

Cap. 7 - Métodos (Funções e procedimentos)

Aprenda a Programar com C# 2018 — Edições Sílabo
<http://www.silabo.pt/livros.asp?num=606>

António Trigo, Jorge Henriques
{antonio.trigo,jmvhenriques}@gmail.com

17 de Abril de 2018

Introdução

Noção de função e procedimento

Passagem de argumentos

Recursividade

Introdução

- ▶ Uma técnica aconselhada em programação é a divisão do problema inicial em subproblemas mais simples ou, por outras palavras, dividir o programa que permite a resolução computacional do problema em subprogramas;
- ▶ Um subprograma constitui um bloco independente dentro de um programa, podendo ser visto como um programa dentro de outro programa;
- ▶ Tal como um programa necessita de comunicar com o ambiente exterior, através da leitura de dados e da apresentação de resultados, também um subprograma deve ter capacidade de aceitar valores e de, eventualmente, devolver resultados ao ambiente em que se insere, ou seja, o programa ou subprograma que controla a sua execução.

Subprogramas

- ▶ Surgem muitas vezes da necessidade de repetir a mesma sequência de instruções em locais diferentes do programa;
- ▶ A sequência de instruções é agrupada num subprograma, a que é dado um nome e invocado nos locais em que a sequência de instruções seja necessária;
- ▶ Um subprograma é controlado por um programa de nível superior que determina quando e sob que condições o subprograma deve ser executado e que pode eventualmente recolher um valor devolvido por este;
- ▶ Os subprogramas são normalmente classificados como funções ou procedimentos.

Funções

- ▶ Em C#, os subprogramas são implementados por funções;
- ▶ Sintaxe:

```
[static] <tipo_de_dados_de_retorno>  
    <nome_da_funcao>(  
[<lista_de_parametros>]) {  
    declaracoes;  
    ...  
    instrucoes;  
    ...  
    return expressao;  
}
```

Funções

```
static float Quadrado(float num) {  
    return num*num;  
}  
  
static void Main(string[] args)  
{  
    const float pi = 3.1416F;  
    float r, alt;  
    Console.WriteLine("Calculo da area e volume do  
        cilindro:");  
    Console.Write("Raio?");  
    r = Convert.ToSingle(Console.ReadLine());  
    Console.Write("Altura?");  
    alt = Convert.ToSingle(Console.ReadLine());  
    Console.WriteLine("area: {0}",  
        pi*Quadrado(r)+2*pi*r*alt);  
    Console.WriteLine("volume: {0}", pi*Quadrado(r) *  
        alt);  
}
```

Procedimentos

- ▶ Um procedimento é uma função que não retorna qualquer valor e é chamado sozinho, sem estar envolvido em qualquer operação;
- ▶ A linguagem C# usa um tipo especial - *void* - como tipo de retorno de uma função que não retorna qualquer valor.

Procedimentos

```
static void IntFrac(float x)
{
    Console.WriteLine("Parte inteira: {0} \n", (int)x);
    Console.WriteLine("Parte fracionaria: {0:F3}\n", x
        - (int)x);
}

static void Main(string[] args)
{
    float num;
    Console.WriteLine("Escreva um numero real: ");
    num = Convert.ToSingle(Console.ReadLine());
    IntFrac(num);
}
```


Parâmetros / Argumentos

- ▶ Muitas vezes é necessário indicar explicitamente ao subprograma as condições sob as quais a sua execução vai ser realizada;
- ▶ Por estas razões, um subprograma poderá aceitar argumentos que determinam essas condições;
- ▶ Os argumentos poderão ser variáveis, constantes ou expressões;
- ▶ Os argumentos podem ser passados por valor (tipos primitivos) ou por referência (ex.: vetores);
- ▶ Na descrição do subprograma terão de estar definidos, por sua vez, os parâmetros que correspondem aos argumentos que o programa do nível acima determinou;
- ▶ Estas considerações são válidas quer para subprogramas definidos pelos programadores quer para subprogramas pré-definidos.

Parâmetros / Argumentos - Passagem por valor

```
static float Area(float x, float y)
{
    return x * y;
}

static void Main(string[] args)
{
    float comp, larg;
    Console.WriteLine("Introduza os dados do
        retangulo: ");
    Console.Write("Comprimento: ");
    comp = Convert.ToSingle(Console.ReadLine());
    Console.Write("Largura: ");
    larg = Convert.ToSingle(Console.ReadLine());
    Console.WriteLine("A area do retangulo e:
        {0}", Area(comp, larg));
}
```

Parâmetros / Argumentos - Valores por defeito e por nome

```
static float Area(float x = 5, float y = 9){  
    return x * y;  
}  
  
static void Main(string[] args){  
    float comp, larg;  
    Console.WriteLine("Introduza os dados do  
        retangulo: ");  
    Console.Write("Comprimento: ");  
    comp = Convert.ToSingle(Console.ReadLine());  
    Console.Write("Largura: ");  
    larg = Convert.ToSingle(Console.ReadLine());  
    Console.WriteLine("A area do retangulo e:  
        {0}", Area(comp, larg));  
    Console.WriteLine("A area do retangulo e:  
        {0}", Area());  
    Console.WriteLine("A area do retangulo e:  
        {0}", Area(y:8));  
}
```

Parâmetros / Argumentos - Passagem por referência

Quando se deseja passar uma variável por referência e não por valor usa-se a palavra reservada **ref**

```
static void Metodo(ref int i){  
    i = i + 44;  
}  
static void Main(string[] args){  
    int val = 1;  
    Metodo(ref val);  
    Console.WriteLine(val);  
    // Output: 45  
}
```

Overload

O Overload é a existência de funções (métodos) com assinaturas diferentes, seja no tipo e número de parâmetros seja no tipo de dados retornado.

```
static float Soma(float x, float y){  
    return x * y;  
}  
static int Soma(int x, int y){  
    return x * y;  
}  
static void Main(string[] args){  
    float a = 2.5F, b = 3F;  
    int c = 4, d = 5;  
    Console.WriteLine("{0:F2}+{1:F2}={2:F2}",  
        a,b,Soma(a,b));  
    Console.WriteLine("{0}+{1}={2}", c,d,Soma(c,d));  
}
```

Variáveis locais e “globais”

- ▶ São variáveis declaradas dentro do corpo de cada função;
- ▶ Só têm validade dentro da função em que são declaradas;
- ▶ As variáveis declaradas dentro da função `Main()` só existem dentro desta.
- ▶ No C# não existe o conceito de variável “global” pois o programa é composto por um conjunto de classes;
- ▶ O mais próximo que temos de uma variável global é a declaração de uma variável (atributo) logo a seguir à declaração da classe, antes da declaração de qualquer função.
- ▶ Quando uma função tem uma variável local com o mesmo nome de uma variável “global” a função dá preferência à variável local;

Recursividade

- ▶ Alguns subprogramas apresentam uma característica interessante que consiste na possibilidade de se chamarem a si próprios, ou seja, um subprograma é invocado recursivamente pelo mesmo subprograma.
- ▶ O cálculo do fatorial de um número inteiro é um caso típico de um algoritmo recursivo.
- ▶ O fatorial de n , também representado por $n!$, em que n é um inteiro não negativo, é calculado pela expressão:
$$n! = n \times (n - 1) \times (n - 2) \times \dots \times 1$$

Exemplo: $5! = 5 \times 4 \times 3 \times 2 \times 1$
- ▶ Mas também pode ser calculado de forma recursiva:
$$n! = n \times (n - 1)!$$

Exemplo: $5! = 5 \times 4!$
- ▶ A função recursiva para o cálculo do fatorial pode ser escrita tendo em consideração que o fatorial de 0 é 1 ($0! = 1$).

Recursividade

```
static decimal Fatorial(int n) {  
    if (n == 0) {  
        return 1;  
    }  
    else {  
        return n * Fatorial(n - 1);  
    }  
}  
  
static void Main(string[] args){  
    Console.Write("n = ");  
    int n = Convert.ToInt32(Console.ReadLine());  
    decimal fatorial = Fatorial(n);  
    Console.WriteLine("{0}! = {1}", n, fatorial);  
}
```