5

Comandos e Construtos de Controle de Fluxo

Nos capítulos anteriores foi descrito como comandos de atribuição devem ser escritos e como estes podem ser ordenados um após o outro para formar uma sequência de código, a qual é executada passo-a-passo. Na maior parte das computações, contudo, esta sequência simples de comandos é, por si só, inadequada para a formulação do problema. Por exemplo, podemos desejar seguir um de dois possíveis caminhos em uma seção de código, dependendo se um valor calculado é positivo ou negativo. Como outro exemplo, podemos querer somar 1000 elementos de uma matriz; escrever 1000 adições e atribuições é uma tarefa claramente tediosa e não muito eficiente. Claramente, a habilidade de realizar uma iteração sobre uma única adição é necessária. Podemos querer passar o controle de uma parte do programa a outra ou ainda parar completamente o processamento.

Para estes propósitos, recursos são disponíveis em Fortran que possibilitam o controle do fluxo lógico através dos comandos no programa. Os recursos contidos em Fortran correspondem aos que agora são geralmente reconhecidos como os mais apropriados para uma linguagem de programação moderna. Sua forma geral é a de um *construto de bloco* (*block construct*), o qual é um construto (uma construção ou estrutura) que começa com uma palavra-chave inicial, pode ter palavras-chave intermediárias e que termina com uma palavra-chave final que identifica a palavra-chave inicial. Cada sequência de comandos entre as palavras-chave é chamada de um *bloco*. Um bloco pode ser vazio, embora tais casos sejam raros.

Construtos executáveis podem ser *aninhados* (*nested*) ou *encadeados*, isto é, um bloco pode conter um outro construto executável. Neste caso, o bloco deve conter o construto interno por inteiro. Execução de um bloco sempre inicia com o seu primeiro comando executável.

Neste capítulo discutiremos os comandos e construtos que regulam o fluxo de execução de um programa em Fortran.

5.1 Comando e construto IF

O comando/construto `IF` fornece um mecanismo para controle de desvio de fluxo, dependendo de uma condição lógica. Há duas formas: o comando `IF` e o construto `IF`, sendo o último uma forma geral do primeiro.

5.1.1 Comando IF

No comando `IF`, o valor de uma expressão lógica escalar é testado e um único comando é executado se e somente se o seu valor for verdadeiro. A forma geral é:

```
IF (<expressão relacional e/ou lógica>) <comando executável>
```

O `<comando executável>` é qualquer, exceto aqueles que marcam o início ou o final de um bloco, como por exemplo `IF, ELSE IF, ELSE, END IF`, outro comando `IF` ou uma declaração `END`. Temos os seguintes exemplos:

```
IF (FLAG) EXIT       !Teste para determinar a saída (exit) de um bloco
IF(X-Y > 0.0) X= 0.0 !Teste para determinar se ocorre a atribuição X= 0.0
IF(COND .OR. P < Q .AND. R <= 1.0)S(I,J)= T(J,I) !Manipulação de matrizes
```

5.1.2 CONSTRUTO IF

Um construto IF permite que a execução de uma sequência de comandos (ou seja, um bloco) seja realizada, dependendo de uma condição ou de um outro bloco, dependendo de outra condição. Há três formas usuais para um construto IF. A forma mais simples tem a seguinte estrutura geral:

```
[<nome>:] IF (<expr-rel-log>) THEN
   <bloco>
END IF [<nome>]
```

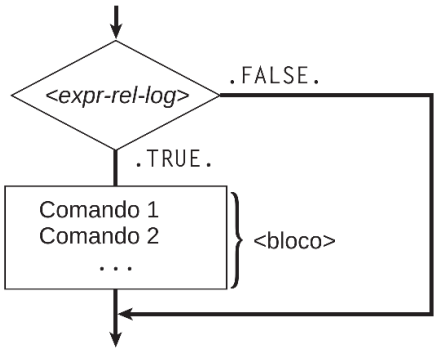

Figura 5.1: Fluxograma de um construto *IF* simples.

onde <bloco> denota uma sequência de linhas de comandos executáveis. O bloco é executado somente se o resultado da expressão relacional e/ou lógica <expr-rel-log> for verdadeiro. O construto IF pode ter um <nome>, o qual deve ser um nome válido em Fortran. O <nome> é opcional, mas se for definido no cabeçalho do construto, ele deve ser também empregado no final, denotado pela declaração END IF <nome>. A figura 5.1 mostra o fluxograma correspondente a esta forma básica do construto IF.

Como exemplo prático de um construto IF que segue o fluxograma da figura 5.1, temos:

```
SWAP: IF (X < Y) THEN
   TEMP= X
   X= Y
   Y= TEMP
END IF SWAP
```

As três linhas de texto entre "SWAP: IF ..." e "END IF SWAP" serão executadas somente se X < Y. Pode-se incluir outro construto IF ou outra estrutura de controle de fluxo no bloco de um construto IF.

A segunda forma usada para o construto IF é a seguinte:

```
[<nome>:] IF (<expr-rel-log>) THEN
   <bloco 1>
ELSE [<nome>]
   <bloco 2>
END IF [<nome>]
```

Na qual o <bloco 1> é executado se o resultado da <expr-rel-log> for verdadeira; caso contrário, o <bloco 2> será executado. Este construto permite que dois conjuntos distintos de instruções sejam executados, dependendo de um teste lógico. Um exemplo de aplicação desta forma intermediária seria:

```
IF (X < Y) THEN
   X= -Y
ELSE
   Y= -Y
END IF
```

Neste exemplo, se o resultado de X < Y for verdadeiro, então X= -Y, senão (ou seja, se X >= Y) a ação será Y= -Y.

A terceira e final versão usa a instrução ELSE IF para realizar uma série de testes independentes, cada um dos quais possui um bloco de comandos associado. Os testes são realizados um após o outro até que um deles seja satisfeito, em cujo caso o bloco associado é executado, enquanto que os outros são solenemente ignorados. Após, o controle do fluxo é transferido para a instrução END IF. Caso nenhuma condição seja satisfeita, nenhum bloco é executado, exceto se houver uma instrução ELSE final, que abarca quaisquer possibilidades não satisfeitas nos testes realizados no construto. A forma geral é:

```
[<nome>:] IF (<expr-rel-log>) THEN
   <bloco>
[ELSE IF (<expr-rel-log>) THEN [<nome>]
   <bloco>]
...
[ELSE [<nome>]
   <bloco>]
END IF [<nome>]
```

Pode haver qualquer número (inclusive zero) de instruções ELSE IF e, no máximo, uma instrução ELSE. Novamente, o <nome> é opcional, mas se for adotado no cabeçalho do construto, então deve ser mencionado em todas as circunstâncias ilustradas acima. A figura 5.2 mostra o fluxograma de um bloco IF aninhado com uma cláusula ELSE IF e a cláusula final ELSE.

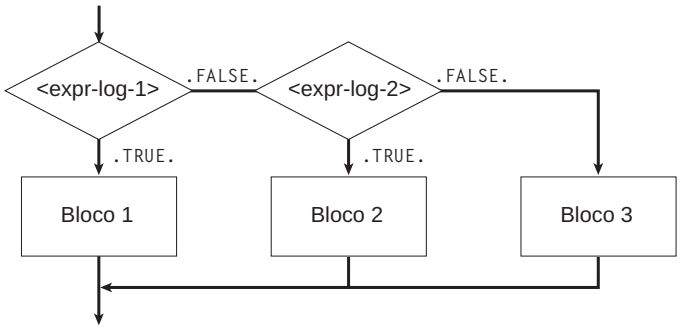

Figura 5.2: Fluxograma de um construto *IF* com uma cláusula *ELSE IF* e uma cláusula *ELSE*.

O exemplo a seguir ilustra um encadeamento de construtos IF que segue o fluxograma ilustrado na figura 5.2. Estruturas ainda mais complicadas são possíveis.

```
IF (I < 0) THEN
   IF (J < 0) THEN
      X= 0.0
      Y= 0.0
   ELSE
      Z= 0.0
   END IF
ELSE IF (K < 0) THEN
   Z= 1.0
ELSE
   X= 1.0
   Y= 1.0
END IF
```

Sugestões de uso & estilo para programação

- Procure sempre avançar a margem esquerda do corpo de um bloco em relação ao cabeçalho por dois ou mais espaços para melhorar a visualização do código.
- Quando há um número grande de construtos aninhados, é recomendável adotar <nomes> para os construtos, como forma de facilitar a leitura e compreensão do programa.

O programa-exemplo a seguir faz uso de construtos IF para implementar o cálculo do fatorial de um número natural, já utilizando o construto DO, abordado na seção 5.2.

```
!Calcula o fatorial de um número natural.
program if_fat
implicit none
integer :: i, fat, j
!
print *, "Entre com valor:"
read *, i
if (i < 0) then
```

```
   print *, "Não é possível calcular o fatorial."
else if (i == 0) then
   print *, "fat(",i,")=",1
else
   fat= 1
   do j= 1, i
      fat= fat*j
   end do
   print *, "fat(",i,")=",fat
end if
end program if_fat
```

5.2 Construto DO

Um laço DO é usado quando for necessário calcular uma série de operações semelhantes, dependendo ou não de algum parâmetro que é atualizado em cada início da série. Por exemplo, para somar o valor de um polinômio de grau N em um dado ponto x:

$$P(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_N x^N = \sum_{i=0}^{N} a_i x^i.$$

Outro tipo de operações frequentemente necessárias são aquelas que envolvem operações matriciais, como o produto de matrizes, triangularização, *etc*. Para este tipo de operações repetidas, é conveniente usar-se um laço DO para implementá-las, ao invés de escrever o mesmo bloco de operações N vezes, como aconteceria se fosse tentado implementar a soma do polinômio acima através das seguintes expressões:

```
REAL :: POL,A0,A1,A2,...,AN
POL= A0
POL= POL + A1*X
POL= POL + A2*X**2
...
POL= POL + AN*X**N
```

A forma geral de um construto DO é a seguinte:

```
[<nome>:] DO [,] [<int-index> = <expressão 1>, <expressão 2> [, <expressão 3>]]
   <bloco>
END DO [<nome>]
```

onde <int-index> é uma variável inteira, denominada *índice do laço*, e as três expressões contidas no cabeçalho do construto devem resultar em liteirais inteiros. No cabeçalho acima, cada uma das expressões indica:

<expressão 1>: o valor inicial do <int-index>;

<expressão 2>: o valor máximo, ou limite, do <int-index> (não necessariamente deve ser o último valor assumido pela variável);

<expressão 3>: o passo (ou incremento) da variável em cada nova iteração. Se o passo for omitido, este será tomado igual a 1; se estiver presente, não pode ser nulo.

O número de iterações realizadas pelo laço é definido antes de se executar o primeiro comando do <bloco> e é dado pelo resultado da conta

```
MAX((<expressão 2> - <expressão 1> + <expressão 3>)/<expressão 3> , 0),
```

onde a função MAX() é definida na seção 8.3.2. Se o resultado do primeiro argumento da função MAX acima for negativo, o laço não será executado. Isto pode acontecer, por exemplo, se <expressão 2> for menor que <expressão 1> e o passo <expressão 3> for positivo. Este tipo de

construto também é denominado um DO *iterativo.* A figura 5.3 mostra o fluxograma de um construto DO iterativo.

O valor do <int-index> (se presente) é incrementado pelo resultado da <expressão 3> ao final de cada laço para uso na iteração subsequente, se houver. Na saída do laço, o último valor adotado por essa variável permanece disponível para o programa, exceto se o laço estava inserido em um construto BLOCK (seção 9.5).

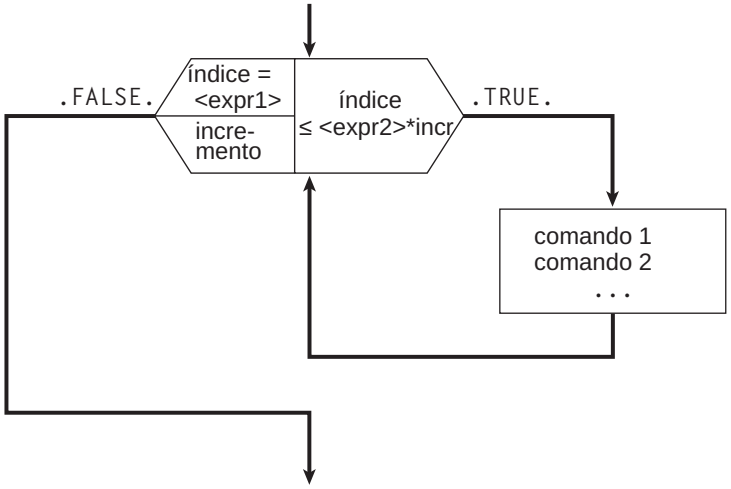

Figura 5.3: Fluxograma de um bloco *DO* iterativo.

O exemplo abaixo ilustra o uso de expressões no cabeçalho do construto:

```
DO I= J + 4, M, -K**2
   ...
END DO
```

Como se pode ver, o incremento pode ser negativo, o que significa, na prática, que os comandos do bloco somente serão executados se `M <= J + 4`. O exemplo a seguir ilustra um bloco DO onde a variável `I` somente assume valores ímpares:

```
DO I= 1, 11, 2
   ...
END DO
```

Caso alguma das expressões envolva o uso de variáveis, estas podem ser modificadas no bloco do laço, inclusive o passo das iterações. Entretanto, o número de iterações e o passo a ser realmente tomado não são modificados, uma vez que foram pré-determinados antes do início das iterações. O programa 5.1 exemplifica este fato.

Abaixo, temos um outro exemplo que ilustra o funcionamento do laço DO:

```
FAT= 1
DO I= 2, N
   FAT= FAT*I
END DO
```

Neste exemplo, o número de iterações será

```
MAX(N-2+1,0) = MAX(N-1,0).
```

Caso `N < 2`, o laço não será executado, e o resultado será `FAT= 1`. Caso `N >= 2`, o valor inicial da variável inteira `I` é 2, esta é usada para calcular um novo valor para a variável `FAT`, em seguida a variável `I` será incrementada por 1 e o novo valor `I= 3` será usado novamente para calcular o novo valor da variável `FAT`. Desta forma, a variável `I` será incrementada e o bloco executado até que `I= N + 1`, sendo este o último valor de `I` e o controle é transferido para o próximo comando após a declaração `END DO`. O exemplo ilustrado acima retornará, na variável `FAT`, o valor do fatorial de `N`.

Temos os seguintes casos particulares e instruções possíveis para um construto DO.

Listagem 5.1: Exemplo de uso do construto *DO*.

```
! Modifica o passo de um laço DO dentro do bloco.
program mod_passo
implicit none
integer :: i,j
! Bloco sem modificação de passo.
do i= 1, 10
   print *, i
end do
! Bloco com modificação do passo.
j= 1
do i= 1, 10, j
   if (i > 5)j= 3
   print *, i,j
end do
! Valor do índice após laço.
print*, 'Valor do índice i=', i
end program mod_passo
```

5.2.1 Construto DO ilimitado

Como caso particular de um construto DO, a seguinte instrução é possível:

```
[<nome>:] DO
   <bloco>
END DO [<nome>]
```

Neste caso, o conjunto de comandos contidos no <bloco> serão realizados sem limite de número de iterações, exceto se algum teste for incluído dentro do bloco, o que possibilita um desvio de fluxo para a primeira instrução após o END DO. Uma instrução que realiza este tipo de desvio é a instrução EXIT, descrita a seguir.

5.2.2 Instrução EXIT

Esta instrução permite a saída de qualquer um dos construtos de controle de fluxo discutidos neste capítulo, exceto dos construtos DO CONCURRENT[1] e CRITICAL.[2]

A forma geral desta instrução é:

```
EXIT [<nome>]
```

onde o <nome> é opcional.

Se o <nome> não for empregado, é assumido que a instrução EXIT ocorre em um construto DO e, neste caso, ele determina a saída do laço mais interno do construto. Este comportamento está ilustrado no fluxograma da figura 5.4

O uso do <nome> é opcional em construtos DO, mas é recomendado se houver diversos laços encadeados. Neste caso, a execução de um EXIT transfere controle ao primeiro comando executável após o END DO [<nome>] correspondente.

Se a instrução EXIT for empregada para forçar a saída de algum outro construto <const>, então o uso do <nome> é obrigatório e a execução da instrução transfere o controle de fluxo ao primeiro comando executável após a declaração end <const>.

Um exemplo que explora diversas possibilidades é o seguinte:

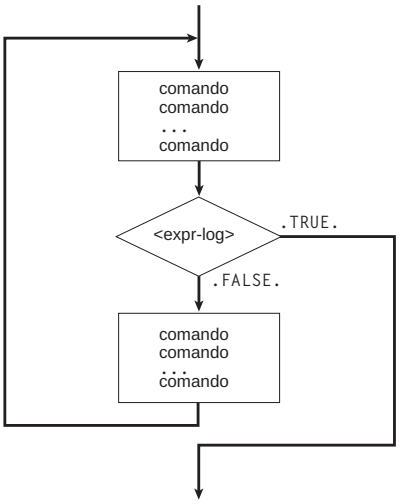

Figura 5.4: Fluxograma de um laço *DO* infinito com uma instrução *EXIT*.

[1] Discutido na seção 6.9.
[2] Empregado em coarrays.

```
DO I= 1, N
   ⋮
   IF(I == J) EXIT
   ⋮
END DO
K= I
```

Há três possibilidades:

1. Se no início do bloco `N` $\leqslant 0$, é realizada a atribuição `I= 1` mas o bloco não é executado; o controle passa para a instrução seguinte ao `END DO`, resultando com `K= 1`.
2. Se `N` $\geqslant$ `J`, a instrução EXIT será executada, resultando com `K= J`.
3. Se $0 <$ `N` $<$ `J`, o laço será executado `N` vezes; no final da última volta ocorrerá a atribuição `I= N + 1` e, ao sair do laço, resultará também `K= N + 1`.

5.2.3 INSTRUÇÃO CYCLE

A forma geral desta instrução é:

```
CYCLE [<nome>]
```

a qual transfere controle à declaração `END DO` do construto correspondente sem ocorrer a execução dos comandos posteriores à instrução. Assim, se outras iterações do laço restarem, estas são realizadas, incrementando-se o valor de <variável> (caso exista) pelo passo dado pela <expressão 3>. O programa a seguir ilustra o uso desta instrução.

```
program do_cycle
implicit none
integer :: indic= 0
do
   indic= indic + 1
   if(indic == 20) cycle ! Pula o valor índice = 20
   if(indic == 30) exit  ! Último valor impresso: índice = 29
   print *, "Valor do índice:",indic
end do
print *, "Saiu do laço.  Fim do programa."
end program do_cycle
```

5.3 CONSTRUTO CASE

O Fortran fornece uma outra alternativa para selecionar uma dentre diversas opções: trata-se do construto `CASE`. A principal diferença entre este construto e um bloco `IF` está no fato de somente uma expressão ser calculada para decidir o fluxo e esta pode ter uma série de resultados pré-definidos. A forma geral do construto `CASE` é:

```
[<nome>:] SELECT CASE (<expressão>)
[CASE (<seletor>) [<nome>]
   <bloco>]
...
[CASE DEFAULT
   <bloco>]
END SELECT [<nome>]
```

A <expressão> deve ser escalar e pode ser dos tipos inteiro, lógico ou de caractere e o valor especificado por cada <seletor> deve ser do mesmo tipo. No caso de variáveis de caracteres, os comprimentos podem diferir, mas não a espécie. Nos casos de variáveis inteiras ou lógicas, as espécies podem diferir. A forma mais simples do <seletor> é uma constante entre parênteses, como na declaração

```
CASE (1)
```

Para <expressões> inteiras ou de caracteres, um intervalo pode ser especificado separando os limites inferior e superior por dois pontos ": "

```
CASE (<inf> : <sup>)
```

Um dos limites pode estar ausente, mas não ambos. Caso um deles esteja ausente, significa que o bloco de comandos pertencente a esta declaração CASE é selecionado cada vez que a <expressão> calcula um valor que é menor ou igual a <sup>, ou maior ou igual a <inf>, respectivamente. Um exemplo é mostrado abaixo:

```
SELECT CASE (NUMERO)      ! NUMERO é do tipo inteiro.
CASE (:-1)                ! Todos os valores de NUMERO menores que 0.
   N_SINAL= -1
CASE (0)                  ! Somente NUMERO= 0.
   N_SINAL= 0
CASE (1:)                 ! Todos os valores de NUMERO > 0.
   N_SINAL= 1
END SELECT
```

A forma geral do <seletor> é uma lista de valores e de intervalos não sobrepostos, todos do mesmo tipo que <expressão>, tal como

```
CASE (1, 2, 7, 10:17, 23)
```

Caso o valor calculado pela <expressão> não pertencer a nenhuma lista dos seletores, nenhum dos blocos é executado e o controle do fluxo passa ao primeiro comando após a declaração END SELECT. Já a declaração

```
CASE DEFAULT
```

é equivalente a uma lista de todos os valores possíveis de <expressão> que não foram incluídos nos outros seletores do construto. Portanto, somente pode haver um CASE DEFAULT em um dado construto CASE. O exemplo a seguir ilustra o uso desta declaração:

```
SELECT CASE (CH)       ! CH é do tipo de caractere.
CASE ('C', 'D', 'R':)
   CH_TYPE= .TRUE.
CASE ('I':'N')
   INT_TYPE= .TRUE.
CASE DEFAULT
   REAL_TYPE= .TRUE.
END SELECT
```

No exemplo acima, os caracteres 'C', 'D', 'R' e todos os caracteres após o último indicam nomes de variáveis do tipo de caractere. Os caracteres 'I' e 'N' indicam variáveis do tipo inteiro, e todos os outros caracteres alfabéticos indicam variáveis reais. Note que a sequência de caracteres determinada pelos intervalos 'R': ou 'I':'N' corresponde à sequência de intercalação do conjunto de caracteres empregado pela espécie.[3]

O programa-exemplo abaixo mostra o uso deste construto. Note que os seletores de caso somente testam o primeiro caractere da variável NOME, embora esta tenha um comprimento igual a 5.

```
program case_string
implicit none
character(len= 5) :: nome
print *, "Entre com o nome (5 caracteres):"
read "(a5)", nome      ! Lê string com 5 caracteres.
select case (nome)
case ("a":"z")         ! Seleciona nome que começa com letras minúsculas.
```

[3] Ver seção 4.6.

```
   print *, "Palavra inicia com Letra minúscula."
case ("A":"Z")          ! Seleciona nome que começa com letras maiúsculas.
   print *, "Palavra inicia com letras maiúscula."
case ("0":"9")          ! Seleciona números.
   print *, "Palavra inicia com números!!!"
case default            ! Outros tipos de caracteres.
   print *, "Palavra inicia com caractere especial!!!"
end select
end program case_string
```

Já o programa abaixo, testa o sinal de números inteiros:

```
program testa_case
implicit none
integer :: a
print *, "Entre com a (inteiro):"
read *, a
select case (a)
case (:-1)
   print *, "Menor que zero."
case (0)
   print *, "Igual a zero."
case (1:)
   print *, "Maior que zero."
end select
end program testa_case
```

Sugestões de uso & estilo para programação

Procure sempre incluir uma cláusula `DEFALT CASE` no construto para, no mínimo, gerar um aviso caso exista algum erro de lógica ou de execução no programa.

O programa abaixo ilustra o uso de alguns dos construtos discutidos neste capítulo.

```
! Imprime uma tabela de conversão das escalas Celsius e Fahrenheit
!   entre limites de temperatura especificados.
program conv_temp
implicit none
character(len= 1) :: escala
integer :: low_temp,high_temp,temp
real :: celsius,fahrenheit
!
read_loop: do ! Lê escala e limites.
   print *, "Escala de temperaturas (C/F):"
   read "(a)", escala
! Confere validade dos dados.
   if (escala /= "C" .and. escala /= "F")then
      print *, "Escala não válida!"
      exit read_loop
   end if
   print *, "Limites (temp. inferior , temp. superior):"
   read *, low_temp,high_temp
!
   do temp= low_temp, high_temp ! Laço sobre os limites de temperatura.
      select case (escala)      ! Escolhe fórmula de conversão
      case ("C")
         celsius= temp
         fahrenheit= 9*celsius/5.0 + 32.0
      case ("F")
         fahrenheit= temp
```

```
        celsius= 5*(fahrenheit - 32)/9.0
      end select
! Imprime tabela
      print *, celsius,"graus C correspondem a",fahrenheit,"graus F."
   end do
end do read_loop
!
! Finalização.
print *, "Final dos dados válidos."
end program conv_temp
```