

ESEIG

**POLITÉCNICO
DO PORTO**

Algoritmia e Estruturas de Dados

Módulo II – C# Métodos e Funções



- I. Subprogramas ou Procedimentos
 1. Conceito de procedimentos e funções
 2. Métodos
 3. Assinatura
 1. Níveis de acesso
 2. Tipo de retorno
 3. Parâmetros
 4. Boas práticas
 5. Variáveis Globais

► Conceito

Conjunto de código (ou bloco de instruções), delimitado de forma clara , e que executa uma tarefa específica.

Um subprograma comporta-se da mesma forma que um programa completo, embora numa escala diferente:

- é executado depois de invocado pelo programa que o chama;
- quando termina, devolve a execução do programa, a partir do local onde foi chamado.

Na programação orientada a objetos, os subprogramas (ou procedimentos) designam-se por **Métodos**.

▶ Conceito

- ▶ Em programação estruturada, os subprogramas podem assumir a forma de procedimentos ou funções:
 - ▶ Um **Procedimento** executa um conjunto de código e **NÃO** retorna qualquer valor
 - ▶ Uma **Função** retorna um valor, no final da sua execução

As variáveis definidas no âmbito de um método (procedimento ou função) são **variáveis locais** – existem dentro desse método, em contraponto a variáveis globais – existem durante toda a execução do programa

▶ Conceito

▶ Objetivos principais dos métodos:

- ▶ Organizar e **dividir** o programa em partes mais pequenas e mais fáceis de codificar e testar;
- ▶ Permitir a **reutilização** de código;
- ▶ Implementar o conceito da **abstração**

► Conceito

- Em C# cada instrução é executada no contexto de um **método**
- **Método:**
 - Bloco de código que contém uma série de instruções
 - Declarado no contexto de uma classe
 - Exemplo: o método **Main** pertence à classe **Program**, é o ponto de entrada de uma aplicação C# e é chamado pelo CLR quando o programa é iniciado

```
class Program
{
    static void Main(string[] args)
    {
        //código do método Main da classe Program
    }
}
```

▶ Assinatura

▶ A definição de um método contempla as seguintes partes:

- ▶ Nível de acesso (ex.: public, private, protected, etc.)
- ▶ Modificadores opcionais (ex.: static, abstract, sealed, etc.)
- ▶ Tipo de retorno
- ▶ Nome do método
- ▶ Parâmetros

▶ Estas peças em conjunto, são a **assinatura do método**

```
public static int soma(int num1, int num2)
{
    // código do método
}
```

Os parâmetros estão entre parênteses e separados por vírgulas. Se os parênteses estiverem vazios significa que o método não tem parâmetros.

- ▶ Assinatura
 - ▶ Níveis de Acesso (I)
 - ▶ **Public**: pode ser usado em qualquer outra classe
 - ▶ **Private**: só pode ser usado na classe em que foi declarado
 - ▶ **Protected**: só pode ser usado na própria classe em que foi declarado, ou em classes filho

Um método definido sem nível de acesso é considerado privado!

▶ Assinatura

▶ Níveis de Acesso (2)

- ▶ Quando os métodos encontram-se em outras classes devemos usar a nomenclatura **nomeClasse.nomeMétodo([argumentos])**

```
class Program
{
    static void Main(string[] args)
    {
        int contVogais = 0;
        String palavra = "";
        Console.Write("Escreva uma palavra: ");
        palavra = Console.ReadLine();
        contVogais = Utils.nVogais(palavra);
        Console.WriteLine("O nº de vogais é {0}",
contVogais);
    }
}
```

```
class Utils
{
    public static int nVogais(String palavra)
    {
        int cont = 0;
        foreach (char letra in palavra)
            if(letra=='a' || letra=='e' || letra=='i' || letra=='o' ||
letra=='u')
                cont++;
        return cont;
    }
}
```

- ▶ Tipo de retorno (I)
 - ▶ Se um método não retornar nada deve ser declarado na assinatura com a palavra-chave **void**

```
// Procedimento de inicializar matriz
static void iniciar_matriz()
{
    int [,] matriz = new int[3,3];
    for (int i=0; i<3; i++)
    {
        for (int j =0; j<3; j++)
        {
            Console.Write("Linha " + (i + 1) + " coluna " + (j + 1)+ " :");
            matriz[i,j]= int.Parse(Console.ReadLine());
        }
    }
}
```

```
static void Main(string[] args)
{
    char opcao=' ';
    while (opcao != '0')
    {
        Console.Clear();
        Console.WriteLine(" -----");
        Console.WriteLine("                MENU");
        Console.WriteLine(" -----");
        Console.WriteLine("1 - Inicializar matriz");
        Console.WriteLine("2 - Calcular Matriz Transposta");
        Console.WriteLine("3 - Determinar valor máximo");
        Console.WriteLine("0 - Sair");
        Console.Write("                Opção: ");
        opcao = Convert.ToChar(Console.ReadLine());
        switch (opcao)
        {
            case '1' : iniciar_matriz();
                        break;
            case '2': matriz_transposta();
                        break;
            case '3': maximo();
                        break;
            case '0' : break;
            default: Console.WriteLine("Opção Inválida");
                     Console.ReadLine();
                     break;
        } // fim do switch
    } // fim do while
} // fim do static void Main
```

- ▶ Tipo de retorno (2)
 - ▶ Os métodos podem retornar um valor para quem os invoca usando a palavra-chave **return**
 - ▶ O tipo de retorno do método (assinatura) deve ser igual ao do valor incluído na instrução precedida de **return**
 - ▶ A palavra-chave **return** termina a execução do método

- ▶ Tipo de retorno (2)
 - ▶ Os métodos podem retornar um valor para quem os invoca usando a palavra-chave **return**

```
static bool iguais (int val1, int val2) ← Método que retorna um  
{                                     valor booleano  
    bool result;  
    if (val1 == val2)  
        result = true;  
    else  
        result = false;  
    return result;  
}
```

// EXEMPLO da chamada de uma função que retorna um valor booleano

```
static void Main(string[] args)  
{  
    Console.Write("Numero1=");  
    int val1 = int.Parse(Console.ReadLine());  
    Console.Write("Numero2=");  
    int val2 = int.Parse(Console.ReadLine());  
    if (iguais(val1, val2) == true)  
        Console.WriteLine("os número são iguais");  
    else  
        Console.WriteLine("os número são diferentes");  
    Console.ReadLine();  
}
```

- ▶ Tipo de retorno (2)
 - ▶ Os métodos podem retornar um valor para quem os invoca usando a palavra-chave **return**

```
static string primeiro_nome(string nome)
{
    int pos = 0;
    pos = nome.IndexOf(" ");
    string nomep = nome.Substring(0, pos);
    return nomep;
}
```

← Método que retorna um valor do tipo string

// EXEMPLO da chamada de uma função que retorna uma string

```
static void Main(string[] args)
{
    Console.Write("nome=");
    string nome = Console.ReadLine();
    // chama função que devolve uma string com o nome próprio
    string nome_inicial = primeiro_nome(nome);

    Console.WriteLine("Nome Próprio=" + nome_inicial);
    Console.ReadLine();
}
```

- ▶ Tipo de retorno (2)
 - ▶ Os métodos podem retornar um valor para quem os invoca usando a palavra-chave **return**

Método que retorna um valor

```
public int dividir(int num1, int num2)
{
    int resultado = 0;
    if (num2 != 0)
        resultado = num1 / num2;
    return resultado;
}
```

Método que não retorna um valor

```
public void escreverArray(int[] numeros)
{
    for(int i=0;i<numeros.Length;i++)
        Console.Write("{0} ", numeros[i]);
}
```

- ▶ O tipo de retorno do método (assinatura) deve ser igual ao do valor incluído na instrução precedida de **return**
- ▶ A palavra-chave **return** pára a execução do método

- ▶ Tipo de Retorno (2)
 - ▶ O método é invocado pelo seu nome e parâmetros (opcional)
 - ▶ Após execução é retornado um valor para quem o invocou
 - ▶ O retorno é entregue a uma variável do mesmo tipo

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

```
{
```

```
    int num1, num2, res;
```

```
    Console.Write("1º número: ");
```

```
    num1 = Convert.ToInt16(Console.ReadLine());
```

```
    Console.Write("2º número: ");
```

```
    num2 = Convert.ToInt16(Console.ReadLine());
```

```
    res = dividir(num1, num2);
```

```
    Console.WriteLine("Resultado: {0}", res);
```

```
}
```

```
public int dividir(int num1, int num2)
```

```
{
```

```
    int resultado = 0;
```

```
    if (num2 != 0)
```

```
        resultado = num1 / num2;
```

```
    return resultado;
```

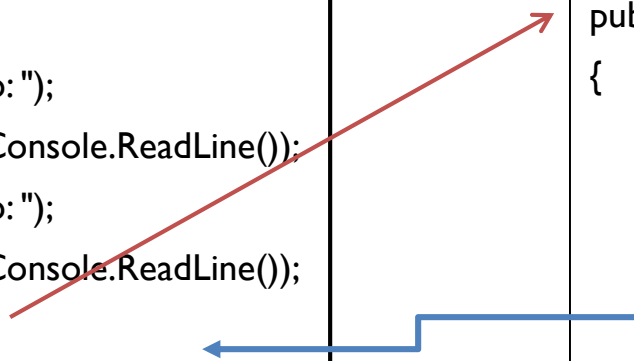
```
}
```

▶ Parâmetros(1)

- ▶ Os valores passados no método de invocação são **argumentos**
- ▶ Os valores recebidos no método são os **parâmetros**
- ▶ Os nomes dos argumentos e parâmetros não tem de coincidir mas os tipos de dados **SIM!**

```
static void Main(string[ ] args)
{
    int num1, num2, res;
    Console.Write("1º número:");
    num1 = Convert.ToInt16(Console.ReadLine());
    Console.Write("2º número:");
    num2 = Convert.ToInt16(Console.ReadLine());
    res = dividir(num1, num2);
    Console.WriteLine("Resultado: {0}", res);
}
```

```
public int dividir(int num1, int num2)
{
    int resultado = 0;
    if (num2 != 0)
        resultado = num1 / num2;
    return resultado;
}
```



▶ Parâmetros(2)

- ▶ Um argumento pode ser passado por **valor** ou por **referência**
- ▶ Por padrão, quando um valor é passado para um método, uma cópia é passada em vez de o próprio objeto. Por conseguinte, as alterações ao argumento não têm efeito sobre a cópia

```
classe Program {  
    static void Main(string[] args) {  
        int n = 4;  
        Console.WriteLine("O valor de n é {0}", n); //Output: 4  
        cubo(n);  
        Console.WriteLine("O valor de n é {0}", n); //Output: 4  
    }  
    static void cubo(int n)  
    {  
        n = n*n*n;  
    }  
}
```

Passagem de um argumento por valor

```
classe Program {  
    static void Main(string[] args) {  
        int n = 4;  
        Console.WriteLine("O valor de n é {0}", n); //Output: 4  
        cubo(ref n);  
        Console.WriteLine("O valor de n é {0}", n); //Output: 64  
    }  
    static void cubo(ref int n)  
    {  
        n = n*n*n;  
    }  
}
```

Passagem de um argumento por referência

- ▶ Boas práticas
 - ▶ Operações recorrentes e contexto alargado de dados implicam a necessidade de criação de métodos e variáveis globais
 - ▶ Criação de classe para métodos e variáveis

```
class Program
{
    static void Main(string[] args) {
        //corpo da aplicação
        Console.WriteLine("Nível {0}", GlobalVars.nivel);
        Utils.escreveTabuleiro();
        ...
        Utils.escreveTabuleiro();
    }
}
```

```
class GlobalVars
{
    public static string [,] tabuleiro = new String[3,3];
    public static int nivel;
    ...
}
```

```
class Utils
{
    public void escreveTabuleiro ()
    {
        for(int i=0;i<GlobalVars.tabuleiro.GetLength(0);i++) {
            for(int j=0;j<GlobalVars.tabuleiro.GetLength(1);j++) {
                Console.WriteLine("{0} ", GlobalVars.tabuleiro[i,j]);
                Console.Write("\n");
            }
        }
        ...
    }
}
```

► Variáveis Globais

Classe de variáveis globais

Invocar uma variável global

```
namespace ConsoleApplication1
{
    class Program
    {
        public class GlobalVars
        {
            public static int numero;
        }

        static void Main(string[] args)
        {
            // guardar dados em ficheiro
            gravar_ficheiro();
        }

        static void gravar_ficheiro()
        {
            GlobalVars.numero= 0;
            // nome do ficheiro a gravar

            string ficheiro = "dados.txt";
            // declara streamwriter - de escrita no ficheiro
        }
    }
}
```

```
static void gravar_ficheiro()
{
    GlobalVars.numero= 0;
    // nome do
    string fich
    // declara
```

Equals	
numero	int GlobalVars.numero
ReferenceEquals	crita no ficheiro