Avelar Brotero
ESCOLA SECUNDÁRIA - COIMBRA

Programação Estruturada

Curso Profissional de Técnico de Gestão e Programação de Sistemas Informáticos

Prof.: Jorge Sousa

EQAVET European Quality Assurance in Vocational Education and Training
SELO DE CONFORMIDADE EQAVET Certificado nº 189/2020
ANQEP AGÊNCIA NACIONAL PARA A QUALIFICAÇÃO E O ENSINO PROFISSIONAL I.P.
Cofinanciado por: POCH PORTUGAL 2020 UNIÃO EUROPEIA Fundo Social Europeu

Avelar Brotero
ESCOLA SECUNDÁRIA - COIMBRA

Programação e Sistemas de Informação

Módulo 3 – Programação Estruturada

EQAVET European Quality Assurance in Vocational Education and Training
SELO DE CONFORMIDADE EQAVET Certificado nº 189/2020
ANQEP AGÊNCIA NACIONAL PARA A QUALIFICAÇÃO E O ENSINO PROFISSIONAL, I.P.
Cofinanciado por: POCH
PORTUGAL 2020
UNIÃO EUROPEIA Fundo Social Europeu

Método

- Um método (que, em programação, pode assumir o nome de subprograma, procedimento ou função) é, simplesmente, um conjunto de instruções (ou código) que constitui uma unidade com uma certa autonomia dentro de um algoritmo (ou programa) principal.
- Os métodos (e os subprogramas a que eles dão origem) constituem, a par das estruturas de controlo, as bases da programação estruturada.
- Os métodos ou **subprogramas podem ajudar a uma melhor estruturação de um algoritmo ou programa**.
- Os métodos ou subprogramas **podem**, se necessário, **ser utilizados mais do que uma vez em diferentes pontos do algoritmo ou programa**.

Vantagens da estruturação de algoritmos em métodos

- Os métodos ou subprogramas **podem ser concebidos** de forma flexível, **de modo a permitirem a sua utilização em diferentes contextos ou com valores diferenciados**
- Os métodos ou subprogramas **podem ser implementados com uma certa autonomia uns dos outros e em relação ao algoritmo ou programa principal – permitindo, assim, que projetos de maiores dimensões sejam divididos por diferentes membros ou equipas de trabalho.**

Avelar Brotero
ESCOLA SECUNDÁRIA - COIMBRA

Conceitos básicos de funções em C#

- Em C# o método **Main()** é o método(função) principal e indispensável de um programa.
- Para aceder a métodos predefinidos (contidos em Bibliotecas) utilizamos a diretiva **#include**, seguida do nome da biblioteca que nos interessar.
- **O programador pode criar os seus próprios métodos.**

  Os métodos são unidades de código fundamentais com que se escrevem e estruturam os programas.

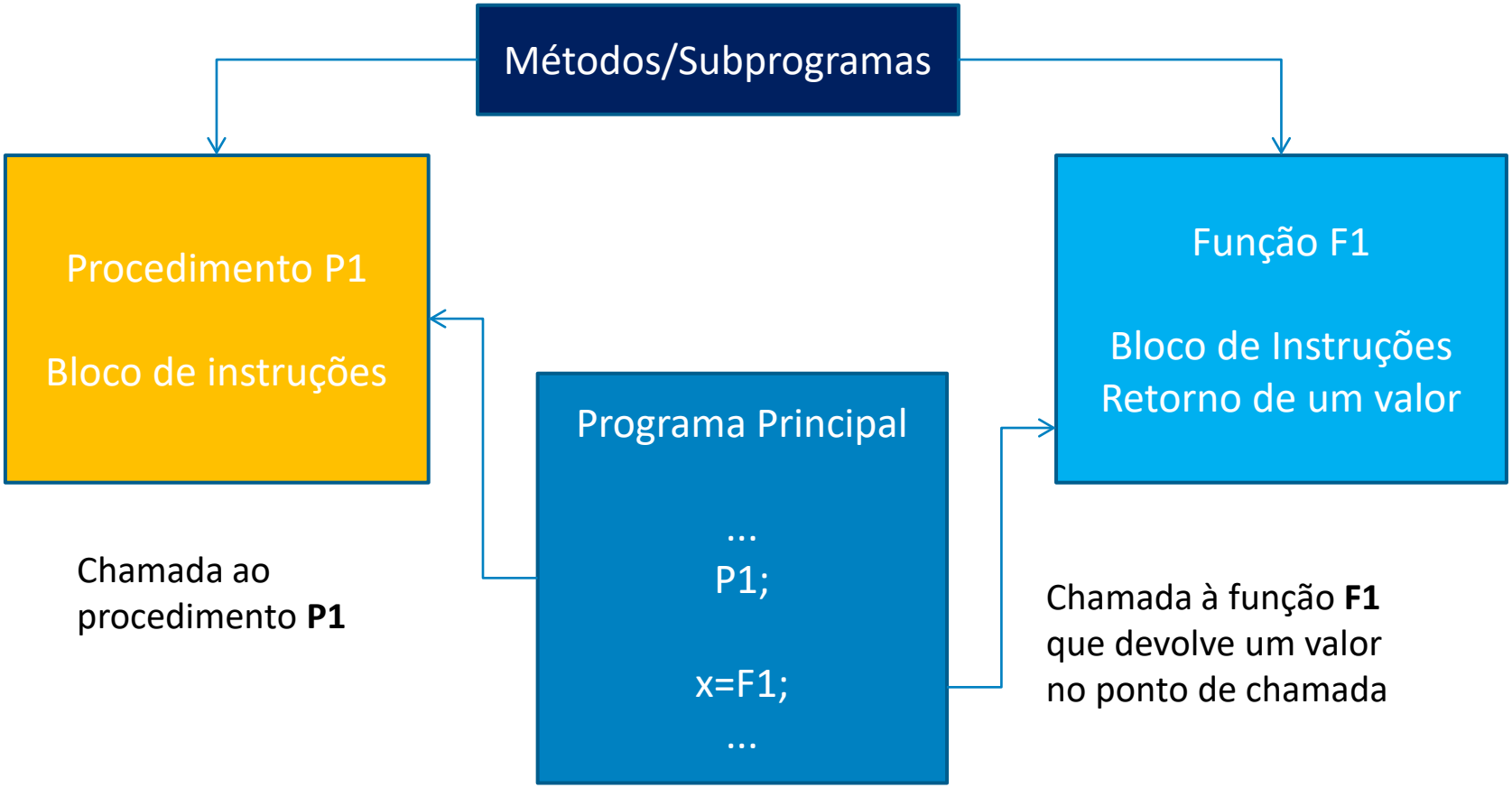
- Os **métodos** têm de ser sempre definidos com a indicação do tipo de dados que é suposto eles denvolverem ao serem chamadas. Assim, podemos ter métodos do tipo **int**, **char**, **float**, **double**, **void** (quando não devolve nenhum valor), etc.

Funções e Procedimentos

- Noutras linguagens de programação, como Pascal, Visual Basic, etc., as unidades de código fundamentais com que se estruturam os programas podem ser de dois tipos: procedimentos (procedures) e funções (functions).
- Um procedimento (procedure em Pascal, sub em Visual Basic) é uma unidade de código ou rotina que ao ser chamada, executa uma determinada sequência de instruções.
- **Uma função** (function em Pascal e em Visual Basic) é também uma unidade de código ou rotina que, ao ser chamada, executa uma sequência de instruções, mas, para além disso, **devolve um valor no ponto em que for chamada**.

Avelar Brotero
ESCOLA SECUNDÁRIA - COIMBRA

Funções e Procedimentos: Representação gráfica



Métodos em C#

- **Em C#, não existe** à partida, essa **diferença entre procedimentos e funções**; nem sequer se usa nenhuma dessas designações na escrita do código. Nesta linguagem, **um método reconhece-se apenas pelos parênteses que se seguem ao nome (identificador) do método**, como por exemplo em **Main()**, rand(), gets(), strcpy(), etc.

Sintaxe:

```
static tipo_retorno nome_método([parâmetros]){
        <corpo do método >
}
```

Avelar Brotero
ESCOLA SECUNDÁRIA - COIMBRA

Variáveis globais e variáveis locais em C#

- **Variavel global (ou extern):** quando é declarada antes de qualquer método; neste caso é utilizável em qualquer parte do programa.
- **Variável local (ou auto):** quando é declarada dentro de um método ou de um bloco de código; neste caso, só é utilizável nessa parte do programa.

```
class Program
{
    public static double soma;

    public static void Main(string[] args)
    {
        int quantos;
        soma=20;
        quantos=5;
        Console.WriteLine(med(quantos));

        Console.ReadKey(true);
    }
    static double med(int n) {
        double m;
        m=soma/n;
        return m;
    }

}
```

soma de uma variável global – pode ser usada em todo o programa

quantos é uma variável local – só pode ser usada na função **Main**

m é uma variável local – só pode ser usada na função **med**

- **Variáveis globais e variáveis locais em C#**
  - **Escopo das variáveis** está relacionado com a visibilidade e duração das variáveis.
    - A visibilidade de uma variável refere-se às partes do programa em que está acessível e pode ser utilizada.
    - A duração de uma variável tem a ver com a parcela de tempo em que a variável existe no decurso do programa.
  - Em C#, uma variável é **global** ou **extern** quando é declarada fora de qualquer método (geralmente, no topo do programa, antes de qualquer método). Nesse caso, a **variável é visíbel em todo o programa**, podendo ser utilizada em qualquer ponto desse programa. Além disso, a **duração da variável é equivalente à do programa**. Ela é criada em memória quando o programa começa a correr e só é extinta quando programa termina.

Variáveis globais e variáveis locais em C#

- Em C#, uma variável é **local** ou **auto** quando é declarada dentro de uma função ou de um bloco de código. Neste caso, a **variável só é visível e utilizável no bloco de código em que foi declarada**. Além disso, a **variável local só existe enquanto a função a que pertence está a ser executada**.

Avelar Brotero
ESCOLA SECUNDÁRIA - COIMBRA

Conceitos básicos de funções em C#

//Programa para calcular a área de um
//círculo dada a medida do seu raio

Versão1

```
class Program
{
    public static void Main(string[] args)
    {
        int raio;
        double area;
        Console.Write("Digite a medida do raio: ");
        raio=Convert.ToInt32(Console.ReadLine());
        area=3.14*raio*raio;
        Console.WriteLine("Área= "+area);

        Console.ReadKey();
    }
}
```

Versão2 (com funções e variável global)

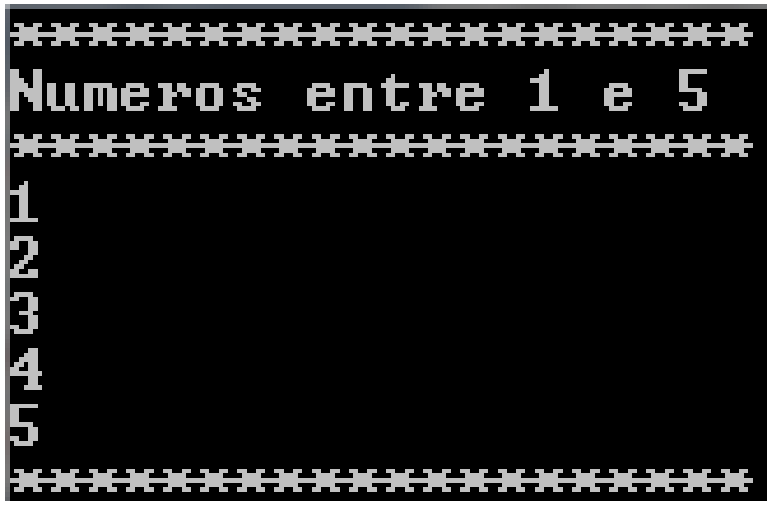
```
class Program
{
        static int raio; //variável global

    public static void Main(string[] args)
    {
        dados(); //chamada da função dados
        escrever(); //chamada da função escrever
        Console.ReadKey(true);
    }
    static void dados(){
        Console.Write("Introduza a mediada do raio: ");
        raio=Convert.ToInt32(Console.ReadLine());
    }

    static double calculo(){
        return Math.PI*raio*raio;
    }

    static void escrever(){
        Console.WriteLine("Área= "+calculo());
    }
}
```

- **Conceitos básicos de métodos em C#**
  - **Exercício1**
    - Escreva um programa que coloque no ecrã o seguinte output, escrevendo a linha de 20 asteriscos através do ciclo for.



Avelar Brotero
ESCOLA SECUNDÁRIA - COIMBRA

- **Estrutura geral de um método**

static **tipo_de_dados nome_da_função([parâmetros])**
{
<declarações e intruções da função>
[return [expressão];]
}

- **Exemplos:**

```
static void obter_raio()
{
    Console.Write("Digite a medida do raio: ");
    raio=Convert.ToInt32(Console.ReadLine());
}
```

```
static double calcular_area(int r)
{
    return Math.PI*r*r;
}
```

Passagem de argumentos a parâmetros por valor

- Quando um método que utiliza um ou mais parâmetros é chamado, é necessário indicar argumentos no lugar desses parâmetros.
- Considerando o seguinte protótipo de um método:

```
static double calculo(int r)
```

- As chamadas ao método podem utilizar, no lugar do parâmetro r, um valor direto, como na seguinte expressão:

```
Console.WriteLine("Área= "+ calculo(2));
```

- Também podem utilizar uma variável, como em:

```
Console.WriteLine("Área= "+ calculo(raio));
```

Se os argumentos usados são variáveis, não são as próprias variáveis que passam para dentro do método, mas apenas os seus valores.

Passagem de argumentos a parâmetros por valor - Exemplo

- O programa a seguir apresentado utiliza um método **troca()** com **dois parâmetros** (**x** e **y**).
- Dentro do método **troca()**, **os valores dos dois parâmetros são trocados um com o outro**.
- No programa, são também declaradas duas variáveis: **n1** e **n2**. Estas variáveis são **inicializadas** com os valores **1** e **2**, respetivamente.

Avelar Brotero
ESCOLA SECUNDÁRIA - COIMBRA

- **Passagem de argumentos a parâmetros por valor - Exemplo**

```
class Program
{
    public static void Main(string[] args)
    {

        int n1=1, n2=2;
        troca(n1,n2); //chamada da função troca

        Console.WriteLine("n1="+n1+"\nn2="+n2);
        Console.ReadKey(true);
    }
    static void troca(int  x,  int  y){
        int aux;
        aux=x;
        x=y;
        y=aux;
        Console.WriteLine("x="+x+"\ny="+y);
    }
}
```

- **Passagem de parâmetros por valor - Exemplo**
  - **Quando o método troca() é chamado**, na instrução **troca(n1,n2)**, **para o método passam apenas os valores dos argumentos n1 e n2**. O **parâmetro x recebe o valor 1 e o parâmetro y o valor 2**.
  - Dentro do método, **após as seguintes instruções**:

```
aux=x;
 x=y;
y=aux;
```

  - **As variáveis** (parâmetros) **x e y ficam com os seus valores trocados**. Porém, quando regressamos à **função Main() as variáveis n1 e n2 continuam com os seus valores inalterados**.

Passagem de parâmetros por referência (ref , out)

- Quando chamamos um método que utiliza parâmetros, **podemos querer passar**, **através dos argumentos**, **não apenas os seus valores**, **mas as próprias variáveis utilizadas como argumentos**. É a chamada **passagem por referência.**
- Neste caso, **as alterações que forem efetuadas nas variáveis passadas como argumentos para o método em causa**, **não acontecem apenas dentro do método, mas também nas próprias variáveis** (que têm a sua existência fora do método).

Avelar Brotero
ESCOLA SECUNDÁRIA - COIMBRA

- **Passagem de parâmetros por referência - Exemplo**

```
class Program
{
    public static void Main(string[] args)
    {

        int n1=1, n2=2;
        troca(ref n1,ref n2); //chamada da função troca

        Console.WriteLine("n1="+n1+"\nn2="+n2);
        Console.ReadKey(true);
    }
    static void troca(ref int  x, ref int  y){
        int aux;
        aux=x;
        x=y;
        y=aux;
        Console.WriteLine("x="+x+"\ny="+y);
    }
}
```

- **Passagem de argumentos a parâmetros por referência (ref) Exemplo**
  - Nesta versão do programa, o método **troca** é escrito com o seguinte **protótipo e cabeçalho**:

```
static void troca(ref int  x, ref int  y)
```

  - Isto quer dizer que **os parâmetros x e y do método troca funcionam como outros nomes para as variáveis que forem passadas como argumentos nas chamadas ao método troca.**
  - Assim sendo, **quando**, na função **Main(), é feita a chamada** `troca(ref n1,ref n2);`

    são as próprias variáveis n1 e n2 que passam para o método.
  - Dentro do método troca, as instruções utilizam os parâmetros x e y sem necessidade de qualquer outra indicação, pois x e y funcionam como outros nomes das variáveis que forem passadas ao método como argumentos.

Programação Estruturada

- **Passagem de argumentos a parâmetros por referência (out) Exemplo**

```
public static void Main(string[] args)
{
    double raio;
    ler_dados(out raio);
    Console.WriteLine("A área da circunferência é {0:0.0}",areas(raio));

    Console.ReadKey(true);
}
static void ler_dados(out double r){
    Console.Write("Insira o valor do raio da circunferência: ");
    r=double.Parse(Console.ReadLine());
}

static double areas(double r){
     return Math.PI*r*r;
}
```

Permite sair o valor lido e guardado na variável real r

Recursividade

- Um método recursivo é aquele que faz uma chamada a si próprio nas suas instruções
- Um método recursivo tem duas características fundamentais:
  - permite simplificar a resolução do problema;
  - tem uma condição de finalização (que evita que a recursividade se prolongue indefinidamente - enquanto a condição de finalização não se verificar o método chama-se sempre a si próprio).

- **Recursividade**
  - Função fatorial (resolução iterativa)

```
class Program
{
    public static void Main(string[] args)
    {
    //  Função fatorial (resolução iterativa)

    const int n=4;
        Console.WriteLine(n+"!="+fatorial(n));

        Console.ReadKey(true);
    }
    static  int fatorial(int n){
        int f;

        if(n==0)
            f= 1;
        else {
                f=1;
                for(int i=1;i<=n;i++)
                f=f*i;
            }
        return f;
    }
}
```

- **Recursividade**
  - Função fatorial (resolução recursiva)

```
class Program
{
    public static void Main(string[] args)
    {
    //  Função fatorial (resolução recursiva)

    const int n=4;
        Console.WriteLine(n+"!="+fatorial(n));

        Console.ReadKey(true);
    }
    static  int fatorial(int n){
        int f;

        if(n==0)
            f=  1;
        else
            f= n*fatorial(n-1);
        return f;
    }
}
```

Avelar Brotero
ESCOLA SECUNDÁRIA - COIMBRA

- **Recursividade**

factorial(6)
  6 * factorial(5)
  6 * 5 * factorial(4)
  6 * 5 * 4 * factorial(3)
  6 * 5 * 4 * 3 * factorial(2)
  6 * 5 * 4 * 3 * 2 * factorial(1)
  6 * 5 * 4 * 3 * 2 * 1 * factorial(0)
  6 * 5 * 4 * 3 * 2 * 1 * 1
  6 * 5 * 4 * 3 * 2 * 1
  6 * 5 * 4 * 3 * 2
  6 * 5 * 4 * 6
  6 * 5 * 24
  6 * 120
  720

Avelar Brotero
ESCOLA SECUNDÁRIA - COIMBRA

- **Recursividade**

NOTA: A programação recursiva embora **simplifique o código** dos métodos é, normalmente, **menos eficiente**.

Avelar Brotero
ESCOLA SECUNDÁRIA - COIMBRA

- *Adaptado de:*
  - **Azul**, Artur Augusto, *Bases de Programação 12º Ano*, Porto Editora, 2006
  - **Azul**, Artur Augusto, *Bases de Programação 10º Ano*, Porto Editora, 2004
  - **Azul**, Artur Augusto, *Linguagens de Programação e Programação e Sistemas de Informação, Ensino profissional – nível3*, Porto Editora, 2010