2013

Decode Team

Instituto Politécnico de tomar

Portugol

Equivalências de estruturas entre Portugol e Fortran(95)

Índice

[Nota Geral: 3](#_Toc360025404)

[Algumas notas sobre Fortran: 3](#_Toc360025405)

[Estrutura Início 3](#_Toc360025406)

[Início: 3](#_Toc360025407)

[Estrutura Fim 4](#_Toc360025408)

[Fim: 4](#_Toc360025409)

[Variáveis 4](#_Toc360025410)

[Equivalência entre TIPOS de variáveis 4](#_Toc360025411)

[Definição e atribuição de variáveis 4](#_Toc360025412)

[Nota importante de definição de variaveis 4](#_Toc360025413)

[Definiçao de variáveis com e sem implicit none 5](#_Toc360025414)

[Se a variável não estiver definida em memória 5](#_Toc360025415)

[Se a variável estiver definida em memória 6](#_Toc360025416)

[Alguns exemplos de definição e atribuição de variáveis 6](#_Toc360025417)

[Estruturas input/output 8](#_Toc360025418)

[Input – Ler 8](#_Toc360025419)

[Se a variável não estiver definida em memória 8](#_Toc360025420)

[Se a variável já estiver definida em memória 8](#_Toc360025421)

[Output – Escrever 9](#_Toc360025422)

[Estruturas de Decisão 9](#_Toc360025423)

[Condição *“if”* e *“if else”* 9](#_Toc360025424)

[Exemplos práticos 10](#_Toc360025425)

[Condição *“while”* 10](#_Toc360025426)

[Condição *“do while”* 11](#_Toc360025427)

[Exemplos práticos 11](#_Toc360025428)

[Estrutura Conector 12](#_Toc360025429)

[Conector 12](#_Toc360025430)

[Funções 13](#_Toc360025431)

[Definir funções 13](#_Toc360025432)

[Definir função *Exemplo* sem parâmetros de entrada 13](#_Toc360025433)

[Definir função *Exemplo* com parâmetros de entrada 13](#_Toc360025434)

[Chamada de funções 14](#_Toc360025435)

[Exemplos do uso de funções 14](#_Toc360025436)

[Estrutura de retorno 16](#_Toc360025437)

[Return 16](#_Toc360025438)

[Operadores 17](#_Toc360025439)

[Aritméticos 17](#_Toc360025440)

[Lógicos 17](#_Toc360025441)

[Relacionais 17](#_Toc360025442)

[ANEXO 18](#_Toc360025443)

[Algoritmo com o uso da condição “if” 18](#_Toc360025444)

[Fluxograma 18](#_Toc360025445)

[Código 18](#_Toc360025446)

[Esquema detalhado 19](#_Toc360025447)

[Algoritmo com o uso da condição “if else” 20](#_Toc360025448)

[Fluxograma 20](#_Toc360025449)

[Código: 20](#_Toc360025451)

[Esquema detalhado 21](#_Toc360025452)

[Algoritmo com o uso da condição “while” 22](#_Toc360025453)

[Fluxograma 22](#_Toc360025454)

[Código 22](#_Toc360025455)

[Esquema detalhado 23](#_Toc360025456)

[Algoritmo com o uso da condição “ do while” 24](#_Toc360025457)

[Fluxograma 24](#_Toc360025458)

[Código 24](#_Toc360025459)

[Esquema detalhado 25](#_Toc360025460)

[Algoritmo com o uso de uma função 26](#_Toc360025464)

[Fluxogramas 26](#_Toc360025465)

[Código 26](#_Toc360025466)

[Esquema detalhado 27](#_Toc360025467)

# 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 1 - 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 | fortran |
| Real | Double |
| Texto | Line |
| Lógico | Boolean |
| INT | Long |
| Char | Char |

Tabela 2 - 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”*

**Nota:** os ciclos *while* e *do while*, funcionam da mesma forma, não há diferença, logo só vai ser mostrado o ciclo While.

condição

true

false

instruções

While (*condição*) do

Instruções

Endwhile

## Condição *“do while”*

instruções

TRUE

condição

FALSE

Instruções

if (condiçao) exit

enddo

Nota : Em fortran no ciclo do while, a condição tem que ser feita como se fosse uma condição de um ciclo IF, ou seja, as instruções vão ser executas se a condição não se verificar, caso se verifique, acaba o ciclo.

### Exemplos práticos

#### Condição “while”

while(l<=10) do

print \*,l

l=l+1

endwhile

# Estrutura Conector

## Conector

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

do

Senão, escrever:

Endif

# 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 3 - 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 4 - 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 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

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.

### Fluxograma

Inicio

i = 0

i

TRUE

i < 0

FALSE

Fim

### Código

program Positivo

integer :: l

do

read \*,l

if (l>0) exit

enddo

end

### Esquema detalhado

program Positivo

integer :: l

do

read \*,l

if (l>0) exit

enddo

end

i

i = 0

Fim

i < 0

Inicio

TRUE

## 

## 

## 

## 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)