

# Curso C# Completo

## Programação Orientada a Objetos + Projetos

**Capítulo: Comportamento de memória, arrays, listas**

<http://educandoweb.com.br>

Prof. Dr. Nelio Alves

## Tipos referência vs. tipos valor

<http://educandoweb.com.br>

Prof. Dr. Nelio Alves

## Classes são tipos referência

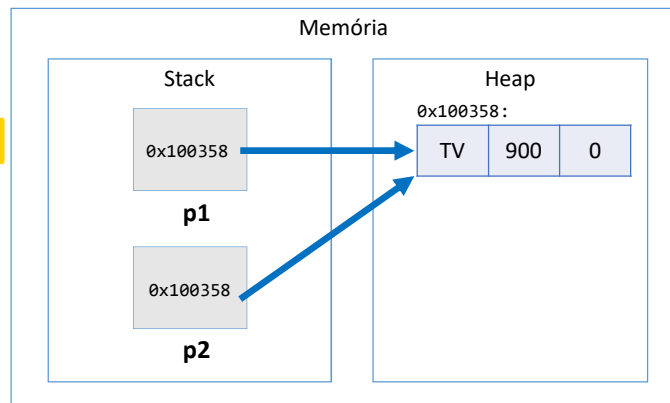
Variáveis cujo tipo são classes não devem ser entendidas como caixas, mas sim “tentáculos” (ponteiros) para caixas

```
Product p1, p2;
```

```
p1 = new Product("TV", 900.00, 0);
```

```
p2 = p1;
```

p2 = p1;  
"p2 passa a apontar para onde p1 aponta"

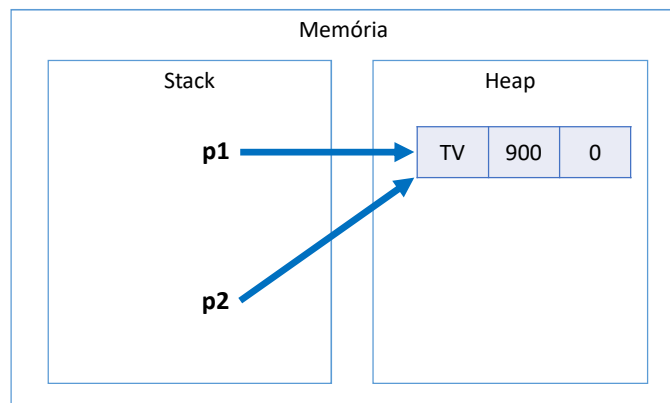


## Desenho simplificado

```
Product p1, p2;
```

```
p1 = new Product("TV", 900.00, 0);
```

```
p2 = p1;
```



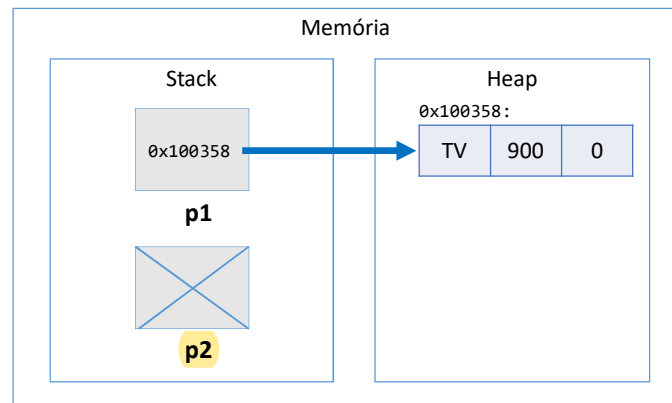
# Valor "null"

Tipos referência aceitam o valor "null", que indica que a variável aponta pra ninguém.

```
Product p1, p2;
```

```
p1 = new Product("TV", 900.00, 0);
```

```
p2 = null;
```



# Structs são tipos valor

A linguagem C# possui também tipos valor, que são os "structs". Structs são CAIXAS e não ponteiros.

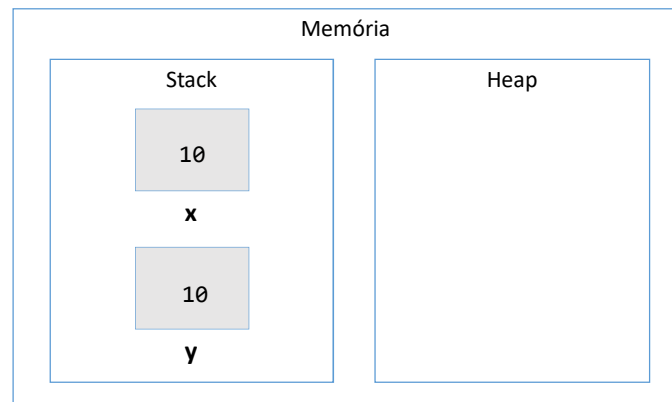
```
double x, y;
```

```
x = 10;
```

```
y = x;
```

```
y = x;
```

"y recebe uma CÓPIA de x"



C# Type	.Net Framework Type	Signed	Bytes	Possible Values
sbyte	System.Sbyte	Yes	1	-128 to 127
short	System.Int16	Yes	2	-32768 to 32767
int	System.Int32	Yes	4	$-2^{31}$ to $2^{31} - 1$
long	System.Int64	Yes	8	$-2^{63}$ to $2^{63} - 1$
byte	System.Byte	No	1	0 to 255
ushort	System.UInt16	No	2	0 to 65535
uint	System.UInt32	No	4	0 to $2^{32} - 1$
ulong	System.UInt64	No	8	0 to $2^{64} - 1$
float	System.Single	Yes	4	$\pm 1.5 \times 10^{-45}$ to $\pm 3.4 \times 10^{38}$ with 7 significant figures
double	System.Double	Yes	8	$\pm 5.0 \times 10^{-324}$ to $\pm 1.7 \times 10^{308}$ with 15 or 16 significant figures
decimal	System.Decimal	Yes	12	$\pm 1.0 \times 10^{-28}$ to $\pm 7.9 \times 10^{28}$ with 28 or 29 significant figures
char	System.Char	N/A	2	Any Unicode character
bool	System.Boolean	N/A	1/2	true or false

Outros structs importantes: DateTime, TimeSpan

## É possível criar seus próprios structs

```
namespace Course {
    struct Point {
        public double X, Y;

        public override string ToString() {
            return "(" + X + ", " + Y + ")";
        }
    }
}
```

## Structs e inicialização

- Demo:

```
Point p;  
Console.WriteLine(p); // erro: variável não atribuída  
p.X = 10;  
p.Y = 20;  
Console.WriteLine(p);  
p = new Point();  
Console.WriteLine(p);
```

## Valores padrão

- Quando **alocamos (new)** qualquer tipo estruturado (classe, struct, array), são atribuídos valores padrão aos seus elementos
  - números: 0
  - bool: False
  - char: caractere código 0
  - objeto: null
- Lembrando: **uma variável apenas declarada, mas não instanciada, inicia em estado "não atribuída", e o próprio compilador não permite que ela seja acessada.**

## Tipos referência vs. tipos valor

CLASSE	STRUCT
Vantagem: usufrui de todos recursos OO	Vantagem: é mais simples e mais performático
Variáveis são ponteiros	Variáveis são caixas
Objetos precisam ser instanciadas usando new, ou apontar para um objeto já existente.	Não é preciso instanciar usando new, mas é possível
Aceita valor null	Não aceita valor null
Suporte a herança	Não tem suporte a herança (mas pode implementar interfaces)
Y = X; "Y passa a apontar para onde X aponta"	Y = X; "Y recebe uma cópia de X"
Objetos instanciados no heap	Objetos instanciados no stack
Objetos não utilizados são desalocados em um momento próximo pelo garbage collector	"Objetos" são desalocados imediatamente quando seu escopo de execução é finalizado

## Desalocação de memória - garbage collector e escopo local

<http://educandoweb.com.br>

Prof. Dr. Nelio Alves

## Garbage collector

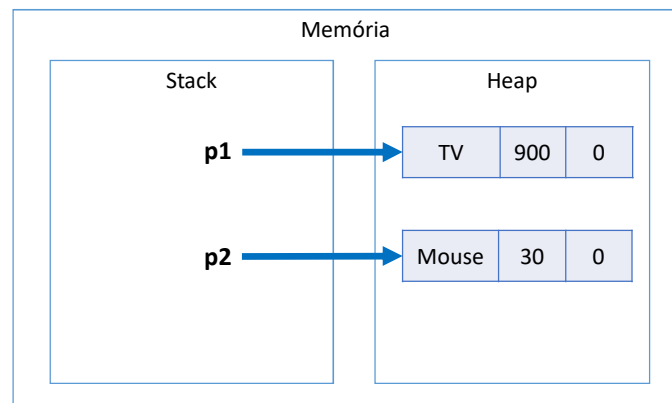
- É um processo que automatiza o gerenciamento de memória de um programa em execução
- O garbage collector monitora os objetos alocados dinamicamente pelo programa (no heap), desalocando aqueles que não estão mais sendo utilizados.

## Desalocação por garbage collector

```
Product p1, p2;
```

```
p1 = new Product("TV", 900.00, 0);
```

```
p2 = new Product("Mouse", 30.00, 0);
```



# Desalocação por garbage collector



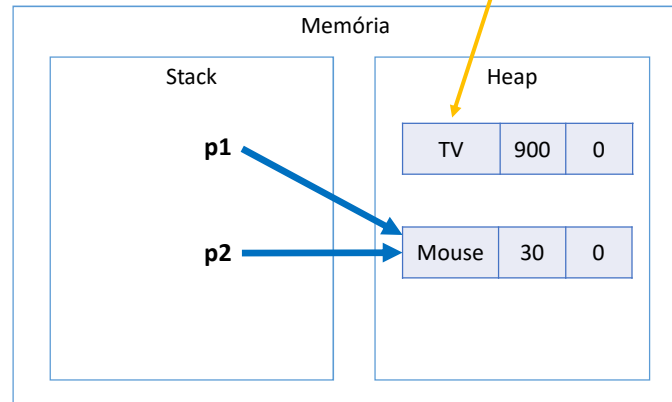
Product p1, p2;

p1 = new Product("TV", 900.00, 0);

p2 = new Product("Mouse", 30.00, 0);

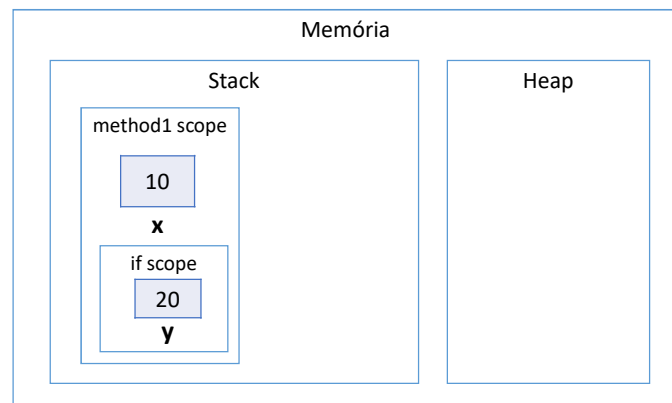
**p1 = p2;**

Será desalocado  
pelo garbage  
collector



# Desalocação por escopo

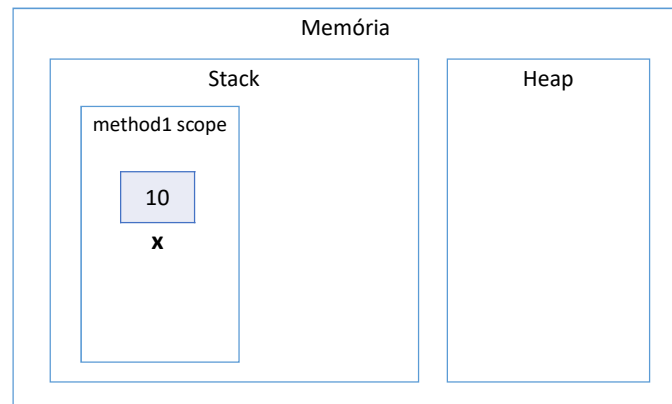

```
void method1() {  
    int x = 10;  
    if (x > 0) {  
        int y = 20;  
        Console.WriteLine(x);  
    }  
}
```





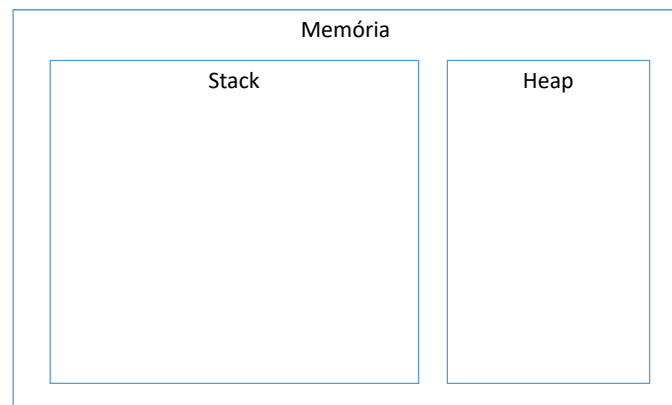

## Desalocação por escopo

```
void method1() {  
    int x = 10;  
    if (x > 0) {  
        int y = 20;  
    }  
    Console.WriteLine(x);  
}
```



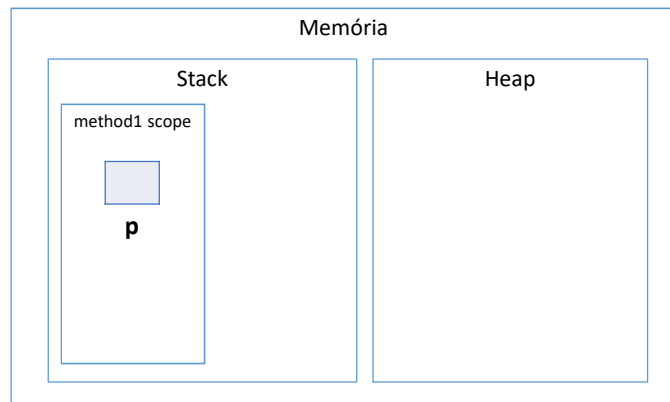
## Desalocação por escopo

```
void method1() {  
    int x = 10;  
    if (x > 0) {  
        int y = 20;  
    }  
    Console.WriteLine(x);  
}
```



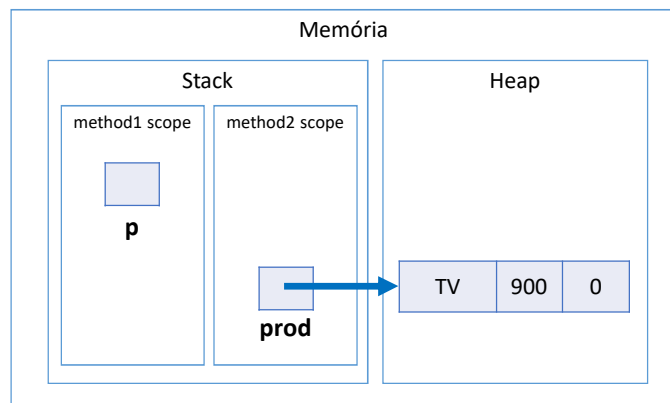
## Outro exemplo

```
void method1() {  
    ➡ Product p = method2();  
    Console.WriteLine(p.Name);  
}  
  
Product method2() {  
    Product prod = new Product("TV", 900.0, 0);  
    return prod;  
}
```



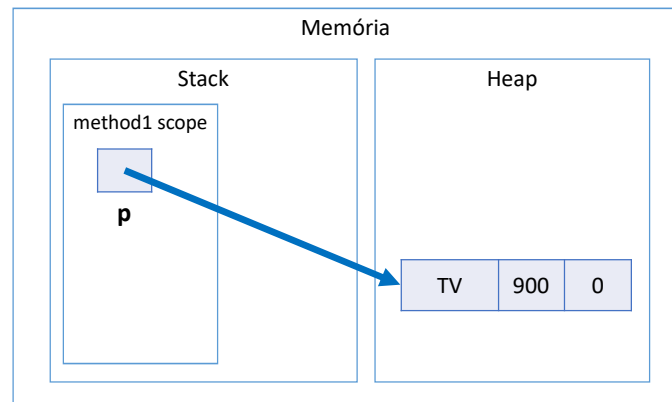
## Outro exemplo

```
void method1() {  
    Product p = method2();  
    Console.WriteLine(p.Name);  
}  
  
Product method2() {  
    ➡ Product prod = new Product("TV", 900.0, 0);  
    return prod;  
}
```



## Outro exemplo

```
void method1() {  
    Product p = method2();  
    Console.WriteLine(p.Name);  
}  
  
Product method2() {  
    Product prod = new Product("TV", 900.0, 0);  
    return prod;  
}
```



## Resumo

- Objetos alocados dinamicamente, quando não possuem mais referência para eles, serão desalocados pelo garbage collector
- Variáveis locais são desalocadas imediatamente assim que seu escopo local sai de execução

# Nullable

<http://educandoweb.com.br>

Prof. Dr. Nelio Alves

## Nullable

- É um recurso de C# para que dados de tipo valor (structs) possam receber o valor null
- Uso comum:
  - Campos de banco de dados que podem valer nulo (data de nascimento, algum valor numérico, etc.).
  - Dados e parâmetros opcionais.

# Demo

```
double x = null; // erro
```

```
Nullable<double> x = null;
```

```
double? x = null;
```

- Métodos:
  - GetValueOrDefault
  - HasValue
  - Value (lança uma exceção se não houver valor)
- Um nullable não pode ser atribuído para um struct comum
- Valor default para tipos:  
<https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/statements-expressions-operators/default-value-expressions>

```
using System;

namespace Course {
    class Program {
        static void Main(string[] args) {

            double? x = null;
            double? y = 10.0;

            Console.WriteLine(x.GetValueOrDefault());
            Console.WriteLine(y.GetValueOrDefault());

            Console.WriteLine(x.HasValue);
            Console.WriteLine(y.HasValue);

            if (x.HasValue)
                Console.WriteLine(x.Value);
            else
                Console.WriteLine("X is null");

            if (y.HasValue)
                Console.WriteLine(y.Value);
            else
                Console.WriteLine("Y is null");

        }
    }
}
```

## Operador de coalescência nula

- <https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/null-conditional-operator>

- Demo:

```
double? x = null;
```

```
double y = x ?? 0.0;
```



## Vetores - Parte 1

<http://educandoweb.com.br>

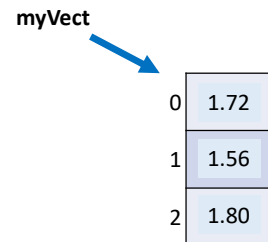
Prof. Dr. Nelio Alves

# Checklist

- Revisão do conceito de vetor
- Manipulação de vetor de elementos tipo structs
- Manipulação de vetor de elementos tipo classe
- Acesso aos elementos
- Propriedade Length

## Vetores

- Em programação, "vetor" é o nome dado a arranjos unidimensionais
- Arranjo é uma estrutura de dados:
  - Homogênea (dados do mesmo tipo)
  - Ordenada (elementos acessados por meio de posições)
  - Alocada de uma vez só, em um bloco contíguo de memória
- Vantagens:
  - Acesso imediato aos elementos pela sua posição
- Desvantagens:
  - Tamanho fixo
  - Dificuldade para se realizar inserções e deleções



The diagram illustrates a variable named 'myVect' pointing to the first element of a vector. The vector is represented as a table with three rows, indexed 0, 1, and 2. The values are 1.72, 1.56, and 1.80 respectively. A blue arrow points from 'myVect' to the first row (index 0).

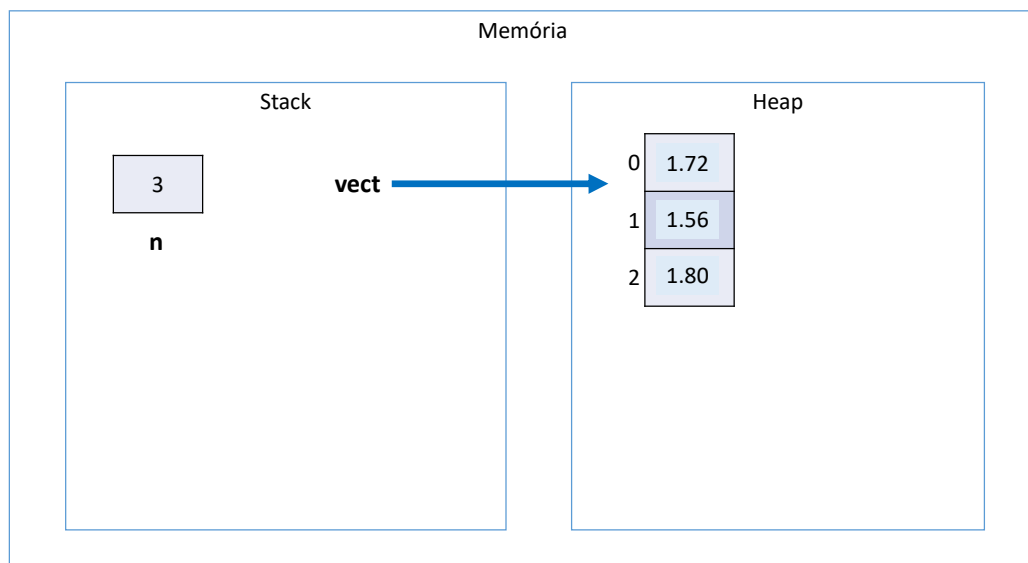
0	1.72
1	1.56
2	1.80

## Problema exemplo 1

Fazer um programa para ler um número inteiro N e a altura de N pessoas. Armazene as N alturas em um vetor. Em seguida, mostrar a altura média dessas pessoas.

### Exemplo:

Entrada:	Saída:
3 1.72 1.56 1.80	AVERAGE HEIGHT = 1.69





```

using System;
using System.Globalization;

namespace Course {
    class Program {
        static void Main(string[] args) {

            int n = int.Parse(Console.ReadLine());

            double[] vect = new double[n];

            for (int i = 0; i < n; i++) {
                vect[i] = double.Parse(Console.ReadLine(), CultureInfo.InvariantCulture);
            }

            double sum = 0.0;
            for (int i = 0; i < n; i++) {
                sum += vect[i];
            }

            double avg = sum / n;

            Console.WriteLine("AVERAGE HEIGHT = " + avg.ToString("F2", CultureInfo.InvariantCulture));
        }
    }
}

```

## Vetores - Parte 2

<http://educandoweb.com.br>

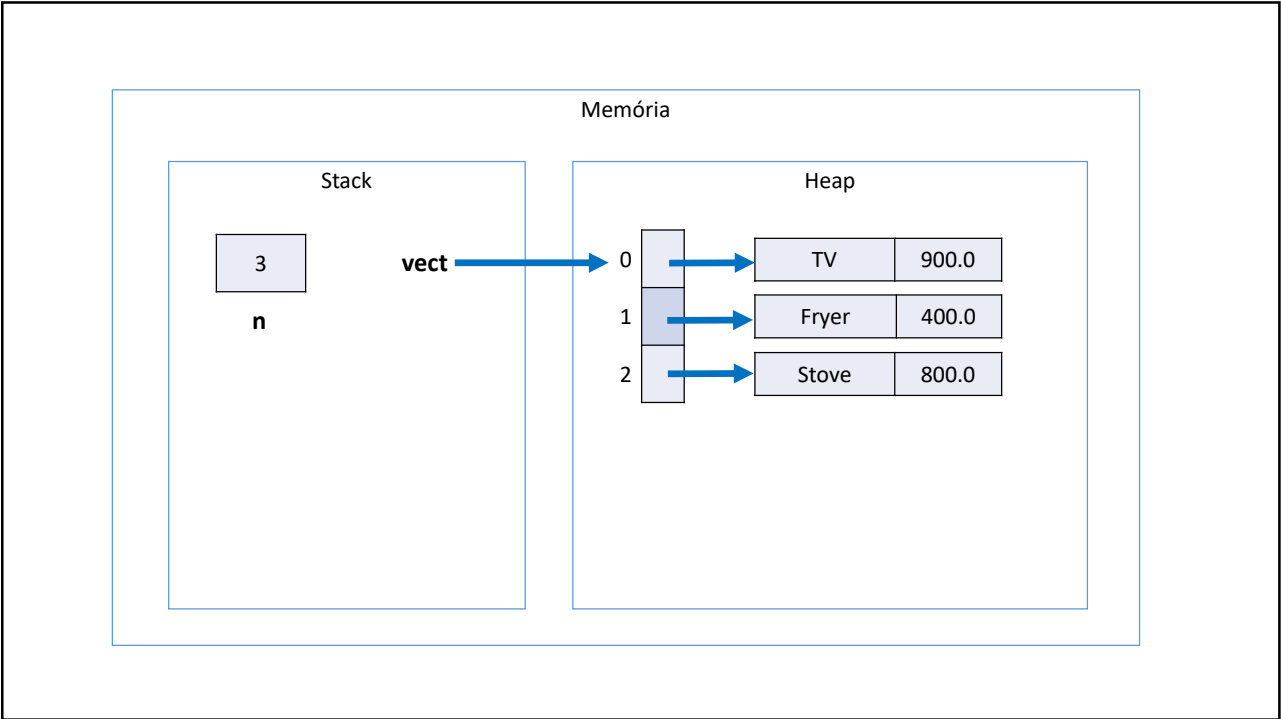
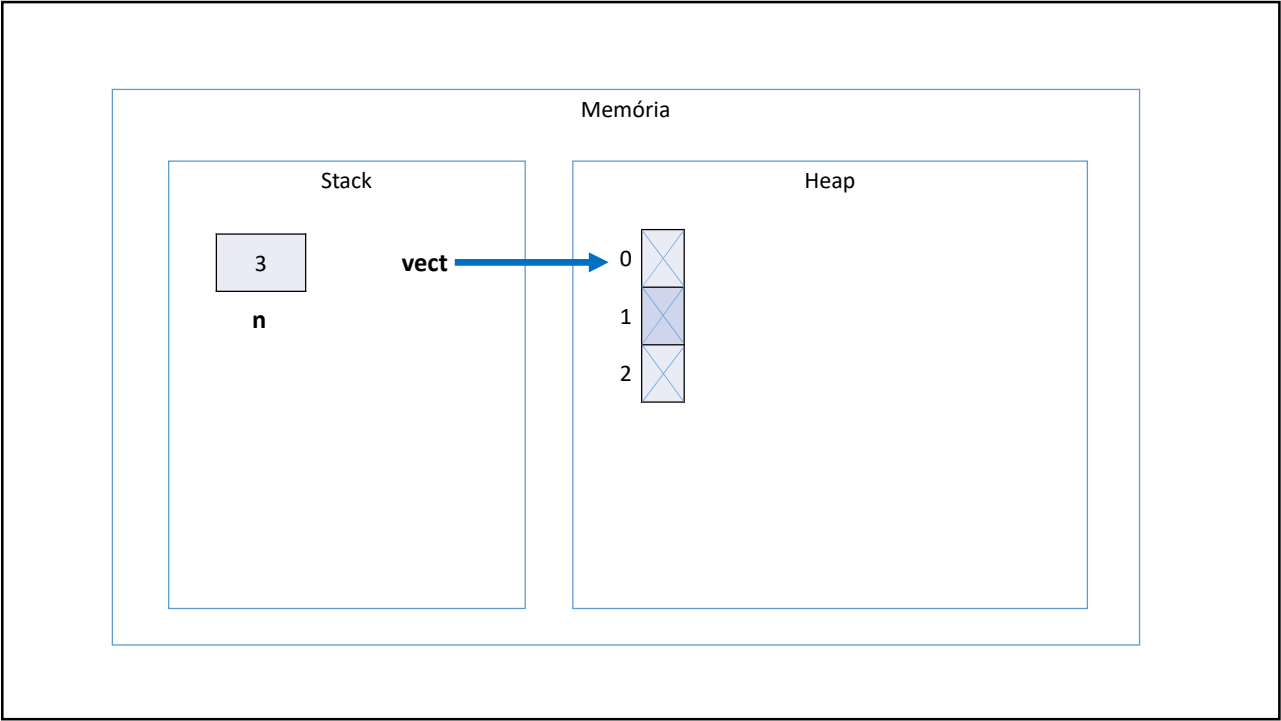
Prof. Dr. Nelio Alves

## Problema exemplo 2

Fazer um programa para ler um número inteiro N e os dados (nome e preço) de N Produtos. Armazene os N produtos em um vetor. Em seguida, mostrar o preço médio dos produtos.

## Example

Input:	Output:
3 TV 900.00 Fryer 400.00 Stove 800.00	AVERAGE PRICE = 700.00



```
using System;
using System.Globalization;

namespace Course {
    class Program {
        static void Main(string[] args) {

            int n = int.Parse(Console.ReadLine());

            Product[] vect = new Product[n];

            for (int i = 0; i < n; i++) {
                string name = Console.ReadLine();
                double price = double.Parse(Console.ReadLine(), CultureInfo.InvariantCulture);
                vect[i] = new Product { Name = name, Price = price };
            }

            double sum = 0.0;
            for (int i = 0; i < n; i++) {
                sum += vect[i].Price;
            }

            double avg = sum / n;
            Console.WriteLine("AVERAGE PRICE = " + avg.ToString("F2", CultureInfo.InvariantCulture));
        }
    }
}
```

## Exercício de fixação

<http://educandoweb.com.br>

Prof. Dr. Nelio Alves

A dona de um pensionato possui dez quartos para alugar para estudantes, sendo esses quartos identificados pelos números 0 a 9.

Quando um estudante deseja alugar um quarto, deve-se registrar o nome e email deste estudante.

Fazer um programa que inicie com todos os dez quartos vazios, e depois leia uma quantidade N representando o número de estudantes que vão alugar quartos (N pode ser de 1 a 10). Em seguida, registre o aluguel dos N estudantes. Para cada registro de aluguel, informar o nome e email do estudante, bem como qual dos quartos ele escolheu (de 0 a 9). Suponha que seja escolhido um quarto vago. Ao final, seu programa deve imprimir um relatório de todas ocupações do pensionato, por ordem de quarto, conforme exemplo.

Quantos quartos serão alugados? 3

Aluguel #1:

Nome: Maria Green

Email: maria@gmail.com

Quarto: 5

Aluguel #2:

Nome: Marco Antonio

Email: marco@gmail.com

Quarto: 1

Aluguel #3:

Nome: Alex Brown

Email: alex@gmail.com

Quarto: 8

Quartos ocupados:

1: Marco Antonio, marco@gmail.com

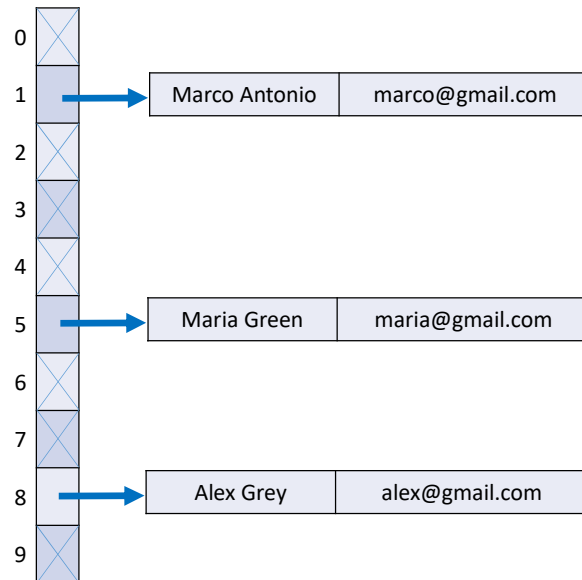
5: Maria Green, maria@gmail.com

8: Alex Brown, alex@gmail.com

## Sugestão

```
if (vect[i] != null)
```

(correção na próxima página)



```
namespace Course {  
    class Estudante {  
        public string Nome { get; set; }  
        public string Email { get; set; }  
  
        public Estudante(string nome, string email) {  
            Nome = nome;  
            Email = email;  
        }  
  
        public override string ToString() {  
            return Nome + ", " + Email;  
        }  
    }  
}
```

```

using System;

namespace Course {
    class Program {
        static void Main(string[] args) {

            Estudante[] vect = new Estudante[10];

            Console.Write("Quantos quartos serão alugados? ");
            int n = int.Parse(Console.ReadLine());

            for (int i = 1; i <= n; i++) {

                Console.WriteLine();
                Console.WriteLine($"Aluguel #{i}:");
                Console.Write("Nome: ");
                string nome = Console.ReadLine();
                Console.Write("Email: ");
                string email = Console.ReadLine();
                Console.Write("Quarto: ");
                int quarto = int.Parse(Console.ReadLine());
                vect[quarto] = new Estudante(nome, email);
            }

            Console.WriteLine();
            Console.WriteLine("Quartos ocupados:");
            for (int i = 0; i < 10; i++) {
                if (vect[i] != null) {
                    Console.WriteLine(i + ": " + vect[i]);
                }
            }
        }
    }
}

```

# Modificador de parâmetros: params

<http://educandoweb.com.br>

Prof. Dr. Nelio Alves

# Modificador params

Suponha que se queira uma calculadora para calcular a soma de uma quantidade variável de valores. Solução ruim usando sobrecarga:

```
namespace Course {  
    class Calculator {  
  
        public static int Sum(int n1, int n2) {  
            return n1 + n2;  
        }  
  
        public static int Sum(int n1, int n2, int n3) {  
            return n1 + n2 + n3;  
        }  
  
        public static int Sum(int n1, int n2, int n3, int n4) {  
            return n1 + n2 + n3 + n4;  
        }  
    }  
}
```

## Solução com vetor:

```
namespace Course {  
    class Calculator {  
  
        public static int Sum(int[] numbers) {  
            int sum = 0;  
            for (int i=0; i<numbers.Length; i++) {  
                sum += numbers[i];  
            }  
            return sum;  
        }  
    }  
}
```



```
int result = Calculator.Sum(new int[] { 10, 20, 30, 40 });
```



## Solução com modificador params:

```
namespace Course {  
    class Calculator {  
  
        public static int Sum(params int[] numbers) {  
            int sum = 0;  
            for (int i=0; i<numbers.Length; i++) {  
                sum += numbers[i];  
            }  
            return sum;  
        }  
    }  
}
```

```
int result = Calculator.Sum(10, 20, 30, 40);
```

## Modificador de parâmetros: ref e out

<http://educandoweb.com.br>

Prof. Dr. Nelio Alves

# Modificador ref

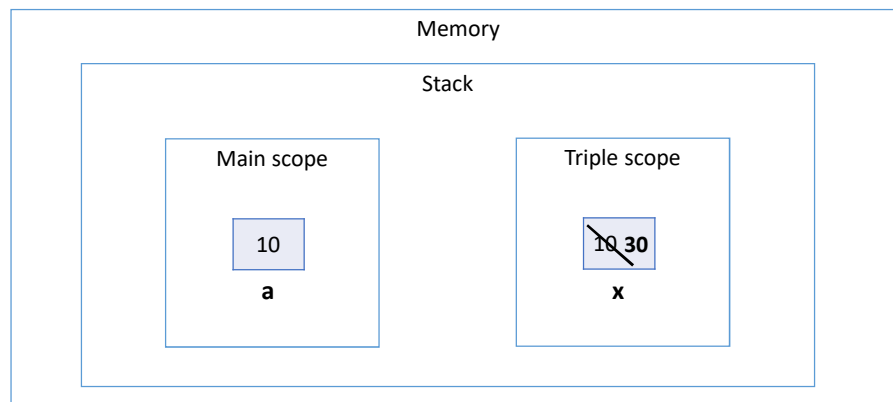
Suponha que se queira uma calculadora com uma operação para triplicar o valor de um número passado como parâmetro. A seguir uma solução que não funciona:

```
class Calculator {  
  
    public static void Triple(int x) {  
        x = x * 3;  
    }  
}
```

```
class Program {  
    static void Main(string[] args) {  
  
        int a = 10;  
        Calculator.Triple(a);  
        Console.WriteLine(a);  
    }  
}
```

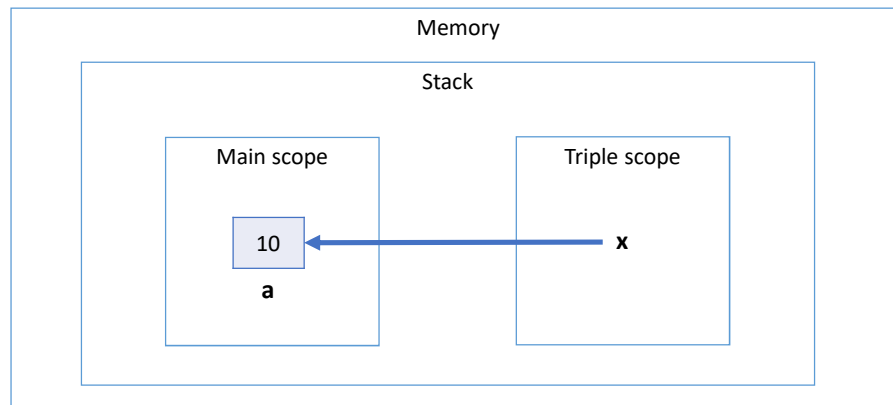
```
class Program {  
    static void Main(string[] args) {  
  
        int a = 10;  
        Calculator.Triple(a);  
        Console.WriteLine(a);  
    }  
}
```

```
class Calculator {  
  
    public static void Triple(int x) {  
        x = x * 3;  
    }  
}
```



```
class Program {
    static void Main(string[] args) {
        int a = 10;
        Calculator.Triple(ref a);
        Console.WriteLine(a);
    }
}
```

```
class Calculator {
    public static void Triple(ref int x) {
        x = x * 3;
    }
}
```



## Modificador out

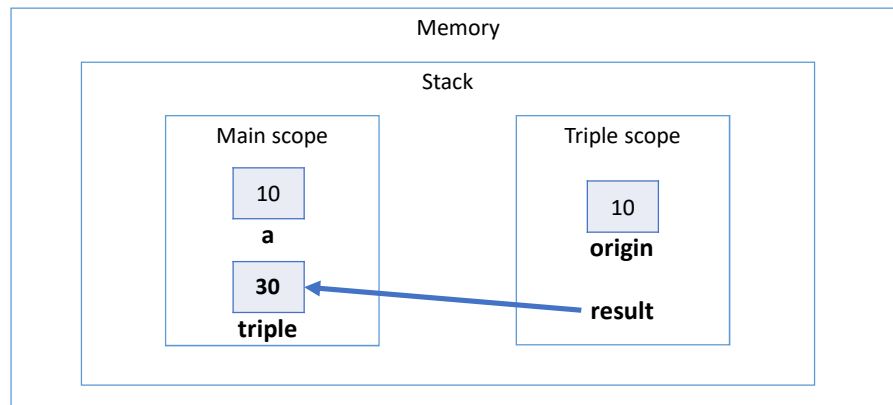
O modificador out é similar ao ref (faz o parâmetro ser uma referência para a variável original), mas não exige que a variável original seja iniciada.

```
class Calculator {
    public static void Triple(int origin, out int result) {
        result = origin * 3;
    }
}
```

```
class Program {
    static void Main(string[] args) {
        int a = 10;
        int triple;
        Calculator.Triple(a, out triple);
        Console.WriteLine(triple);
    }
}
```

```
class Program {
    static void Main(string[] args) {
        int a = 10;
        int triple;
        Calculator.Triple(a, out triple);
        Console.WriteLine(triple);
    }
}
```

```
class Calculator {
    public static void Triple(int origin, out int result) {
        result = origin * 3;
    }
}
```



## Considerações sobre ref e out

- Diferença:
  - A variável passada como parâmetro **ref** DEVE ter sido iniciada
  - A variável passada como parâmetro **out** não precisa ter sido iniciada
- Conclusão: ambos são muito similares, mas ref é uma forma de fazer o compilador obrigar o usuário a iniciar a variável.
- Nota: ambos são considerados "code smells" (design ruim) e devem ser evitados.

# Boxing e unboxing

<http://educandoweb.com.br>

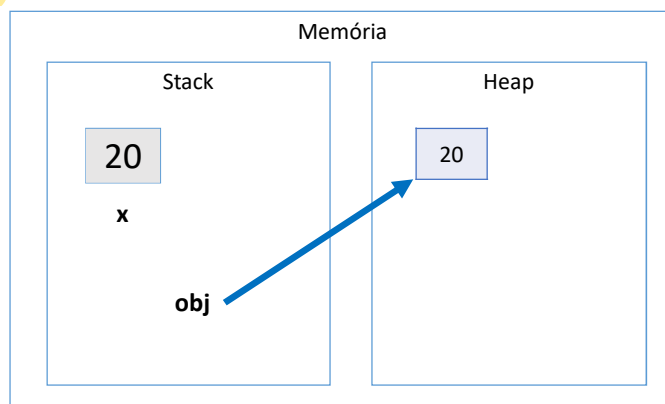
Prof. Dr. Nelio Alves

## Boxing

- É o processo de conversão de um objeto tipo valor para um objeto tipo referência compatível

```
int x = 20;
```

```
Object obj = x;
```



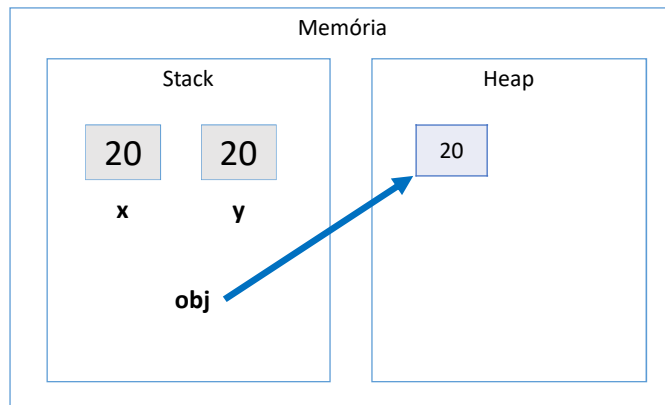
## Unboxing

- É o processo de conversão de um objeto tipo referência para um objeto tipo valor compatível

```
int x = 20;
```

```
Object obj = x;
```

```
int y = (int) obj;
```



## Sintaxe opcional: laço foreach

<http://educandoweb.com.br>

Prof. Dr. Nelio Alves

## Laço for each

Sintaxe opcional e simplificada para percorrer coleções

Leitura: "para cada objeto 'obj' contido em vect, faça:"

```
string[] vect = new string[] { "Maria", "Bob", "Alex"};

foreach (string obj in vect) {
    Console.WriteLine(obj);
}
```

## Listas - Parte 1

<http://educandoweb.com.br>

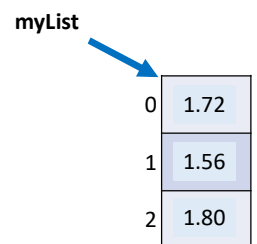
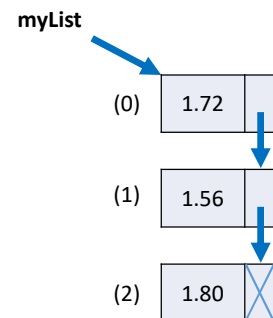
Prof. Dr. Nelio Alves

# Checklist

- Conceito de lista
- Tipo List - Declaração, instanciação
- Referência: [https://msdn.microsoft.com/en-us/library/6sh2ey19\(v=vs.110\).aspx](https://msdn.microsoft.com/en-us/library/6sh2ey19(v=vs.110).aspx)
- Assuntos pendentes:
  - generics
  - predicados (lambda)

# Listas

- Lista é uma estrutura de dados:
  - Homogênea (dados do mesmo tipo)
  - Ordenada (elementos acessados por meio de posições)
  - Inicia vazia, e seus elementos são alocados sob demanda
  - Cada elemento ocupa um "nó" (ou nodo) da lista
- Classe: List
- Namespace: `System.Collections.Generic`
- Vantagens:
  - Tamanho variável
  - Facilidade para se realizar inserções e deleções
- Desvantagens:
  - Acesso sequencial aos elementos \*



(desenho simplificado)



# Listas - Parte 2

<http://educandoweb.com.br>

Prof. Dr. Nelio Alves

## Demo

- Inserir elemento na lista: `Add`, `Insert`
- Tamanho da lista: `Count`
- Encontrar primeiro ou último elementos da lista que satisfaça um predicado: `list.Find`, `list.FindLast`
- Encontrar primeira ou última posição de elemento da lista que satisfaça um predicado: `list.FindIndex`, `list.FindLastIndex`
- Filtrar a lista com base em um predicado: `list.FindAll`
- Remover elementos da lista: `Remove`, `RemoveAll`, `RemoveAt`, `RemoveRange`
- Assuntos pendentes:
  - Generics
  - Predicados (lambda)

```

List<string> list = new List<string>();

list.Add("Maria");
list.Add("Alex");
list.Add("Bob");
list.Add("Anna");
list.Insert(2, "Marco");

foreach (string obj in list) {
    Console.WriteLine(obj);
}
Console.WriteLine("List count: " + list.Count);

string s1 = list.Find(x => x[0] == 'A');
Console.WriteLine("First 'A': " + s1);

string s2 = list.FindLast(x => x[0] == 'A');
Console.WriteLine("Last 'A': " + s2);

int pos1 = list.FindIndex(x => x[0] == 'A');
Console.WriteLine("First position 'A': " + pos1);

int pos2 = list.FindLastIndex(x => x[0] == 'A');
Console.WriteLine("Last position 'A': " + pos2);

List<string> list2 = list.FindAll(x => x.Length == 5);
Console.WriteLine("-----");
foreach (string obj in list2) {
    Console.WriteLine(obj);
}

list.Remove("Alex");
Console.WriteLine("-----");
foreach (string obj in list) {
    Console.WriteLine(obj);
}

list.RemoveAll(x => x[0] == 'M');
Console.WriteLine("-----");
foreach (string obj in list) {
    Console.WriteLine(obj);
}

```

# Exercício de fixação (listas)

<http://educandoweb.com.br>

Prof. Dr. Nelio Alves

Fazer um programa para ler um número inteiro N e depois os dados (id, nome e salario) de N funcionários. Não deve haver repetição de id.

Em seguida, efetuar o aumento de X por cento no salário de um determinado funcionário. Para isso, o programa deve ler um id e o valor X. Se o id informado não existir, mostrar uma mensagem e abortar a operação. Ao final, mostrar a listagem atualizada dos funcionários, conforme exemplos.

Lembre-se de aplicar a técnica de encapsulamento para não permitir que o salário possa ser mudado livremente. Um salário só pode ser aumentado com base em uma operação de aumento por porcentagem dada.

(exemplo na próxima página)

How many employees will be registered? 3

Employee #1:

Id: 333

Name: Maria Brown

Salary: 4000.00

Employee #2:

Id: 536

Name: Alex Grey

Salary: 3000.00

Employee #3:

Id: 772

Name: Bob Green

Salary: 5000.00

Enter the employee id that will have salary increase : 536

Enter the percentage: 10.0

Updated list of employees:

333, Maria Brown, 4000.00

536, Alex Grey, 3300.00

772, Bob Green, 5000.00

How many employees will be registered? **2**

Employee #1:

Id: **333**

Name: **Maria Brown**

Salary: **4000.00**

Employee #2:

Id: **536**

Name: **Alex Grey**

Salary: **3000.00**

Enter the employee id that will have salary increase: **776**

This id does not exist!

Updated list of employees:

333, Maria Brown, 4000.00

536, Alex Grey, 3000.00

Employee
- id : Integer - name : String - salary : Double
+ increaseSalary(percentage : double) : void

<https://github.com/acenelio/list1-csharp>

# Matrizes

<http://educandoweb.com.br>

Prof. Dr. Nelio Alves

## Checklist

- Revisão do conceito de matriz
- Declaração e instanciação
- Acesso aos elementos / como percorrer uma matriz
- Propriedade Length, Rank e GetLength
- Referência: <https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/arrays/multidimensional-arrays>

# Matrizes

- Em programação, "matriz" é o nome dado a arranjos **bidimensionais**

- Arranjo é uma estrutura de dados:

- **Homogênea** (dados do mesmo tipo)
- **Ordenada** (elementos acessados por meio de posições)
- **Alocada de uma vez só, em um bloco contíguo de memória**


- Vantagens:

- **Acesso imediato aos elementos pela sua posição**

- Desvantagens:



- **Tamanho fixo**
- **Dificuldade para se realizar inserções e deleções**


myMat





	0	1	2	3
0	3.5	17.0	12.3	8.2
1	4.1	6.2	7.5	2.9
2	11.0	9.5	14.8	21.7


## Demo

   
`double[,] mat = new double[2, 3];`

`Console.WriteLine(mat.Length);` 

`Console.WriteLine(mat.Rank);` 

`Console.WriteLine(mat.GetLength(0));` 

`Console.WriteLine(mat.GetLength(1));` 

## Exercício resolvido

<http://educandoweb.com.br>

Prof. Dr. Nelio Alves

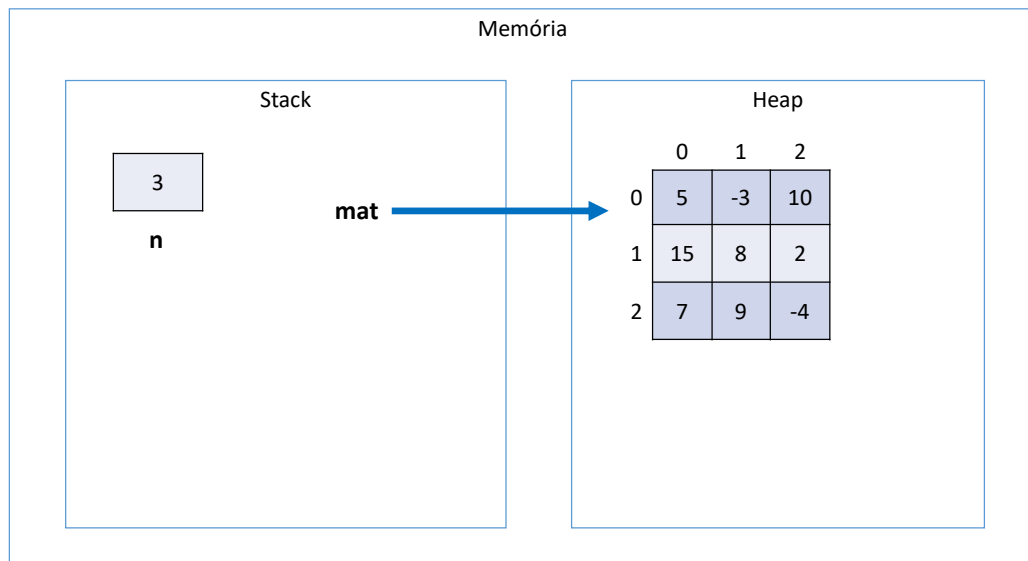
## Exercício resolvido

Fazer um programa para ler um número inteiro  $N$  e uma matriz de ordem  $N$  contendo números inteiros. Em seguida, mostrar a diagonal principal e a quantidade de valores negativos da matriz.

## Example

Input:	Output:
3 5 -3 10 15 8 2 7 9 -4	Main diagonal: 5 8 -4 Negative numbers = 2

<https://github.com/acenelio/matrix1-csharp>





# Exercício de fixação

<http://educandoweb.com.br>

Prof. Dr. Nelio Alves

Fazer um programa para ler dois números inteiros M e N, e depois ler uma matriz de M linhas por N colunas contendo números inteiros, podendo haver repetições. Em seguida, ler um número inteiro X que pertence à matriz. Para cada ocorrência de X, mostrar os valores à esquerda, acima, à direita e abaixo de X, quando houver, conforme exemplo.

## Example

```
3 4
10 8 15 12
21 11 23 8
14 5 13 19
8
Position 0,1:
Left: 10
Right: 15
Down: 11
Position 1,3:
Left: 23
Up: 12
Down: 19
```

<https://github.com/acenelio/matrix2-csharp>