2013

Decode Team

Instituto Politécnico de tomar

Portugol

Equivalências de estruturas entre Portugol e Pascal

Índice

[Nota Geral: 4](#_Toc358292969)

[Algumas notas sobre Pascal 4](#_Toc358292970)

[Estrutura Início 4](#_Toc358292971)

[Início: 4](#_Toc358292972)

[Estrutura Fim 5](#_Toc358292973)

[Fim: 5](#_Toc358292974)

[Variáveis 5](#_Toc358292975)

[Equivalência entre TIPOS de variáveis 5](#_Toc358292976)

[Definição e atribuição de variáveis 5](#_Toc358292977)

[Se a variável não estiver definida em memória 5](#_Toc358292978)

[Se a variável estiver definida em memória 5](#_Toc358292979)

[Alguns exemplos de definição e atribuição de variáveis 6](#_Toc358292980)

[Estruturas input/output 8](#_Toc358292981)

[Input – Ler 8](#_Toc358292982)

[Se a variável não estiver definida em memória 8](#_Toc358292983)

[Se a variável já estiver definida em memória 8](#_Toc358292984)

[Output – Escrever 9](#_Toc358292985)

[Estruturas de Decisão 9](#_Toc358292986)

[Condição *“if”* e *“if else”* 9](#_Toc358292987)

[Exemplos práticos 10](#_Toc358292988)

[Condição *“while”* 10](#_Toc358292989)

[While (condição) do 10](#_Toc358292990)

[Begin 10](#_Toc358292991)

[Instruções 10](#_Toc358292992)

[End; 10](#_Toc358292993)

[Condição *“do while”* 11](#_Toc358292994)

[Exemplos práticos 11](#_Toc358292995)

[Estrutura Conector 12](#_Toc358292996)

[Conector 12](#_Toc358292997)

[Funções 13](#_Toc358292998)

[Definir funções 13](#_Toc358292999)

[Function NOME (a : TIPO; B : TIPO; …) : RETURN\_TIPO; 13](#_Toc358293000)

[Begin 13](#_Toc358293001)

[Definir função *Exemplo* sem parâmetros de entrada 13](#_Toc358293002)

[Definir função *Exemplo* com parâmetros de entrada 13](#_Toc358293003)

[Chamada de funções 14](#_Toc358293004)

[Exemplos do uso de funções 14](#_Toc358293005)

[Estrutura de retorno 14](#_Toc358293006)

[Return 14](#_Toc358293007)

[Operadores 15](#_Toc358293008)

[Aritméticos 15](#_Toc358293009)

[Lógicos 15](#_Toc358293010)

[Relacionais 15](#_Toc358293011)

[ANEXO 16](#_Toc358293012)

[Algoritmo com o uso da condição “if” 16](#_Toc358293013)

[Fluxograma 16](#_Toc358293014)

[Código 16](#_Toc358293015)

[Var n: Integer; 16](#_Toc358293016)

[Begin 16](#_Toc358293017)

[read(n); 16](#_Toc358293018)

[IF(n mod 2 = 0)then 16](#_Toc358293019)

[Begin 16](#_Toc358293020)

[Write('Par') 16](#_Toc358293021)

[End; 16](#_Toc358293022)

[End. 16](#_Toc358293023)

[Esquema detalhado 17](#_Toc358293024)

[**Var n: Integer;** 17](file:///C:\Users\Neo\Desktop\Escola\PSI\Decode\Equivalencias%20Finais\equivalencias_Portugol_pascal.docx#_Toc358293025)

[**Begin** 17](file:///C:\Users\Neo\Desktop\Escola\PSI\Decode\Equivalencias%20Finais\equivalencias_Portugol_pascal.docx#_Toc358293026)

[**read(n);** 17](file:///C:\Users\Neo\Desktop\Escola\PSI\Decode\Equivalencias%20Finais\equivalencias_Portugol_pascal.docx#_Toc358293027)

[**IF(n mod 2 = 0)then** 17](file:///C:\Users\Neo\Desktop\Escola\PSI\Decode\Equivalencias%20Finais\equivalencias_Portugol_pascal.docx#_Toc358293028)

[**Begin** 17](file:///C:\Users\Neo\Desktop\Escola\PSI\Decode\Equivalencias%20Finais\equivalencias_Portugol_pascal.docx#_Toc358293029)

[**Write('Par')** 17](file:///C:\Users\Neo\Desktop\Escola\PSI\Decode\Equivalencias%20Finais\equivalencias_Portugol_pascal.docx#_Toc358293030)

[**End;** 17](file:///C:\Users\Neo\Desktop\Escola\PSI\Decode\Equivalencias%20Finais\equivalencias_Portugol_pascal.docx#_Toc358293031)

[**End.** 17](file:///C:\Users\Neo\Desktop\Escola\PSI\Decode\Equivalencias%20Finais\equivalencias_Portugol_pascal.docx#_Toc358293032)

[Algoritmo com o uso da condição “if else” 18](#_Toc358293033)

[Fluxograma 18](#_Toc358293034)

[18](#_Toc358293035)

[Código: 18](#_Toc358293036)

[Esquema detalhado 19](#_Toc358293037)

[19](#_Toc358293038)

[Algoritmo com o uso da condição “while” 20](#_Toc358293039)

[Fluxograma 20](#_Toc358293040)

[Código 20](#_Toc358293041)

[Esquema detalhado 21](#_Toc358293042)

[21](#_Toc358293043)

[21](#_Toc358293044)

[21](#_Toc358293045)

[21](#_Toc358293046)

[21](#_Toc358293047)

[21](#_Toc358293048)

[21](#_Toc358293049)

[Algoritmo com o uso da condição “ do while” 22](#_Toc358293050)

[Fluxograma 22](#_Toc358293051)

[Código 22](#_Toc358293052)

[Esquema detalhado 23](#_Toc358293053)

[23](#_Toc358293054)

[23](#_Toc358293055)

[23](#_Toc358293056)

[23](#_Toc358293057)

[23](#_Toc358293058)

[23](#_Toc358293059)

[23](#_Toc358293060)

[Algoritmo com o uso de uma função 24](#_Toc358293061)

[Fluxogramas 24](#_Toc358293062)

[Código 24](#_Toc358293063)

[Esquema detalhado 25](#_Toc358293064)

# Nota Geral:

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

# Algumas notas sobre Pascal

* O pascal não é case sensitive.
* O pascal utiliza o ponto-e-vírgula para indicar o final da linha de código.
* O fim do programa é sinalizado com um “End.” Sendo neste caso um ponto final e não um ponto-e-vírgula.
* As variáveis tem de ser declaradas antes do “Main”. (Ou por outras palavras, antes do Begin em que elas iram ser utilizadas).
* As funções e Processos, têm de ser criados antes do código “Main”.
* Qualquer tipo de texto tem se ser escrito dentro de pelicas (Exemplo -> ‘Ola Mundo’).
* Para fazer comentários em pascal é utilizado //
  + Exemplo: //comentário

# Estrutura Início

## Início:

Início

Begin //inicio

<Código>

**Nota:** O programa vai ter de ser guardado com a extensão “.pas” para que possa ser executado.

# Estrutura Fim

## Fim:

Fim

End. //Fim

# Variáveis

## Equivalência entre TIPOS de variáveis

|  |  |  |
| --- | --- | --- |
| TIPO | Portugol | Java |
| Inteiro | Inteiro | Integer |
| Real | Real | Real |
| Texto | Texto | String |
| Caracter | Caracter | Char |
| Lógico | Logico | Boolean |

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: Var <variavel> : <Tipo>;

### Se a variável estiver definida em memória

<variável> := <atribuição>;

### Alguns exemplos de definição e atribuição de variáveis

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

* Integer
  + *1 – Definir e atribuir variável*
  + Var variável : Integer;
  + Variável := valor;
  + **Nota 1**: Pode ser definido como *Integer* ou *integer.*
  + **Nota 2**: *valor* é um número inteiro.
* Real
  + *1 – Definir e atribuir variável*
  + Var variável : Real;
  + Variável := valor;
  + **Nota 1**: Pode ser definido como *Real* ou real.
  + **Nota 2**: *valor* é um número decimal. Ex: 5.3.
* String
  + *1 – Definir e atribuir variável*
  + Var variável : String;
  + Variável := ‘Valor’;
  + **Nota 1**: Pode ser definido como *String* ou string.
  + **Nota 2:** têm de ser usadas pelicas.
* char
  + *1 – Definir e atribuir variável*
  + Var variável : Char;
  + Variável := ‘Valor’;
  + **Nota 1**: Pode ser definido como Char ou char.
  + **Nota 2:** têm de ser usadas pelicas.
* Boolean
  + *1 – Definir e atribuir variável*
  + Var variável : *Boolean*;
  + Variável := true;
  + **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 | Java |
| Real | Real |
| Texto | Line |
| Lógico | Boolean |
| INT | Integer |
| Char | Char |

Tabela - 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:

Var <variavel> : <Tipo>;

**Passo 3:** Read(variavel) ; //Ler e continuar na mesma linha(ao escrever no ecrã)

Readln(variavel) ; // Ler e mudar de linha (ao escrever no ecrã )

### 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:

Write(expressão); //escrever e continuar na mesma linha(ao escrever no ecrã)

Writeln(expressão); //escrever e mudar de linha(ao escrever no ecrã)

# Estruturas de Decisão

## Condição *“if”* e *“if else”*

condição

FALSE

TRUE

Instruções 2

Instruções 1

Para TRUE, escrever:

If (*condição*) then

Begin

Instruções 1

Para FALSE:

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

Senão, escrever:

End

Else

Begin

Instruções 2

### Exemplos práticos

#### Condição “if”

If (n mod 2 = 0) then

Begin

Writeln(‘Par’);

End;

#### Condição “if else”

If (5 mod 2 = 0) then

Begin

Writeln('Par');

End

else

Begin

Writeln('Impar');

End;

## Condição *“while”*

condição

true

false

instruções

## While (condição) do

## Begin

## Instruções

## End;

## Condição *“do while”*

instruções

TRUE

condição

FALSE

Instruções

End;

Until(*condição*);

### Exemplos práticos

#### Condição “while”

While (i <= 10) do

Begin

Writeln(i);

I := i + 1;

End;

#### Condição “do while”

repeat

Begin

Read(i);

End;

Until(i < 0);

# Estrutura Conector

## Conector

Se for uma condição “*repeat*” escrever:

Repeat

Begin

Senão, escrever:

End;

# Funções

## Definir funções

Exemplo( a , b, . . .)

**Nota**: Nas funções existe duas maneiras de obter retorno de um valor, uma delas é atribuir o valor a retornar ao nome da função, a outra maneira é criar uma variável local (tem de ser a primeira variável local declarada) e atribuir o valor a retornar.

### Function NOME (a : TIPO; B : TIPO; …) : RETURN\_TIPO;

### Begin

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

Function NOME : RETURN\_TIPO;

Begin

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

Function NOME (PARAMETRO: TIPO) : RETURN\_TIPO;

Begin

**TIPO** – Tipo de dados do parâmetro.

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

**RETURN\_TIPO** – Tipo de dados de retorno da função.

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

Function fact(k : Integer) : Integer;

Begin

if(k > 2) then

Begin

fact := k \* fact(k-1);

End

Else

Begin

fact := k;

End;

End;

Var i, j : Integer;

Begin

i := 5;

j := fact(i);

writeln(j);

End.

# Estrutura de retorno

## Return

expressao

**Nota**: Nas funções existe duas maneiras de obter retorno de um valor, uma delas é atribuir o valor a retornar ao nome da função, a outra maneira é criar uma variável local (tem de ser a primeira variável local declarada) e atribuir o valor a retornar.

# Operadores

### Aritméticos

|  |  |  |
| --- | --- | --- |
| Nome | Portugol | Java |
| 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 | exp(ln(base)\*expoente) |

Tabela - Equivalência de operadores aritméticos

### Lógicos

|  |  |  |
| --- | --- | --- |
| Nome | Portugol | Java |
| Disjunção | a E b | a and b |
| Conjunção | a OU b | a or b |
| Conjunção Exclusiva | a XOU b | a xor b |
| Negação | NAO a |  |

Tabela - Equivalência de operadores lógicos

### Relacionais

|  |  |  |
| --- | --- | --- |
| Nome | Portugol | Java |
| 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

### Var n: Integer;

### Begin

### read(n);

### IF(n mod 2 = 0)then

### Begin

### Write('Par')

### End;

### End.

### Esquema detalhado

### **Var n: Integer;**

### **Begin**

### 

### **read(n);**

### 

### **IF(n mod 2 = 0)then**

### **Begin**

### **Write('Par')**

### **End;**

### **End.**

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:

Var n: Integer;

Begin

read(n);

IF(n mod 2 = 0)then

Begin

Write('Par');

End

else

Begin

Write('Impar');

End;

End.

### Esquema detalhado

**Var n: Integer;**

**Begin**

**read(n);**

**IF(n mod 2 = 0)then**

**Begin**

**Write('Par');**

**End**

**else**

**Begin**

**Write('Impar');**

**End;**

**End.**

TRUE

FALSE

Inicio

n

n%2=0

“Par”

Fim

“Impar”

## Algoritmo com o uso da condição “while”

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

### Fluxograma

Inicio

i <= 10

i

Fim

i=1

i=i + 1

TRUE

FALSE

### Código

Var i: Integer;

Begin

i := 1;

While (i <= 10) Do

Begin

writeln(i);

i := i + 1;

End;

End.

### Esquema detalhado

## 

Inicio

i <= 10

i

Fim

i=1

i=i + 1

TRUE

FALSE

**Var i: Integer;**

**Begin**

**i := 1;**

**While (i <= 10) Do**

**Begin**

**writeln(i);**

**i := i + 1;**

**End;**

**End.**

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

**Problema:** Pedir um número positivo.

### Fluxograma

Inicio

i < 0

Fim

i = 0

i

TRUE

FALSE

### Código

Var i : Integer;

Begin

i := 0;

repeat

Begin

readln(i);

End;

until(i < 0 )

End.

### Esquema detalhado

**Var i : Integer;**

**Begin**

**i := 0;**

**repeat**

**Begin**

**readln(i);**

**End;**

**until(i < 0 )**

**End.**

## 

Inicio

i < 0

Fim

i = 0

i

TRUE

FALSE

## Algoritmo com o uso de uma função

**Problema:** Factorial de um número.

### Fluxogramas

**Código principal**

**Função fact(k)**

fact(k)

k > 2

Fim

k \* fact(k - 1)

k

FALSE

TRUE

Inicio

Fim

i = 5

j = fact ( i )

j

### Código

Function fact(k : Integer) : Integer;

Begin

if(k > 2) then

Begin

fact := k \* fact(k-1);

End

Else

Begin

fact := k;

End;

End;

Var i, j : Integer;

Begin

i := 5;

j := fact(i);

writeln(j);

End.

### Esquema detalhado

**Var i, j : Integer;**

**Begin**

**i := 5;**

**j := fact(i);**

**writeln(j);**

**End.**

**Function fact(k : Integer) : Integer;**

**Begin**

**if(k > 2) then**

**Begin**

**fact := k \* fact(k-1);**

**End**

**Else**

**Begin**

**fact := k;**

**End;**

**End;**

fact(k)

k > 2

Fim

k \* fact(k - 1)

k

FALSE

TRUE

Inicio

Fim

i = 5

j = fact ( i )

j