 1;
while (index <= 10)
{
    Console.WriteLine(index);
    index = index + 1;
}
```

```
for (int index=1; index <= 10; index = index + 1){
    Console.WriteLine(index);
}
```

```
for (var i = 0; i < 11; i++)
{
    Console.WriteLine(i);
}
```

```
for (initializer; condition; iterator)
{
        //code block
}
```

- Initializer – orice instructiune
- Conditie – orice operatie care are ca rezultat o valoare bool
- Iterator – orice instructiune

```
for (var i = 1; i != n; i++)
{
    Console.WriteLine(i);
}
```

Instructiunea for - exercitii

- Ex 3: Scrieti un program care va afisa numerele de la 1 pana la *10*

Instructiunea for – capcane

```
var n = 10;

for (var i = 1; i != n; i++)
{
     n = n - i;
     Console.WriteLine(i);
}
```

- Care va fi rezultatul rularii pt n = 10

Instructiunea for – capcane

```
var n = 10;

for (var i = 1; i != n; i++)
{
    n = n - i;
    Console.WriteLine(i);
}
```

- N = conditia matematica de iesire
- Evitati o conditie prea specifica
  - Folositi < , <= in locul lui == sau !=
- Evitati modificarea conditiei matematice in interiorul if-ului
  - Spre deosebire de while, aici iteratorul este modificat!

Instructiunea for - exercitii

- Ex 4: Scrieti un program care va afisa piramida numerelor

1

2 2

3 3 3

4 4 4 4

…

n n n n n ….. n

Instructiuni repetitive - terminologie

- Iteratie
  - o executie a block-ului de cod din interiorul buclei
- Iterator
  - Un numar intreg identificator al iteratiei
- Conditia booleana de intrare
  - conditia booleana care va determina intrarea in bucla de executie
- Complexitate ciclomatica O(n), big O
  - Numarul de bucle imbricate
  - O(n), O(2n), O(3n) ….. O(n)4
  - O(n^2), O($2n^2$), O($3n^2$)….. O(n^2)
  - O(n^3), O(n^4)… etc
  - Ideal : O(1), O(log_2n)…

Instructiuni repetitive

do…while

Instructiunea do…while

```
do
{
    // instructions
}
while (condition);
```

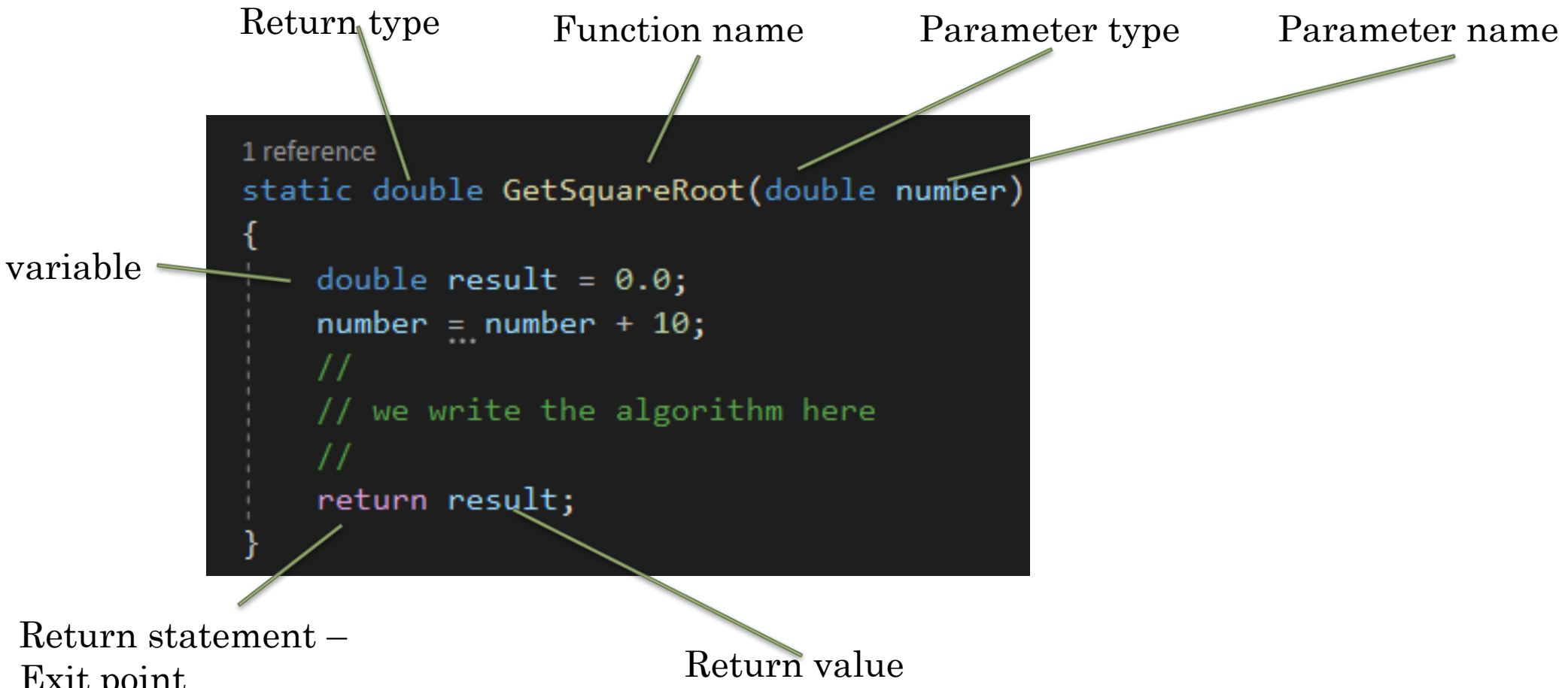
- executa INSTRUCTIUNI ATATA TIMP cat CONDITIA este adevarata
- Conditia
  - Bool
  - Evaluata la finalul fiecarei iteratii
- Intotdeauna conditia booleana de iesire trebuie modificata in interiorul buclei
- Instructiunea break
- Instructiunea continue

Functii

Functii

- Exercitiu:
- Scrieti un program care citind de la tastatura un numar intreg n, si un numar intreg k, va afisa $A_n^k = \frac{n!}{(n-k)!}$ si $C_n^k = \frac{n!}{k!*(n-k)!}$.

Functii


Return type
Function name
Parameter type
Parameter name
variable
1 reference
static double GetSquareRoot(double number)
{
double result = 0.0;
number = number + 10;
//
// we write the algorithm here
//
return result;
}
Return statement –
Exit point
Return value

Functii

- Numele
  - PascalCase
- Parametrii
  - Transmisi prin valoare **(exteriorul nu se modifica)**
  - Transmisi prin referinta (este trimisa adresa de memorie a)
    - Majoritatea tipurilor reference-type, vectori
    - *"out"* – antipattern, asa nu!
  - Vizibili in interiorul functiei
  - De la 0 la 65536 parametri
    - Max 3-7 params
    - Despartiti prin virgula
- Rezultatul functiei
  - Orice tip de date C#, functii
  - Tipul void
- Return
  - Marcheaza iesirea din functie
  - Daca functia are return type trebuie sa returneze o valoare pe toate beranch-urile sale

Functii

- Ajuta la managementul complexitatii
  - Impart un algoritm complex intr-un set de algoritmi mai simpli
  - PornesteMasina => DeschideUsa, InchideUsa, PuneCentura, ApasaButonulDePornire
- Fac codul reutilizabil
  - OpresteMasina => DeschideUsa, UnchideUsa

Functii - Solutie

Scrieti un program care citind de la tastatura un numar intreg n, si un numar intreg k, va afisa $A_n^k = \frac{n!}{(n-k)!}$ si $C_n^k = \frac{n!}{k!*(n-k)!}$.

```
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;
}
```

Functii -exemple

```
0 references
static void PrintFancyNumbers(int number1, int number2)
{
    Console.WriteLine("Fancy " + number1 + "\nFancy " + number2);
}
```

- 2 parametri, intregi
- Return type – void
- Invocare:

```
PrintFancyNumbers(1, 2);
```

Functii – return

```
0 references
static void SayHello() {
    Console.WriteLine("Hello");
}
```

- No params, no return type

```
// multiple exit points
0 references
static void PrintNumberSign(int number) {
    if (number > 0)
    {
        Console.WriteLine("positive");
        return;
    }
    if (number<0)
    {
        Console.WriteLine("negative");
        return;
    }
    Console.WriteLine("zerrro");
}
```

- Multiple returns, no return value

```
// multiple exit points with values
0 references
static int GetNumberSign(int number)
{
    if (number > 0)
    {
        return 1;
    }
    if (number < 0)
    {
        return -1;
    }
    return 0;
}
```

- Multiple exit points
- Invocare:

```
int sign = GetNumberSign(-20);
```

Functii interne – internal functions

- Functii definite in interiorul functiilor
  - Scope limitat
  - Variabilele din functia-parinte sunt vizibile in interiorul functiei interne
  - Antipattern

```
0 references
public static int Aranjamente(int n, int k) {

    int Factorial(int x)
    {
        int fact = 1;
        for(var i = 1; i <= x; i++)
        {
            fact *= i;
        }
        return fact;
    }

    return Factorial(n) / Factorial(n-k);
}
```

Functii – best practices

- Naming
  - PascalCase
  - Numele trebuie sa reflecte o actiune si sa contina un VERB
    - AfiseazaIntregi, TiparesteVector, CitesteUnSirDeLaTastatura
    - WriteLine, GetLength, ReadLine,

Single responsibility principle

- Single responsibility principle
- The single-responsibility principle (SRP) is a computer-programming principle that states that every module, class or **function** in a computer program should have **responsibility over a single part** of that program's functionality, and it should **encapsulate** that part.
  - **responsibility over a single part** – o functie trebuie sa faca un singur lucru
    - Ex: CalculSumaIntregi – doar calcul nu si afisare
    - Reutilizarea codului
  - **Encapsulate -** executia unei functii trebuie sa depinda doar de parametrii acesteia si nu de alte date “oculte”
    - Ex functii interne
    - Contraexemple – constante
    - Code duplication

Functii predefinite – clasa Math

- O colecie de funcii pentru calcul numeric
- Suporta
  - Numere intregi
  - **Virgula flotanta**
- Min, Max,
- Sqrt – square root
- Pow, Exp, Sin, Cos, Asin, Acos
- Abs
  - Exercitiu: scrieti un programa care compara egalitatea a doua numere double
- Round – rotunjire
- Ceiling – rotunjire la valoare superioara
- Floor – rotunjire la valoare inferioara

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

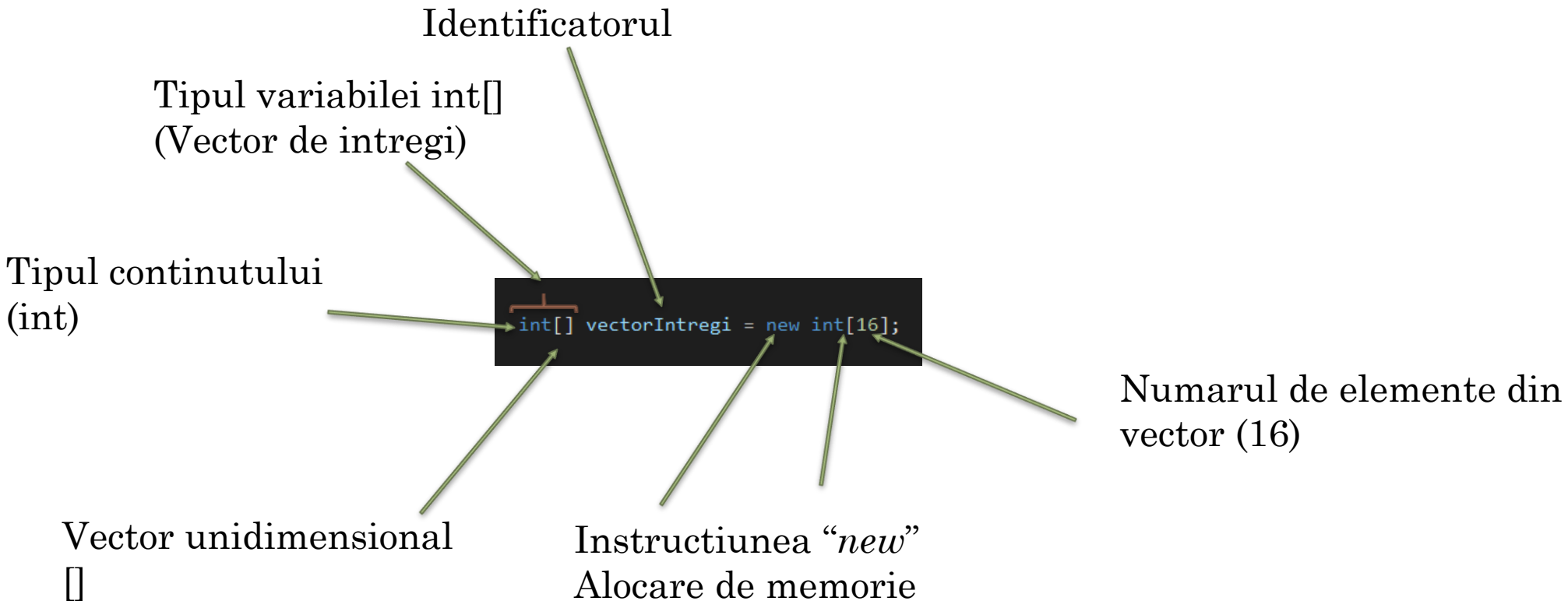
true	false	false	true	true	false

  - Vector double

1.5	5.333	9	14	0	53

  - Elementele unui vector au acelaşi 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 identificatorului
- 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

- Indecsi

0	1	2	3	4	5

- Valori

v[0]	v[1]	v[2]	v[3]	v[4]	v[5]

- **Indecsii incep de la 0 si merg pana la vector.Length-1**
  - vector_name.Length = lungimea vectorului
- Fiecare element se găsește pe o anumită poziție (index). Indecșii se numerotează începând de la 0
- Vectorii au dimensiune fixa (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;
```

Vectori – considerente

- Utilizari ale vectorilor:
  - Lucrul cu colecţii de date de acelaşi 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.

Vectori - exercitii

- Scrieti un program care va inversa elementele unui vector
  - Lungimea vectorului va fi citita de la tastatura

Va multumesc!