Programação Estruturada

Estruturas e Funções

Prof. Edson Alves

Faculdade UnB Gama

2020

Sumário

Estruturas de seleção

- Em Fortran, a principal estrutura de seleção é o construto IF-THEN-ELSE, cuja sintaxe é

```
if (condicao) then
    blocoA
else
    blocoB
end if
```

- A condicao é uma variável ou expressão lógica
- Se a condicao for **verdadeira**, o blocoA é executado, e ao fim deste a execução segue para a código que segue o **end if**
- Caso contrário, o blocoB é executado
- A cláusula **else** é opcional
- Se um **if** segue imediatamente um **else**, é criada uma cascata de blocos mutuamente excludentes, sendo executado o primeiro cuja condição associada for verdadeira (ou o último, caso exista uma cláusula **else** final)

Exemplo de uso do construto IF-ELSE

```
1  ! Calcula o imposto de renda mensal
2  program IRRF
3
4      implicit none
5
6      real :: salario, aliquota, deducao, imposto
7
8      write(*,*) 'Insira o salário mensal: '
9      read(*,*) salario
10
11     ! Determinar a aliquota e a dedução a partir do salário
12     if (salario <= 1903.98) then
13         aliquota = 0
14     else if (salario <= 2826.65) then
15         aliquota = 0.075
16         deducao = 142.80
17     else if (salario <= 3751.05) then
18         aliquota = 0.15
19         deducao = 354.80
20     else if (salario <= 4664.68) then
21         aliquota = 0.225
22         deducao = 636.13
```

Exemplo de uso do construto IF-ELSE

```
23      else
24          aliquota = 0.275
25          deducao = 869.36
26      end if
27
28      ! Imprime o imposto a ser pago
29      if (aliquota == 0) then
30          write(*,*) 'Isento'
31      else
32          imposto = salario * aliquota - deducao
33          write(*,1) imposto
34      end if
35
36  1   format ('Imposto devido: ', F9.2)
37
38  end program IRRF
```

SELECT-CASE

- Outra estrutura de seleção disponível em Fortran é o construto SELECT-CASE, cuja sintaxe é

```
select case (seletor)
    case (lista de rótulos 1)
        bloco1
    case (lista de rótulos 2)
        bloco2
    ...
    case (lista de rótulos N)
        blocoN
    case default
        bloco_padrao
end select
```

- O seletor é uma variável ou expressão cujo tipo é **integer**, **character** ou **logical**
- As listas de rótulos descrevem os rótulos que compõem cada caso, separados por vírgulas

SELECT-CASE

- Os rótulos podem ser especificados de quatro maneiras:

```
x
a : b
L :
: R
```

- Na primeira forma, um único valor x é especificado
- Na segunda forma, são especificados todos os valores no intervalo [a, b] (aqui, a deve ser necessariamente menor do que b)
- Na terceira forma, são especificados todos os valores maiores ou iguais a L; na quarta, todos os valores menores ou iguais a R
- Uma vez determinado o valor do seletor, será executado o primeiro bloco cujo valor está relacionado na lista de rótulos, e em seguida a execução segue para a linha que sucede o **end select**
- O **case** default é opcional, e seu bloco será executado apenas se o valor do seletor não estiver listado em nenhum **case**

Exemplo de uso do construto SELECT-CASE

```
1  ! Determina a prioridade de atendimento do paciente, de acordo com a idade
2  program priority
3
4      implicit none
5
6      integer :: idade
7      character (len = 6) :: prioridade
8
9      write(*,*) 'Insira a idade do paciente: '
10     read(*,*) idade
11
12     select case (idade)
13         case (0 : 6)
14             prioridade = 'media'
15         case (65 : )
16             prioridade = 'maxima'
17         case (7 : 64)
18             prioridade = 'minima'
19         case default
20             write(*,*) 'Idade inválida!'
21             return
22     end select
```

Exemplo de uso do construto SELECT-CASE

```
23
24      ! Imprime a prioridade do paciente
25      write(*,1) idade, prioridade
26
27 1    format (I3, ' anos, prioridade: ', A6)
28
29 end program priority
```

Estruturas de laço

- Fortran disponibiliza duas estruturas de repetição
- A primeira delas é a estrutura `DO`, cuja sintaxe é

```
do variavel = a, b [, delta]
    bloco
end do
```

- A `variavel` de controle deve ser do tipo **`integer`**
- A `variavel` terá a como valor inicial e b como valor final
- Após cada execução do `bloco`, o valor da variável é acrescido do `delta`
- Se o `delta` (passo) for omitido, ele assume o valor 1 (um)
- O comando **`exit`**, se executado, encerra o laço imediatamente
- Já o comando **`cycle`** finaliza a execução do bloco, seguindo imediatamente para a atualização da `variavel`

Exemplo de uso do construto DO

```
1  ! Computa o fatorial de n
2  program priority
3
4      implicit none
5
6      integer :: n, i, factorial = 1
7
8      read(*,*) n
9
10     do i = 1, n
11         factorial = factorial * i
12    end do
13
14     write(*,1) n, factorial
15
16 1   format ('Fatorial de ', I2, I10)
17
18 end program priority
```

DO-WHILE

- Fortran possui uma segunda estrutura de repetição: o construto DO-WHILE, cuja sintaxe é

```
do while (condicao)
    bloco
end do
```

- A condicao é uma variável ou uma expressão do tipo **logical**
- Se a condicao for verdadeira, o bloco associado será executado
- Após a execução do bloco, a condição é reavaliada e, se permanecer verdadeira, o bloco é executado novamente
- Se o bloco não modifica as variáveis que compõem a condição de modo que ela possa eventualmente se tornar falsa, o laço será infinito
- Os comandos **exit** e **cycle** também podem ser usados neste construto, com o mesmo significado do construto DO

Exemplo de uso do construto DO-WHILE

```
1  ! Computa a^n com complexidade O(log n)
2  program fast_exp
3
4      implicit none
5
6      integer(16) :: a, n, res = 1, base      ! Inteiros de 128-bits
7      read(*,*) a, n
8
9      base = a
10
11     do while (n > 0)
12         if (iand(n, 1) > 0) then            ! iand(x, y) = x & y
13             res = res * base
14         end if
15
16         base = base * base
17         n = ishft(n, -1)                    ! n = n >> 1
18     end do
19
20     write(*,*) res
21
22 end program fast_exp
```

Vetores

- Fortran tem suporte nativo para **vetores** (*arrays*) de elementos de um mesmo tipo
- A sintaxe para a declaração de um vetor é

```
tipo_de_dado :: nome(dim1, dim2, ..., dimN)
```

- O parêtesis que segue o nome da variável, e as dimensões listadas, determinam a **forma** (*shape*) do vetor
- A notação de parêntesis também pode ser utilizada para acessar os elementos individuais do vetor
- Fortran utilizar a indexação matemática, de modo que o primeiro elemento do vetor tem índice 1
- A palavra-chave **`allocatable`** pode ser utilizada para declarar vetores dinâmicos
- A função **`allocate`**() reserva espaço em memória para tais vetores, e esta memória deve ser liberada após o uso por meio da função **`deallocate`**()

Exemplo de uso de vetores

```
1  ! Computa a média e o desvio padrão dos elementos do vetor xs
2  program statistics
3
4      implicit none
5
6      integer, allocatable :: xs(:)   ! Vetor dinâmico
7      integer :: n, i                 ! n = dimensão de xs
8      real :: stats(2)                ! Vetor com duas posições
9
10     write(*,*) 'Insira o número de entradas: '
11     read(*,*) n
12
13     if (n < 1) then
14         return
15     end if
16
17     allocate(xs(n))
18
19     do i = 1, n
20         write(*,*) 'Insira a entrada ', i, ': '
21         read(*,*) xs(i)
22     end do
```

Exemplo de uso de vetores

```
24      stats(1) = 0.0                      ! Média
25
26      do i = 1, n
27          stats(1) = stats(1) + xs(i)
28      end do
29
30      stats(1) = stats(1) / n
31
32      stats(2) = 0.0                      ! Desvio-padrão
33
34      do i = 1, n
35          stats(2) = stats(2) + (xs(i) - stats(1)) ** 2
36      end do
37
38      write(*,*) 'Média = ', stats(1)
39
40      stats(2) = sqrt(stats(2)/n)
41
42      write(*,*) 'Desvio = ', stats(2)
43
44      deallocate(xs)
45
46  end program statistics
```

Manipulação de vetores

- Fortran disponibiliza uma série de características úteis para a manipulação de vetores
- Por exemplo, se a, b e c são vetores de mesma dimensão (N), a expressão "c = a + b" equivale a

```
do i = 1, N
    c(i) = a(i) + b(i)
end do
```

- A atribuição "xs = k" atribui o valor k a todos os elementos do vetor xs
- A atribuição também pode ser utilizada para copiar vetores de mesma dimensão
- Além disso, há várias funções intrínsecas que manipulam vetores diretamente, como `dot_product`, `matmul`, `maxval`, `minval`, `product` e `sum`

Exemplo de manipulação de vetores

```
1  ! Calcula o ângulo entre dois vetores
2  program angle
3
4      real, parameter :: pi = acos(-1.0)
5      real :: theta, xlen, ylen
6      real :: xs(2) = (/ 1, 0 /), ys(2) = 1    ! ys = (1, 1)
7      real :: A(2, 2) = reshape((/ 0, 1, -1, 0 /), (/ 2, 2 /))
8
9      xlen = sqrt(dot_product(xs, xs))
10     ylen = sqrt(dot_product(ys, ys))
11     theta = acos(dot_product(xs, ys) / (xlen * ylen))
12     theta = theta * 180 / pi
13
14     write(*,*) 'Ângulo, em graus: ', theta
15
16     ys = matmul(A, ys)
17     theta = acos(dot_product(xs, ys) / (xlen * ylen))
18     theta = theta * 180 / pi
19
20     write(*,*) 'Ângulo após rotação, em graus: ', theta
21
22 end program angle
```

Tipos de dados de usuário

- Fortran também permite ao usuário definir novos tipos de dados, denominados dados **derivados** (*derived data types*)
- Estes dados são compostos pelo agrupamento de dados de tipos primitivos, ou mesmo de outros dados derivados
- Eles equivalem a uma **`struct`** da linguagem C
- A sintaxe para a declaração de um tipo de dado derivado é

```
type nome_do_novo_tipo
    tipo_1 :: nome_var_1
    tipo_2 :: nome_var_2
    ..
    tipo_N :: nome_var_N
end type nome_do_novo_tipo
```

- Variáveis do nome tipo são declaradas usando a sintaxe

```
type(nome_do_novo_tipo) :: var1, var2, ..., varM
```

- Os membros do novo tipo são acessados por meio do operador `'%'`

Exemplo de uso de dados derivados

```
1  ! Exemplifica a declaração e instânciação de um dado derivado
2  program pacient
3
4      type Paciente
5          character(len=256)  :: nome
6          integer             :: idade
7          real                :: peso, altura
8      end type Paciente
9
10     type(Paciente) :: p
11
12     write(*,*) 'Insira o nome do paciente: '
13     read(*,1) p%nome
14     write(*,*) 'Insira a idade, peso e altura, nesta ordem: '
15     read(*,*) p%idade, p%peso, p%altura
16
17     write(*,2) p%nome, p%idade
18
19 1   format (A10)
20 2   format ('Paciente "', A10, '" (', I3, ' anos) registrado com sucesso')
21
22 end program pacient
```

Subrotinas e Funções

- Em Fortran, uma **subrotina** difere de uma **função** no sentido de que não possui um valor de retorno
- Ambas podem ser declaradas no próprio arquivo do programa, ou em arquivos separados
- Funções são invocadas da mesma maneira que as funções intrínsecas da linguagem
- As subrotinas são invocadas por meio de um comando **`call`**
- A comunicação entre o programa e as funções e subrotinas se dá por meio de **argumentos** (ou **parâmetros**) e do **retorno**, no caso das funções
- Ambas são fundamentais em programas estruturados, no sentido que permite a organização e reuso de trechos de código, formando unidades semânticas

Funções

- A sintaxe para a declaração de uma função é a seguinte:

```
function nome_da_funcao(par1, par2, ..., parN)
    ! Declaração dos tipos dos argumentos
    ! Declaração das variáveis locais da função

    ! bloco de comandos
end function [nome_da_funcao]
```

- O retorno da função deve armazenado em uma variável local de mesmo nome da função
- O bloco de comandos pode ser encerrado prematuramente, por meio do comando **return**
- As variáveis são passadas por referência
- A primeira implicação deste fato é que os parâmetros devem ter o mesmo tipo da variável passada como parâmetro na chamada
- A segunda implicação é que, caso um parâmetro seja modificado na função, esta mudança será feita na variável original
- As funções devem ser declaradas a partir do ponto marcado pela palavra-chave **contains**

Exemplo de declaração e uso de funções

```
1  ! Calcula o coeficiente binominal (n, m)
2  program binomial
3
4      implicit none
5      integer(8) :: n, m
6
7      write(*,*) 'Insira os valores de n e m: '
8      read(*,*) n, m
9
10     write(*,*) binom(n, m)
11
12 contains
13
14     function factorial(n)
15         integer(8) :: i, n, factorial
16         factorial = 1
17
18         do i = 2, n
19             factorial = factorial * i
20         end do
21
22     end function factorial
```

Exemplo de declaração e uso de funções

```
23
24      function binom(n, m)
25          integer(8) :: n, m, binom
26
27          if ((n < 0) .or. (m < 0) .or. (n < m)) then
28              binom = 0
29              return
30          end if
31
32          binom = factorial(n) / (factorial(m) * factorial(n - m))
33
34      end function binom
35
36  end program binomial
```

Subrotinas

- A sintaxe para a declaração de subrotinas é semelhante à declaração de funções:

```
subroutine nome_da_subrotina(par1, par2, ..., parN)
    ! Declaração dos tipos dos argumentos
    ! Declaração das variáveis locais da subrotina

    ! bloco de comandos
end subroutine [nome_da_subrotina]
```

- Assim como as funções, as subrotinas recebem os valores de seus argumentos por referência, o que permite a modificação destes parâmetros
- Não há retorno em subrotinas
- As subrotinas também devem ser declaradas após a palavra-chave **contains**, e encerradas a qualquer momento por meio do comando **return**

Referências

1. **annefou**. Why Derived Data Types?, acesso em 10/02/2020.
2. **CHEUNG**, Shun Yan. Loops (DO, DO WHILE, EXIT, CYCLE), acesso em 04/02/2020.
3. GNU Fortran. IAND – Bitwise logical and, acesso em 04/02/2020.
4. GNU Fortran. ISHFT – Shift bits, acesso em 04/02/2020.
5. **PADMAN**, Rachael. Computer Physics: Self-study guide 2 – Programming in Fortran 95, University of Cambridge, Departament of Physics, 2007.
6. **SHALOM**, Elad. *A Review of Programming Paradigms Througout the History – With a Suggestion Toward a Future Approach*, Amazon, 2019.
7. **SHENE**, C. K. SELECT CASE Statement, acesso em 04/02/2020.