2013

Decode Team

Instituto Politécnico de tomar

Portugol

Equivalências de estruturas entre Portugol e C#

Índice

[Nota Geral: 3](#_Toc358208669)

[Algumas notas sobre C#: 3](#_Toc358208670)

[Estrutura Início 3](#_Toc358208671)

[Início: 3](#_Toc358208672)

[Estrutura Fim 4](#_Toc358208673)

[Fim: 4](#_Toc358208674)

[Variáveis 4](#_Toc358208675)

[Equivalência entre TIPOS de variáveis 4](#_Toc358208676)

[Definição e atribuição de variáveis 4](#_Toc358208677)

[Se a variável não estiver definida em memória 4](#_Toc358208678)

[Se a variável estiver definida em memória 4](#_Toc358208679)

[Alguns exemplos de definição e atribuição de variáveis 5](#_Toc358208680)

[Estruturas input/output 7](#_Toc358208681)

[Input – Ler 7](#_Toc358208682)

[Se a variável já estiver definida em memória 7](#_Toc358208683)

[Se a variável não estiver definida em memória 7](#_Toc358208684)

[Output – Escrever 8](#_Toc358208685)

[Estruturas de Decisão 8](#_Toc358208686)

[Condição *“if”* e *“if else”* 8](#_Toc358208687)

[Exemplos práticos 9](#_Toc358208688)

[Condição *“while”* 9](#_Toc358208689)

[Condição *“do while”* 10](#_Toc358208690)

[Exemplos práticos 10](#_Toc358208691)

[Estrutura Conector 11](#_Toc358208692)

[Conector 11](#_Toc358208693)

[Funções 12](#_Toc358208694)

[Definir funções 12](#_Toc358208695)

[Definir função *Exemplo* sem parâmetros de entrada 12](#_Toc358208696)

[Definir função *Exemplo* com parâmetros de entrada 12](#_Toc358208697)

[Chamada de funções 13](#_Toc358208698)

[Exemplos do uso de funções 13](#_Toc358208699)

[Estrutura de retorno 13](#_Toc358208700)

[Return 13](#_Toc358208701)

[Operadores 14](#_Toc358208702)

[Aritméticos 14](#_Toc358208703)

[Lógicos 14](#_Toc358208704)

[Relacionais 14](#_Toc358208705)

[ANEXO 15](#_Toc358208706)

[Algoritmo com o uso da condição “if” 15](#_Toc358208707)

[Fluxograma 15](#_Toc358208708)

[Código 15](#_Toc358208709)

[Esquema detalhado 16](#_Toc358208710)

[Algoritmo com o uso da condição “if else” 17](#_Toc358208711)

[Fluxograma 17](#_Toc358208712)

[Código: 17](#_Toc358208714)

[Esquema detalhado 18](#_Toc358208715)

[Algoritmo com o uso da condição “while” 19](#_Toc358208716)

[Fluxograma 19](#_Toc358208717)

[Código 19](#_Toc358208718)

[Esquema detalhado 20](#_Toc358208719)

[Algoritmo com o uso da condição “ do while” 20](#_Toc358208720)

[Fluxograma 21](#_Toc358208721)

[Código 21](#_Toc358208722)

[Esquema detalhado 21](#_Toc358208723)

[Algoritmo com o uso de uma função 23](#_Toc358208724)

[Fluxogramas 23](#_Toc358208725)

[Código 23](#_Toc358208726)

[Esquema detalhado 24](#_Toc358208727)

[Esquema detalhado 25](#_Toc358208728)

# Nota Geral:

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

# Algumas notas sobre C#:

* É case sensitive.
* Usa o ponto e virgula (;) para terminar uma linha de código.
* As funções podem ser definidas antes ou depois no main.
* A primeira função a ser codificada deve ser o início.
* As classes em C# estão sempre dentro de namespaces;
* O código fica guardado em ficheiros do tipo .cs;

# Estrutura Início

## Início:

Início

using System;

namespace nome{

class nome{

public static void Main(string[] args){

*Resto do programa*

}

**Nota:** O nome da classe (*nome*), é um nome que identifica o algoritmo que está a ser resolvido, e não necessita de estar associado ao ficheiro .cs. O nome do namespace também não necessita de estar associado ao projecto.

# Estrutura Fim

## Fim:

Fim

}}

# Variáveis

## Equivalência entre TIPOS de variáveis

|  |  |  |
| --- | --- | --- |
| TIPO | Portugol | C# |
| Inteiro | Inteiro | int |
| Real | Real | double |
| Texto | Texto | String |
| Caracter | Caracter | char |
| Lógico | Logico | bool |

Tabela - 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: TIPO variavel = 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.

* **Int**
  + ***1 – Definir e atribuir variável no mesmo passo:***
  + int variavel =valor;
  + ***2 – Definir e atribuir variável em passos separados:***
  + int variavel;
  + variável = valor;
  + **Nota 1**: *valor* é um número inteiro.
  + **Nota 2:** Podem ser introduzidas várias variáveis através do uso da vírgula ( int a,b;).
* **Double**
  + ***1 – Definir e atribuir variável no mesmo passo:***
  + double variavel =valor;
  + ***2 – Definir e atribuir variável em passos separados:***
  + double variavel;
  + variavel=valor;
  + **Nota 1**: *valor* é um número decimal. Ex: 5.3.
* **String**
  + ***1 – Definir e atribuir variável no mesmo passo:***
  + string variavel="valor";
  + ***2 – Definir e atribuir variável em passos separados:***
  + string variavel;
  + variavel=”valor”;
  + **Nota 1**: tem que ser definido com letra minúscula.
  + **Nota 2:** têm de ser usadas aspas.
* **Char**
  + ***1 – Definir e atribuir variável no mesmo passo:***
  + char variavel =’X’;
  + ***2 – Definir e atribuir variável em passos separados:***
  + char variavel;
  + variavel=’X’;
  + **Nota 1**: tem que ser definido com letra minúscula
  + **Nota 2**: *X* é um caracter e deve estar dentro de pelicas.
* **Bool**
  + ***1 – Definir e atribuir variável no mesmo passo:***
  + bool variavel =false;
  + ***2 – Definir e atribuir variável em passos separados:***
  + bool variavel;
  + variavel=false;
  + **Nota 1:** Este tipo de dados pode assumir o valor *true* ou *false.*

# Estruturas input/output

## Input – Ler

variavel

|  |  |
| --- | --- |
| Tipo | C# |
| Real | double |
| Texto | string |
| Lógico | bool |
| INT | int |
| Char | char |

Tabela - Tipo de variáveis para leitura

### Se a variável já estiver definida em memória

### 

**Passo 1:** Em C# a leitura de inputs necessita de ser transformada no tipo da variável que vai receber esse input, isso e feito através do método Parse(), que tem como parâmetro o comando de leitura:

variavel =tipo.Parse(Console.ReadLine());

**Nota 1:** tipo é substituido pelo tipo da variavel ou seja por int, double, char, string etc.

### Se a variável não estiver definida em memória

**Passo 1:** Definir a variável:

TIPO variavel;

**Passo 2:** Especificar que o input é desse tipo e associar o input à variável:

variavel= tipo.Parse(Console.ReadLine());

## Output – Escrever

expressao

Para escrever no ecrã:

Console.Write(expressão + “”);

Para escrever no ecrã e passar de linha:

Console.WriteLine(expressão + “”);

# Estruturas de Decisão

## Condição *“if”* e *“if else”*

condição

FALSE

TRUE

Instruções 2

Instruções 1

Para TRUE, escrever:

If (*condição*) {

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%2==0) {

Console.WriteLine("Par");

}

#### Condição “if else”

if (n % 2 == 0) {

Console.WriteLine ("Par");

} else {

Console.WriteLine ("Impar");

}

## Condição *“while”*

condição

true

false

instruções

while (*condição*){

Instruções

}

## Condição *“do while”*

instruções

TRUE

condição

FALSE

Instruções

do{

}while(*condição*);

### Exemplos práticos

#### Condição “while”

while(i<=10){

Console.Write(i);

i++;

}

#### Condição “do while”

do{

i=int.Parse(Console.Read());

}while(i<0);

# Estrutura Conector

## Conector

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

do {

Senão, escrever:

}

# Funções

## Definir funções

Exemplo( a , b, . . .)

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

public static RETURN\_TIPO exemplo( TIPO1 a , TIPO2 b, . . .. )

{

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

public static TIPO NOME () {

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

public static TIPO NOME (PARAMETRO) {

**TIPO** – Executa a função e calcula o tipo de retorno.

Consultar *tabela 1* no ponto [*Equivalência entre TIPOS de variáveis*](#_Equivalência_entre_TIPOS).

**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

public class Funcao {

public static void main(String[] args) {

int i=5;

int j;

j=fact(i);

Console.WriteLine(j);

}

public static int fact(int k){

if (k>2) {

return k\*fact(k-1);

}else{

return k;

}

}

}

# Estrutura de retorno

## Return

expressao

return expressao;

# Operadores

### Aritméticos

|  |  |  |
| --- | --- | --- |
| Nome | Portugol | C# |
| 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 |
| Potenciação |  | Math.pow(base,expoente); |
| Concatenação de texto |  | + |

Tabela - Equivalência de operadores aritméticos

### Lógicos

|  |  |  |
| --- | --- | --- |
| Nome | Portugol | C# |
| Disjunção | a && b | a && b |
| Conjunção | a || b | a || b |
| Conjunção Exclusiva | a ^ b | a ^ b |
| Negação |  | a !b |

Tabela - Equivalência de operadores lógicos

### Relacionais

|  |  |  |
| --- | --- | --- |
| Nome | Portugol | C# |
| 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 - 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

using System;

namespace nome{

class VerificaPAR {

public static void main(String[] args) {

int n;

n=int.Parse(Console.ReadLine());

if (n%2==0) {

Console.WriteLine("Par");

}

}

}

### Esquema detalhado

|  |  |
| --- | --- |
| Inicio | using System;  namespace nome{  class VerificaPAR {  public static void main(String[] args) { |
| n | int n;  n = int.Parse(Console.ReadLine()); |
| FALSE  “Par”  n%2=0  TRUE | if (n%2==0) {  Console.WriteLine("Par"); |
|  | } |
| Fim | }}} |

## 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:

using System;

namespace nome{

class ParOuImpar {

public static void main(String[] args) {

int n;

n = int.Parse(Console.ReadLine());

if (n % 2 == 0) {

Console.WriteLine("Par");

} else {

Console.WriteLine("Impar");

}

}

}}

### Esquema detalhado

|  |  |
| --- | --- |
| Inicio | using System;  namespace nome{  class ParOuImpar {  public static void main(String[] args) { |
| n | int n;  n = int.Parse(Console.ReadLine()); |
| n%2=0  FALSE  “Par”  TRUE  “Impar” | if (n % 2 == 0) {  Console.WriteLine("Par");  } else {  Console.WriteLine("Impar"); |
|  | } |
| Fim | }}} |

## 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

using System;

namespace nome{

class Numeros1ate10 {

public static void main(String[] args) {

int i=1;

while(i<=10){

Console.WriteLine(i);

i++;

}

}

}}

### Esquema detalhado

|  |  |
| --- | --- |
| Inicio | using System;  namespace nome{  class Numeros1ate10 {  public static void main(String[] args) { |
| i=1 | int i=1; |
| i <= 10  FALSE  TRUE  i=i + 1  i | while(i<=10){  Console.WriteLine(i);  i++;  } |
| Fim | }}} |

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

**Problema:** Pedir um número positivo.

### Fluxograma

Inicio

i = 0

i

TRUE

i < 0

FALSE

Fim

### Código

using System;

namespace nome{

class Positivo {

public static void main(String[] args) {

int i=0;

do{

i=int.Parse(Console.ReadLine());

}while(i<0);

}

}}

### Esquema detalhado

|  |  |
| --- | --- |
| Inicio | using System;  namespace nome{  class Positivo {  public static void main(String[] args) { |
| i = 0 | int i=0; |
|  | do{ |
| i  FALSE  TRUE  i < 0 | i=int.Parse(Console.ReadLine());  }while(i<0); |
| Fim | }}} |

## 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

using System;

namespace nome{

class Funcao {

public static void main(String[] args) {

int i=5;

int j;

j=fact(i);

Console.WriteLine(j);

}

public static int fact(int k){

if (k>2) {

return k\*fact(k-1);

}else{

return k;

}

}

}

}

### Esquema detalhado

|  |  |
| --- | --- |
| Inicio | using System;  namespace nome{  class VerificaPAR {  public static void main(String[] args) { |
| i = 5 | int n;  n = int.Parse(Console.ReadLine()); |
| j = fact ( i ) | j=fact(i); |
| j | Console.WriteLine(j); |
| Fim | }}} |

### Esquema detalhado

|  |  |
| --- | --- |
| fact(k) | public static int fact(int k){ |
| k > 2  TRUE  FALSE  k  k \* fact(k - 1) | if (k>2) {  return k\*fact(k-1);  }else{  return k; |
|  | } |
| Fim | } |