2013

Decode Team

Instituto Politécnico de tomar

Portugol

Equivalências de estruturas entre Portugol e Visual Basic

Índice

[Nota Geral: 3](#_Toc360551488)

[Algumas notas sobre Visual Basic: 3](#_Toc360551489)

[Estrutura Início 3](#_Toc360551490)

[Início: 3](#_Toc360551491)

[Estrutura Fim 4](#_Toc360551492)

[Fim: 4](#_Toc360551493)

[Variáveis 4](#_Toc360551494)

[Equivalência entre TIPOS de variáveis 4](#_Toc360551495)

[Definição e atribuição de variáveis 4](#_Toc360551496)

[Se a variável não estiver definida em memória 4](#_Toc360551497)

[Se a variável estiver definida em memória 4](#_Toc360551498)

[Alguns exemplos de definição e atribuição de variáveis 5](#_Toc360551499)

[Estruturas input/output 7](#_Toc360551500)

[Input – Ler 7](#_Toc360551501)

[Se a variável não estiver definida em memória 7](#_Toc360551502)

[Se a variável já estiver definida em memória 7](#_Toc360551503)

[Output – Escrever 8](#_Toc360551504)

[Estruturas de Decisão 8](#_Toc360551505)

[Condição *“if”* e *“if else”* 8](#_Toc360551506)

[Exemplos práticos 9](#_Toc360551507)

[Condição *“while”* 9](#_Toc360551508)

[Condição *“do while”* 10](#_Toc360551509)

[Exemplos práticos 10](#_Toc360551510)

[Estrutura Conector 11](#_Toc360551511)

[Conector 11](#_Toc360551512)

[Funções 12](#_Toc360551513)

[Definir funções 12](#_Toc360551514)

[Definir função *Exemplo* sem parâmetros de entrada 12](#_Toc360551515)

[Definir função *Exemplo* com parâmetros de entrada 12](#_Toc360551516)

[Chamada de funções 13](#_Toc360551517)

[Exemplos do uso de funções 13](#_Toc360551518)

[Estrutura de retorno 13](#_Toc360551519)

[Return 13](#_Toc360551520)

[Operadores 14](#_Toc360551521)

[Aritméticos 14](#_Toc360551522)

[Lógicos 14](#_Toc360551523)

[Relacionais 14](#_Toc360551524)

[ANEXO 15](#_Toc360551525)

[Algoritmo com o uso da condição “if” 15](#_Toc360551526)

[Fluxograma 15](#_Toc360551527)

[Código 15](#_Toc360551528)

[Esquema detalhado 16](#_Toc360551529)

[Algoritmo com o uso da condição “if else” 17](#_Toc360551530)

[Fluxograma 17](#_Toc360551531)

[Código: 17](#_Toc360551532)

[Esquema detalhado 18](#_Toc360551533)

[Algoritmo com o uso da condição “while” 19](#_Toc360551534)

[Fluxograma 19](#_Toc360551535)

[Código 19](#_Toc360551536)

[Esquema detalhado 20](#_Toc360551537)

[Algoritmo com o uso da condição “ do while” 21](#_Toc360551538)

[Fluxograma 21](#_Toc360551539)

[Código 21](#_Toc360551540)

[Esquema detalhado 22](#_Toc360551541)

[Algoritmo com o uso de uma função 23](#_Toc360551542)

[Fluxogramas 23](#_Toc360551543)

[Código 23](#_Toc360551544)

[Esquema detalhado 24](#_Toc360551545)

# Nota Geral:

Devido à especificação da linguagem, a tradução só é possível depois de ser executado o fluxograma.

# Algumas notas sobre Visual Basic:

* Usa a mudança de linha para terminar uma linha de código.
* Normalmente faz uso de eventos para executar o código mas neste caso foi usado o modo de consola
* As funções podem ser definidas antes ou depois do Main.
* O código dever ser guardado num ficheiro com o mesmo nome do módulo (Module) e com extensão *.vb*.
* A linguagem permite fazer a inclusão de bibliotecas através da instrução Imports. As bibliotecas devem ser incluídas imediatamente antes da definição do módulo (ver [*estrutura início*](#_Estrutura_Início)).
* A primeira função a ser codificada deve ser o início.

# Estrutura Início

## Início:

Início

Module Programa

Sub Main()

*Resto do programa*

End Module

**Nota:** O nome do módulo (*Programa*), é um nome que identifica o algoritmo que está a ser resolvido. Todo o código dever ser guardado no ficheiro Programa.vb.

# Estrutura Fim

## Fim:

Fim

Console.ReadLine()

End Sub

**Nota:** É usado “Console.ReadLine()” para pausar o programa antes de fechar a consola.

**Nota 2:** Para terminar uma função, usa-se:

End Function

# Variáveis

## Equivalência entre TIPOS de variáveis

|  |  |  |
| --- | --- | --- |
| TIPO | Portugol | Visual Basic |
| Inteiro | Inteiro | Long |
| Real | Real | Double |
| Texto | Texto | String |
| Caracter | Caracter | Char |
| Lógico | Logico | Boolean |

Tabela 1 - Tipos de variáveis

## Definição e atribuição de variáveis

variavel <- expressao

### Se a variável não estiver definida em memória

**Passo 1**: Avaliar a expressão (VALOR).

**Passo 2**: Calcular Tipo do VALOR.

**Passo 3**: Declarar a variável: Dim variável As TIPO = expressao

### Se a variável estiver definida em memória

variavel = expressao

### Alguns exemplos de definição e atribuição de variáveis

Existem duas formas de definir variáveis e proceder à sua atribuição.

* Long
  + *1 – Definir e atribuir variável no mesmo passo:*

Dim i As Long = valor

* + *2 – Definir e atribuir variável em passos separados:*

Dim variavel As Long

variavel = valor

* + **Nota 1**: Pode ser definido como *Long* ou *long.*
  + **Nota 2**: *valor* é um número inteiro.
* Double
  + *1 – Definir e atribuir variável no mesmo passo:*

Dim variavel As Double = valor

* + *2 – Definir e atribuir variável em passos separados:*

Dim variavel As Double

variavel = valor

* + **Nota 1**: Pode ser definido como *Double* ou *double*
  + **Nota 2**: *valor* é um número decimal. Ex: 5.3.
* String
  + *1 – Definir e atribuir variável no mesmo passo:*

Dim variavel As String = "valor"

* + *2 – Definir e atribuir variável em passos separados:*

Dim variavel As String

variavel = "valor"

* + **Nota 1**: Pode ser definido como *String* ou *string*
  + **Nota 2:** têm de ser usadas aspas.
* char
  + *1 – Definir e atribuir variável no mesmo passo:*

Dim variavel As Char = "X"

* + *2 – Definir e atribuir variável em passos separados:*

Dim variavel As Char

variavel = "X"

* + **Nota 1**: Pode ser definido como *Char* ou *char*
  + **Nota 2**: *X* é um caracter e deve estar dentro de aspas.
* Boolean
  + *1 – Definir e atribuir variável no mesmo passo:*

Dim variavel As Boolean = False

* + *2 – Definir e atribuir variável em passos separados:*

Dim variavel As Boolean

variavel = False

* + **Nota 1**: Pode ser definido como *Boolean* ou *boolean.*
  + **Nota 2:** Este tipo de dados pode assumir o valor *True* ou *False.*

# Estruturas input/output

## Input – Ler

variavel

|  |  |
| --- | --- |
| Tipo | Visual Basic |
| Real | Double |
| Texto | String |
| Lógico | Boolean |
| INT | Long |
| Char | Char |

Tabela 2 - Tipo de variáveis para leitura

### Se a variável não estiver definida em memória

**Passo 1:** Identificar o tipo (TIPO) de dados que foi lido.

**Passo 2:** Definir a variável:

Dim variável As TIPO

**Passo 3:** variavel = Console.ReadLine()

**Nota:** É usado “.ReadLine()” para deixar uma linha depois da leitura do valor. Caso isto não seja necessário, usa-se “.Read()”.

### Se a variável já estiver definida em memória

**Passo 1:** Realizar apenas o ***Passo 3*** do ponto anterior.

## Output – Escrever

expressao

Para escrever no ecrã:

Console.WriteLine(expressao)

* + **Nota 1**: Caso seja necessário devolver texto também, é necessário converter as variáveis para string (.ToString)

Ex: Console.WriteLine(expressao.ToString + "")

# Estruturas de Decisão

## Condição *“if”* e *“if else”*

condição

FALSE

TRUE

Instruções 2

Instruções 1

Para TRUE, escrever:

If (condição) Then

Instruções 1

Para FALSE:

Se Instruções 2 for igual a (conector) não fazer nada.

Senão, escrever:

Else

Instruções 2

### Exemplos práticos

#### Condição “if”

If (n Mod 2 = 0) Then

Console.Write("Par")

End If

#### Condição “if else”

If (n Mod 2 = 0) Then

Console.Write("Par")

Else

Console.Write("Impar")

End If

## Condição *“while”*

condição

true

false

instruções

While (condição)

Instruções

End While

## Condição *“do while”*

instruções

TRUE

condição

FALSE

Instruções

Loop While (condição)

### Exemplos práticos

#### Condição “while”

While (i <= 10)

Console.WriteLine(i)

i = i + 1

End While

#### Condição “do while”

Do

i = Console.ReadLine()

Loop While (i < 0)

# Estrutura Conector

## Conector

Se for uma condição “*do while*” escrever:

Do

Senão, escrever:

End If

# Funções

## Definir funções

Exemplo( a , b, . . .)

**Nota:** Depois da função ser executada pelo menos uma vez ( ver [*Algumas notas sobre Java*](#_Algumas_notas_sobre)), o tipo de retorno das função RETURN\_TIPO e o TIPOx dos parametros pode ser identificado:

Function exemplo(a, b, …)

### Definir função *Exemplo* sem parâmetros de entrada

Function NOME()

### Definir função *Exemplo* com parâmetros de entrada

Function NOME(PARAMETRO)

**NOME** – Nome dado à função.

**PARAMETRO** – Variável utilizada pela função para auxiliar o cálculo.

## Chamada de funções

NOME(PARAMETRO)

NOME(PARAMETRO);

### Exemplos do uso de funções

Module Funcao

Sub Main()

Dim i As Long

i = Console.ReadLine()

Dim j As Double

j = factorial(i)

Console.WriteLine(i)

Console.ReadLine()

End Sub

Function factorial(k)

If k > 2 Then

Return k \* factorial(k - 1)

Else

Return k

End If

End Function

End Module

# Estrutura de retorno

## Return

expressao

Return expressao

# Operadores

### Aritméticos

|  |  |  |
| --- | --- | --- |
| Nome | Portugol | Visual Basic |
| Adição | a + b | a + b |
| Subtração | a – b | a - b |
| Divisão | a / b | a / b |
| Multiplicação | a \* b | a \* b |
| Resto da divisão inteira | a % b | a Mod b |
| Potenciação | a ^ b | a ^ b |
| Concatenação de texto | a, b | a + b |

Tabela 3 - Equivalência de operadores aritméticos

### Lógicos

|  |  |  |
| --- | --- | --- |
| Nome | Portugol | Visual Basic |
| Disjunção | a E b | a And b |
| Conjunção | a OU b | a Or b |
| Conjunção Exclusiva | a XOU b | a Xor b |
| Negação | NAO b | Not b |

Tabela 4 - Equivalência de operadores lógicos

### Relacionais

|  |  |  |
| --- | --- | --- |
| Nome | Portugol | Visual Basic |
| Igual | a = b | a = b |
| Diferente | a =/= b | a <> b |
| Maior | a > b | a > b |
| Maior ou igual | a >= b | a >= b |
| Menor | a < b | a < b |
| Menor ou igual | a <= b | a <= b |

Tabela 5 - Equivalência de operadores relacionais

# ***ANEXO***

Para uma compreensão mais abrangente do uso das estruturas, ficam alguns exemplos mais extensivos, com o uso de várias estruturas em algoritmos completos.

## Algoritmo com o uso da condição “if”

**Problema:** Verificar se um número introduzido pelo utilizador é par.

### Fluxograma

Inicio

n

n%2=0

TRUE

FALSE

“Par”

Fim

### Código

Module VerificaPAR

Sub main()

Dim n As Integer

n = Console.ReadLine()

If (n Mod 2 = 0) Then

Console.WriteLine("Par")

End If

Console.ReadLine()

End Sub

End Module

### Esquema detalhado

Console.ReadLine()

End Sub

Module VerificaPAR

Sub main()

Dim n As Integer

n = Console.ReadLine()

End Module

End If

Console.WriteLine("Par")

If (n Mod 2 = 0) Then

Fim

TRUE

“Par”

n%2=0

n

Inicio

FALSE

## Algoritmo com o uso da condição “if else”

**Problema:** Verificar se um número introduzido pelo utilizador é par ou ímpar.

### Fluxograma

### 

TRUE

Fim

“Par”

n%2=0

n

Inicio

FALSE

“Impar”

### Código:

Module ParOuImpar

Sub main()

Dim n As Integer

n = Console.ReadLine()

If (n Mod 2 = 0) Then

Console.WriteLine("Par")

Else

Console.WriteLine("Impar")

End If

Console.ReadLine()

End Sub

End Module

### Esquema detalhado

If (n Mod 2 = 0) Then

Dim n As Integer

n = Console.ReadLine()

Module ParOuImpar

Sub main()

### 

TRUE

FALSE

Inicio

n

n%2=0

“Par”

Fim

“Impar”

Console.WriteLine("Par")

Else

Console.WriteLine("Impar")

End If

Console.ReadLine()

End Sub

End Module

## Algoritmo com o uso da condição “while”

**Problema:** Escrever um número de 1 a 10.

### Fluxograma

Inicio

i=1

i <= 10

FALSE

TRUE

i

i=i + 1

Fim

### Código

Module Numeros1ate10

Sub main()

Dim i As Integer

i = 1

While (i <= 10)

Console.WriteLine(i)

i = i + 1

End While

Console.ReadLine()

End Sub

End Module

### Esquema detalhado

## 

Module Numeros1ate10

Sub main()

Inicio

i <= 10

i

Fim

i=1

i=i + 1

TRUE

FALSE

## 

## 

Dim i As Integer

i = 1

## 

While (i <= 10)

## 

Console.WriteLine(i)

i = i + 1

End While

Console.ReadLine()

End Sub

End Module

## Algoritmo com o uso da condição “ do while”

**Problema:** Pedir um número positivo.

### Fluxograma

Inicio

i = 0

i

TRUE

i < 0

FALSE

Fim

### Código

Module Positivo

Sub main()

Dim i As Integer

i = 0

Do

i = Console.ReadLine()

Loop While (i < 0)

Console.ReadLine()

End Sub

End Module

### Esquema detalhado

## 

Module Positivo

Sub main()

## 

Dim i As Integer

i = 0

Inicio

i < 0

Fim

i = 0

i

TRUE

FALSE

## 

## 

Do

## 

i = Console.ReadLine()

## 

Loop While (i < 0)

Console.ReadLine()

End Sub

## 

## 

End Module

## Algoritmo com o uso de uma função

**Problema:** Factorial de um número.

### Fluxogramas

**Código principal**

**Função fact(k)**

fact(k)

Inicio

TRUE

FALSE

k > 2

i = 5

j = fact ( i )

k

k \* fact(k - 1)

j

Fim

Fim

### Código

Module Funcao

Sub Main()

Dim i As Integer

i = 5

Dim j As Integer

j = factorial(i)

Console.WriteLine(j)

Console.ReadLine()

End Sub

Function factorial(k)

If k > 2 Then

Return k \* factorial(k - 1)

Else

Return k

End If

End Function

End Module

### Esquema detalhado

Module Funcao

Sub main()

fact(k)

k > 2

Fim

k \* fact(k - 1)

k

FALSE

TRUE

Inicio

Fim

i = 5

j = fact ( i )

j

Dim i As Integer

i = 5

Dim j As Integer

j = factorial(i)

Console.WriteLine(j)

Console.ReadLine()

End Sub

Function factorial(k)

If k > 2 Then

Return k \* factorial(k - 1)

End Function

End If

Return k

Else

End Module