Sub-Rotinas

Um matemático uma vez disse que um grande problema se resolve dividindo-o em pequenas partes e resolvendo tais partes em separado. Estes dizeres servem também para a construção de programas. Os profissionais de informática quando necessitam construir um grande sistema, o fazem, dividindo tal programa em partes, sendo então desenvolvido cada parte em separado, mais tarde, tais partes serão acopladas para formar o sistema. Estas partes são conhecidas por vários nomes. Nós adotaremos uma destas nomenclaturas: sub-rotinas.

Podemos dar um conceito simples de sub-rotina dizendo ser um pedaço de código computacional que executa uma Função bem definida, sendo que esta sub-rotina pode ser utilizada várias vezes no programa.

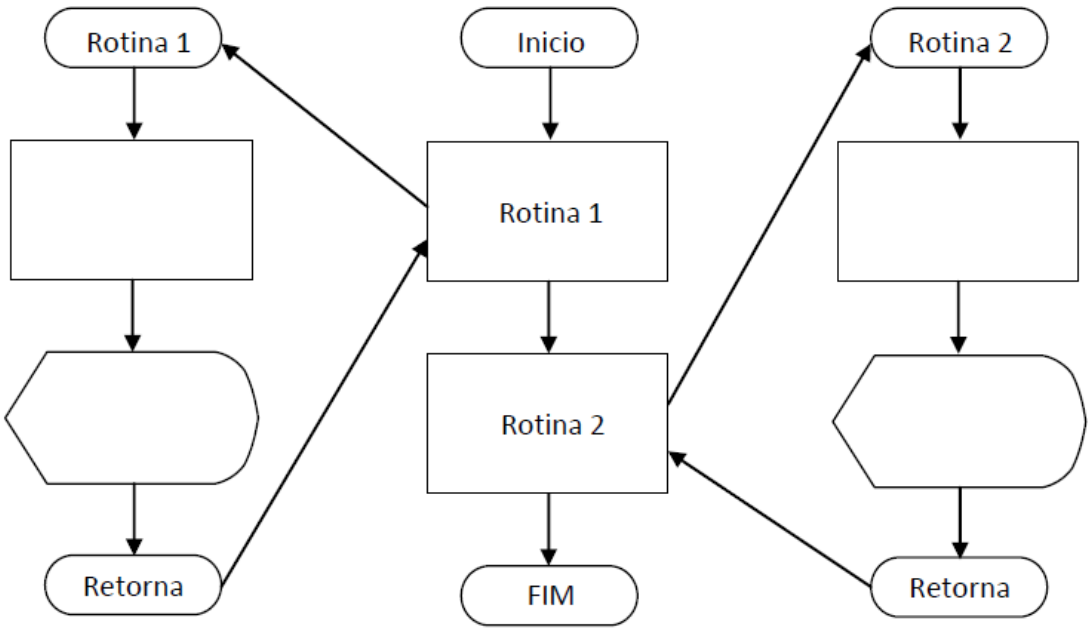
São importantes na:

Subdivisão de algoritmos complexos, facilitando o seu entendimento;

Estruturação de algoritmos, facilitando, principalmente, a detecção de erros e a documentação de sistemas;

Modularização de sistemas que facilita a manutenção de software e a reutilização de subalgoritmos já implementados.

Garante maior legibilidade



Existem **2 formas** de implementação e uso de sub-rotinas:

• **PROCEDIMENTOS**: que não retorna valores ao algoritmo chamador;

• **FUNÇÕES**: que retorna um e, somente um, valor ao algoritmo chamador.

PROCEDIMENTOS (procedures)

É um tipo de subalgoritmo que se assemelha muito com um algoritmo e tem como objetivo executar uma ação que não irá devolver valores ao (sub)algoritmo chamador.
É ativada através da colocação de seu nome em alguma parte do programa ou de outra sub-rotina. A chamada de procedimentos só é feita em comandos isolados dentro de um algoritmo através do seu nome e dos respectivos **parâmetros**.
Desta forma, assim que o nome de um PROCEDIMENTO é encontrado, ocorre um desvio no programa, para que os comandos da sub-rotina sejam executados. Ao término da sub-rotina, a execução retornará ao ponto subseqüente a da chamada do PROCEDIMENTO.

FUNÇÕES (Functions)

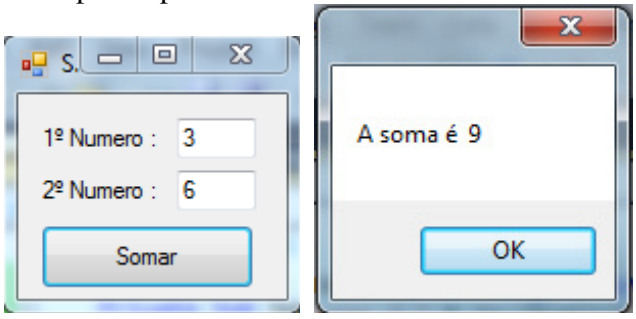
Uma sub-rotina do tipo **FUNÇÃO** possui as mesmas características de um PROCEDIMENTO no que se refere a passagem de parâmetros, variáveis globais e locais, mas possui uma importante diferença, que é o **retorno de um valor ao término de sua execução**, ou seja, uma **FUNÇÃO** sempre deverá **retornar um valor** ao chamador. O conceito de FUNÇÃO é originário da idéia de função matemática (por exemplo, raiz quadrada, seno, logaritmo, entre outras), onde um valor é calculado a partir de outro(s) fornecido(s) à função.
Na **definição** de uma FUNÇÃO deverá ser informado qual o **tipo do valor retornado**.

Aplicação de Procedimentos

Procedimentos são chamados como instruções isoladas.
O objetivo de se declarar um procedimento é associá-lo a um identificador para que o mesmo possa ser ativado por um comando do programa.
A chamada ou ativação de um procedimento é feita referenciando-se o seu nome no local do programa, onde o mesmo deve ser ativado.
Ao terminar a execução dos comandos de um procedimento, a seqüência do programa retorna sempre a instrução que provocou a sua chamada.
Definição de Procedimentos em Visual Basic:

<Idenficador de Escopo> Sub <Nome do Procedimento>(<Parâmetros>)
<Comandos>
End Sub

Exemplo de procedimento



Calculo da soma de dois Números

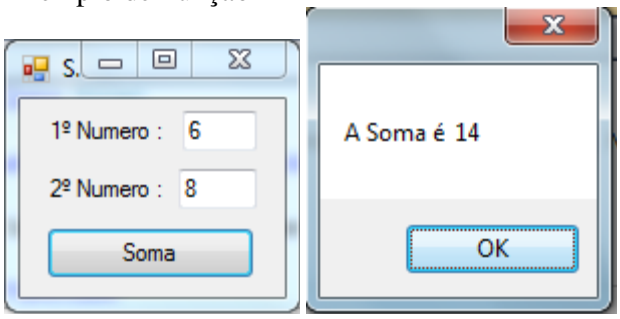
```
Public Class Form1
        Private Sub Soma(ByRef Numero1 As Integer, ByVal Numero2 As Integer)
                MessageBox.Show("A soma é " & Str(Numero1 + Numero2))
        End Sub
        Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
        System.EventArgs) Handles Button1.Click
                Dim N1 As Integer
                Dim N2 As Integer
                N1 = Val(Me.TextBox1.Text)
                N2 = Val(Me.TextBox2.Text)
                Soma(N1, N2)
        End Sub
End Class
```

Aplicação de Funções

Distinguem-se das procedures pela característica de retornarem um valor. A maioria das linguagens de programação possuem algumas funções pré-definidas, facilitando o trabalho dos programadores.
O objetivo de se declarar uma função é associá-la a um identificador para que o mesmo possa ser ativado por um comando do programa.
Definição de Funções em Visual Basic:

```
<Idenficador de Escopo> Function <Nome da Função>(<Parametros>)
<Comandos>
End Function
```

Exemplo de Função



Calculo da soma de dois Números

```
Public Class Form1
        Private Function Soma(ByVal Numero1 As Integer, ByVal Numero2 As Integer)
                Soma = Numero1 + Numero2
        End Function
        Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
        System.EventArgs) Handles Button1.Click
                Dim N1 As Integer
                Dim N2 As Integer
```

```
        Dim Somatoria As Integer
        N1 = Me.TextBox1.Text
        N2 = Me.TextBox2.Text
        Somatoria = Soma(N1, N2)
        MessageBox.Show("A Soma é " & Str(Somatoria))
    End Sub
End Class
```

PASSAGEM DE PARAMETROS

Parâmetros têm por finalidade servir como um ponto de comunicação bidirecional entre uma sub-rotina e o programa principal, ou com outra sub-rotina hierarquicamente de nível mais alto. Desta forma, é possível passar valores de uma sub-rotina ou rotina chamadora à outra sub-rotina e vice-versa.
A especificação dos parâmetros consiste na declaração dos tipos das variáveis que compõem a lista dos parâmetros.
Existem dois tipos de passagem de parâmetros utilizados em linguagens de programação:

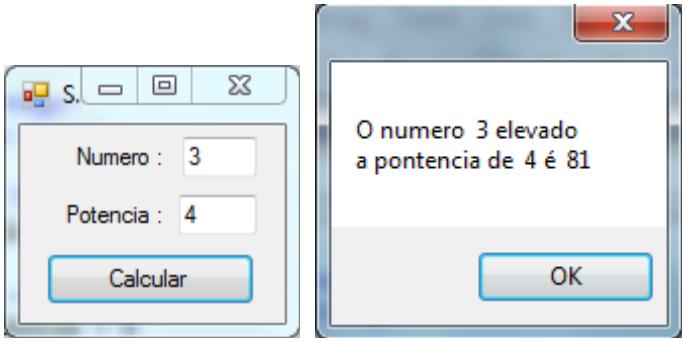
- Passagem por Valor
- Passagem pó Referencia

Passagem de parâmetros por Valor – ByVal:

Este é o método padrão, ou seja, ao declarar os argumentos de uma função / procedimento, não é preciso usar a palabra ByVal, pois automaticamente, os argumentos assumem a opção ByVal. As duas declarações a seguir são equivalentes:

```
Public Sub DobraValor(Num As Integer)
Ou
Public Sub DobraValor(ByVal Num As Integer)
```

Na segunda declaração, explicitamente, estou utilizando a opção ByVal, para indicar que o parâmetro Num será passado por valor. **Mas o que significa a passagem de um parâmetro por valor??**
Significa que o procedimento receberá apenas o valor do parâmetro, uma copia da informação passada e não uma referência ao endereço de memória onde está armazenado o valor do parâmetro. Com isso, quaisquer alterações que sejam feitas no valor do parâmetro, dentro do procedimento, não afetarão o valor original, o qual será o mesmo de antes da chamada da função/procedimento. Em resumo, apenas o valor é passado para a função/procedimento, este valor é utilizado pelo código da função/procedimento, sem afetar o valor original do parâmetro.
Exemplo de passagem de parâmetros por valor:



Calculo da Potencia de um numero

```
Public Class Form1
    Private Function Potencia(ByVal Num As Integer, ByVal Exp As Integer)
        Dim K As Integer
        Dim Base As Integer
        If Exp = 0 Then
            Potencia = 1
        Else
            If Num = 0 Then
                Potencia = 0
            Else
                Base = Num
                For K = 1 To Exp - 1
                    Num = Num * Base
                Next
                Potencia = Num
            End If
        End If
    End Function
    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
    System.EventArgs) Handles Button1.Click
        Dim Valor As Integer
        Dim Expoente As Integer
        Dim Resultado As Integer
        Valor = Me.TextBox1.Text
        Expoente = Me.TextBox2.Text
        Resultado = Potencia(Valor, Expoente)
        MessageBox.Show("O numero " & Str(Valor) & " elevado " + vbCrLf +
        "a pontencia de " & Str(Expoente) & " é " & Str(Resultado))
    End Sub
End Class
```

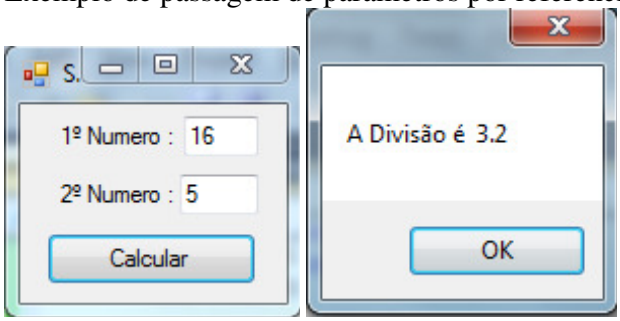
Passagem de parâmetros por Referência – ByRef:

Para poder atualizar/alterar o valor original, a função/procedimento, tem que receber o parâmetro por referência – ByRef, ou seja, a função/procedimento tem que receber uma referência ao endereço de memória da variável passada como parâmetro e não uma simples cópia do valor da variável (que é o que acontece na passagem ByVal).

Ao receber um parâmetro por referência (ByRef), as alterações que a função/procedimento fizer, serão feitas diretamente na variável original, pois agora, a função/procedimento tem acesso ao endereço da variável na memória e não mais apenas uma cópia do seu valor. Para que um procedimento possa receber um parâmetro por referência, você deve utilizar a palavra ByRef, conforme o exemplo a seguir:

```
Public Sub DobraValor(ByRef Num As Integer)
```

Exemplo de passagem de parâmetros por referência:



Efetuar as quadro operações básicas entre dois números

```
Public Class Form1
    Private Sub Calculo(ByVal Numero1 As Integer, ByVal Numero2 As Integer,
    ByVal Operacao As String, ByRef Resultado As Single)
        Select Case Operacao
        Case "+"
            Resultado = Numero1 + Numero2
        Case "-"
            Resultado = Numero1 - Numero2
        Case "*"
            Resultado = Numero1 * Numero2
        Case "/"
            Resultado = Numero1 / Numero2
        Case Else
            Resultado = 0
            MessageBox.Show("Não é uma Operação valida")
        End Select
    End Sub
    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
    System.EventArgs) Handles Button1.Click
        Dim Valor1 As Integer
        Dim Valor2 As Integer
        Dim Resultado As Single
        Valor1 = Me.TextBox1.Text
        Valor2 = Me.TextBox2.Text
        Calculo(Valor1, Valor2, "+", Resultado)
        MessageBox.Show("A Soma é " & Str(Resultado))
        Calculo(Valor1, Valor2, "-", Resultado)
        MessageBox.Show("A Subtração é " & Str(Resultado))
        Calculo(Valor1, Valor2, "*", Resultado)
        MessageBox.Show("A Multiplicação é " & Str(Resultado))
        Calculo(Valor1, Valor2, "/", Resultado)
```

```
            MessageBox.Show("A Divisão é " & Str(Resultado))
        End Sub
End Class
```