2013

Decode Team

Instituto Politécnico de tomar

Portugol

Equivalências de estruturas entre Portugol e Fortran(95)

Índice

[Nota Geral: 3](#_Toc357770139)

[Algumas notas sobre Java: 3](#_Toc357770140)

[Estrutura Início 3](#_Toc357770141)

[Início: 3](#_Toc357770142)

[Estrutura Fim 4](#_Toc357770143)

[Fim: 4](#_Toc357770144)

[Variáveis 4](#_Toc357770145)

[Equivalência entre TIPOS de variáveis 4](#_Toc357770146)

[Definição e atribuição de variáveis 4](#_Toc357770147)

[Se a variável não estiver definida em memória 4](#_Toc357770148)

[Se a variável estiver definida em memória 4](#_Toc357770149)

[Alguns exemplos de definição e atribuição de variáveis 5](#_Toc357770150)

[Estruturas input/output 7](#_Toc357770151)

[Input – Ler 7](#_Toc357770152)

[Se for a primeira vez a ler do teclado 7](#_Toc357770153)

[Se a variável não estiver definida em memória 7](#_Toc357770154)

[Se a variável já estiver definida em memória 7](#_Toc357770155)

[Output – Escrever 8](#_Toc357770156)

[Estruturas de Decisão 8](#_Toc357770157)

[Condição *“if”* e *“if else”* 8](#_Toc357770158)

[Exemplos práticos 9](#_Toc357770159)

[Condição *“while”* 9](#_Toc357770160)

[Condição *“do while”* 10](#_Toc357770161)

[Exemplos práticos 10](#_Toc357770162)

[Estrutura Conector 11](#_Toc357770163)

[Conector 11](#_Toc357770164)

[Funções 12](#_Toc357770165)

[Definir funções 12](#_Toc357770166)

[Definir função *Exemplo* sem parâmetros de entrada 12](#_Toc357770167)

[Definir função *Exemplo* com parâmetros de entrada 12](#_Toc357770168)

[Chamada de funções 13](#_Toc357770169)

[Exemplos do uso de funções 13](#_Toc357770170)

[Estrutura de retorno 13](#_Toc357770171)

[Return 13](#_Toc357770172)

[Operadores 14](#_Toc357770173)

[Aritméticos 14](#_Toc357770174)

[Lógicos 14](#_Toc357770175)

[Relacionais 14](#_Toc357770176)

[ANEXO 15](#_Toc357770177)

[Algoritmo com o uso da condição “if” 15](#_Toc357770178)

[Fluxograma 15](#_Toc357770179)

[Código 15](#_Toc357770180)

[Algoritmo com o uso da condição “if else” 16](#_Toc357770181)

[Fluxograma 16](#_Toc357770182)

[Código: 16](#_Toc357770184)

[Algoritmo com o uso da condição “while” 17](#_Toc357770185)

[Fluxograma 17](#_Toc357770186)

[Código 17](#_Toc357770187)

[Algoritmo com o uso da condição “ do while” 18](#_Toc357770188)

[Fluxograma 18](#_Toc357770189)

[Código 18](#_Toc357770190)

[Algoritmo com o uso de uma função 19](#_Toc357770191)

[Fluxogramas 19](#_Toc357770192)

[Código 19](#_Toc357770193)

# Nota Geral:

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

# Algumas notas sobre Fortran:

* Não é case sensitive.
* Não necessita de qualquer pontuação para terminar uma linha de codigo.
* As funções podem ser definidas antes ou depois no main.
* O código dever ser guardado num ficheiro com extensão *.f95*.

# Estrutura Início

## Início:

Início

Em Fortran 95, o inicio de um programa não é definido, logo o programa começa onde a primeira variável é definida(caso sejam usadas variaveis), ou onde o primeiro output ou input apareça.

Apesar disso, podemos começar um programa por :

Program programa

Apesar disto não fazer diferença do inicio do programa, é algo que nos ajuda a identificar o programa.

**Nota:** O nome do programa (*Programa*), é um nome que queremos dar ao programa.

**Nota2:** o comando : implicit none , é um comando que impede a possibilidade de haver nomes de variáveis não definidos. Não é obrigatório usar, mas é aconselhável que o seja feito.

Este comando a ser usado, é usado logo no inicio do código

Program programa

implicit none

# Estrutura Fim

## Fim:

Fim

end

# Variáveis

## Equivalência entre TIPOS de variáveis

|  |  |  |
| --- | --- | --- |
| TIPO | Portugol | Fortran |
| Inteiro | Inteiro | integer |
| Real | Real | real |
| Texto | Texto | character |
| Caracter | Caracter | character |
| Lógico | Logico | logical |

Tabela - Tipos de variáveis

## Definição e atribuição de variáveis

### Nota importante de definição de variaveis

Existem duas formas para definir uma variável :

Integer :: n

Ou

Integer n

### Definiçao de variáveis com e sem implicit none

Exemplo 1:

Program exemplo

read \*,n

print \*,n

end

O exemplo 1 vai receber um valor do teclado, e escrever para o ecrã, sem que seja necessário definir a variável n

Exemplo 2:

Program exemplo

implicit none

integer :: n

read \*,n

print \*,n

end

O exemplo 2, como tem o comando implicit none, é obrigatório definir a variável n.

Se a variável não for definida, o código da erro a compilar.

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.

* integer
  + *1 – Definir e atribuir variável no mesmo passo:*
  + Integer :: variavel =valor
  + *2 – Definir e atribuir variável em passos separados:*
  + integer :: variavel
  + Variável=valor
  + **Nota 1**: Pode ser definido como *integer ou Integer.*
  + **Nota 2**: *valor* é um número inteiro.
* real
  + *1 – Definir e atribuir variável no mesmo passo:*
  + Real :: variavel =valor
  + *2 – Definir e atribuir variável em passos separados:*
  + Real :: variavel
  + variavel=valor
  + **Nota 1**: Pode ser definido como *real* ou *Real*
  + **Nota 2**: *valor* é um número decimal. Ex: 5.3.
* Character
  + *1 – Definir e atribuir variável no mesmo passo:*
  + Character(len=comprimento) :: variavel="valor"
  + *2 – Definir e atribuir variável em passos separados:*
  + Character(len=comprimento) :: variavel
  + variavel=”valor”
  + **Nota 1**: comprimento é o numero de caracteres que vamos usar
  + **Nota 2:** têm de ser usadas aspas.
* logical
  + *1 – Definir e atribuir variável no mesmo passo:*
  + Logical :: variavel =.false.
  + *2 – Definir e atribuir variável em passos separados:*
  + Logical :: variavel

variavel=.false.

* + **Nota 1**: Pode ser definido como *Logical* ou *logical.*
  + **Nota 2:** Este tipo de dados pode assumir o valor *true* ou *false.*

# Estruturas input/output

## Input – Ler

variavel

|  |  |
| --- | --- |
| Tipo | Java |
| Real | Double |
| Texto | Line |
| Lógico | Boolean |
| INT | Long |
| Char | Char |

Tabela - Tipo de variáveis para leitura

### Se a variável não estiver definida em memória

**NOTA:** apenas é necessário definir a variável caso usemos o comando *implicit none* no inicio do programa

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

**Passo 2:** Definir a variável:

TIPO :: variavel

**Passo 3:** read \*, variavel

### 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:

Print \*, expressao

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

print \*,"PAR"

endif

#### Condição “if else”

if(mod(n,2)==0) then

print \*,"PAR"

else

print \*,"IMPAR"

endif

## Condição *“while”*

condição

true

false

instruções

While (*condição*) do

Instruções

Endwhile

## Condição *“do while”*

instruções

TRUE

condição

FALSE

Instruções

Do while(*condição*)

End do

### Exemplos práticos

#### Condição “while”

while(l<=10) do

print \*,l

l=l+1

endwhile

#### Condição “do while”

do while(l<=0)

read \*,l

enddo

# Estrutura Conector

## Conector

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

do while

Senão, escrever:

enddo

# Funções

## Definir funções

Exemplo( a , b, . . .)

**Nota:** Depois da função ser executada pelo menos uma vez o tipo de retorno das função RETURN\_TIPO e o TIPOx dos parametros pode ser identificado:

RETURN\_TIPO function nome( TIPO1 a , TIPO2 b, . . .. )

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

TIPO function nome ()

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

TIPO function 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

Variavel<-NOME(PARAMETRO)

Variável=NOME(PARAMETRO)

### Exemplos do uso de funções

# PROGRAM calculo

# REAL :: a,b,c

# REAL :: av

# REAL :: media

# a=2.0

# b=3.0

# c=5.0

# AV = media(A,B,C)

# PRINT \*,”media :”,AV

# END

# REAL FUNCTION media(X,Y,Z)

# REAL :: X,Y,Z,SUM,x

# SUM = X + Y + Z

# media = SUM /3.0

# 

# END

Outro tipo de função são as funções recursivas:

**Exemplo:**

recursive integer function fact(k) result(res)

!implicit none

!integer res,k

if(k>2) then

res=k\*fact(k-1)

else

res=k

end if

endfunction

Para definir funções do tipo recursivo :

**recursive** TIPO function nome(PARAMETRO) **result(variavel)**

**recursive :** temos que escrever no inicio da definição da função de forma a que esta funcione de forma recursiva

**result(variavel) :** a clausula result é obrigatória no tipo de funções recursivas, e o valor da variável é o valor que iremos retornar.

# Estrutura de retorno

## Return

expressao

NOME\_DA\_FUNÇÂO = expressao

# Operadores

### Aritméticos

|  |  |  |
| --- | --- | --- |
| Nome | Portugol | fortran |
| Adição | a + b | a + b |
| Subtração | a – b | a - b |
| Divisão | a / b | a / b |
| Multiplicação | a \* b | a \* b |
| Resto da divisão inteira |  | Mod(a,2) |
| Potenciação |  | Base \*\* expoente |
| Concatenação de texto |  | , |

Tabela - Equivalência de operadores aritméticos

### Lógicos

|  |  |  |
| --- | --- | --- |
| Nome | Portugol | fortran |
| Disjunção | a && b | a .and. b |
| Conjunção | a || b | a .or. b |
| Conjunção Exclusiva | a ^ b |  |
| Negação |  | a.not.b |

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

### program PAR

### Integer :: n

### read \*,n

### if(mod(n,2)==0) then

### print \*,"PAR"

### endif

### end

### Esquema detalhado

### program PAR

### Integer :: n

### read \*,n

### if(mod(n,2)==0) then

### print \*,"PAR"

### endif

### end

Inicio

n

n%2=0

“Par”

Fim

TRUE

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:

program ParOuImpar

Integer :: n

read \*,n

if(mod(n,2)==0) then

print \*,"PAR"

else

print \*,"IMPAR"

endif

end

### 

### Esquema detalhado

program ParOuImpar

Integer :: n

read \*,n

if(mod(n,2)==0) then

print \*,"PAR"

else

print \*,"IMPAR"

endif

end

“Impar”

FALSE

TRUE

Fim

“Par”

n%2=0

n

Inicio

## 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

program numeros1ate10

integer :: l

l=1

while(l<=10) do

print \*,l

l=l+1

endwhile

end

## 

### Esquema detalhado

program numeros1ate10

integer :: l

l=1

while(l<=10) do

print \*,l

l=l+1

endwhile

end

i=i + 1

i=1

Fim

i

i <= 10

Inicio

FALSE

TRUE

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

**Problema:** Pedir um número positivo.

**NOTA:** Em fortran 95 **não existe diferença** entre ciclo WHILE e ciclo DO WHILE.

Por isso o próximo exercício vai ser resolvido com um ciclo While, também com o Do While.

### Fluxograma

Inicio

i = 0

i

TRUE

i < 0

FALSE

Fim

program Positivo

integer :: l

l=0

do while(l<=0)

read \*,l

enddo

end

### Código

program Positivo

integer :: l

l=0

while(l<=0)do

read \*,l

enddo

end

## 

### Esquema detalhado

program Positivo

integer :: l

l=0

while(l<=0)do

read \*,l

endwhile

end

TRUE

i

i = 0

Fim

i < 0

Inicio

FALSE

## 

## 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

|  |  |
| --- | --- |
| **Codigo Principal**  program factorial  Integer :: l,j,fact  l=5  j=fact(l)  print \*, j  end | **Função**  Recursive Integer function fact(k) result(res)    if(k>2) then  res=k\*fact(k-1)  else  res=k  end if  endfunction |

### Esquema detalhado

**MAIN**

program factorial

Integer :: l

Integer :: j,fact

l=5

j=fact(l)

print \*, j

end

j

j = fact ( i )

i = 5

Fim

Inicio

**FUNÇÂO**

Recursive Integer function fact(k) result(res)

if(k>2) then

res=k\*fact(k-1)

else

res=k

end if

endfunction

TRUE

FALSE

k

k \* fact(k - 1)

Fim

k > 2

fact(k)