Fortran 77

Autor: Prof. Luiz Bianchi
Universidade Regional de Blumenau

Sumário

1. Prólogo

O objetivo deste tutorial Fortran 77 é permitir uma rápida introdução às características mais comuns da linguagem de programação Fortran 77. O perfil deste tutorial foi inspirado no excelente tutorial desenvolvido na Universidade de Stanford EUA, em 1995.

2. Introdução

O Fortran é uma linguagem de programação de finalidade geral, desenvolvida principalmente para projetos da área de engenharia. Fortran é um acrônimo de FORmula TRANslation. Foi a primeira linguagem de programação de alto-nível, criada pela IBM em 1957. Convencionalmente, a versão da linguagem Fortran é conhecida pelos dois últimos algarismos do ano em que foi introduzida. As últimas versões distribuídas são: Fortran 66, Fortran 77 e Fortran 90 (95).

A versão mais utilizada hoje em dia é, sem dúvida, a 77, embora a versão 90 esteja crescendo rapidamente em popularidade. A versão 95 é uma versão revisada do Fortran 90 a ser avaliada e aprovada pela American National Standards Institute (ANSI). Há várias outras versões da linguagem Fortran, algumas delas visam atender sistemas específicos de computador como, por exemplo, sistemas de processamento paralelo ou de multiprogramação. Uma das mais importantes é a High Performance Fortran (HPF), considerada o autêntico padrão das versões Fortran.
Muitos compiladores Fortran 77 admitem subconjuntos de comandos, isto é, aceitam extensões fora do padrão estabelecido pela ANSI.

Por que Fortran?

Fortran é uma linguagem de programação usada em aplicações de engenharia. De tempos em tempos, os assim chamados especialistas predizem que o Fortran será rapidamente enfraquecido em popularidade e logo se tornará extinto. Essas predições invariavelmente têm falhado. É a mais duradoura e histórica linguagem de programação de computador. Uma das principais razões de sobrevivência do Fortran é o software inércio ou já desenvolvido e geralmente resistente à modificação. Uma vez que uma empresa tenha despendido muitos anos-homens e quiçá milhões de dólares em produtos de softwares, é improvável que tente converter o software para uma linguagem diferente, ainda porque a translação é sempre uma tarefa dificultosa.

Portabilidade

A principal vantagem do Fortran é que é uma linguagem padronizada pelos institutos ANSI e ISO (International Standards Organization). Conseqüentemente, um programa escrito em ANSI Fortran 77, rodará em qualquer computador que tenha o compilador Fortran 77. Desse modo, os programas Fortran são portáveis através de plataformas de máquinas.

3. Programa Fortran 77 básico

Um programa Fortran é exatamente uma seqüência de linhas de texto. O texto deve obedecer uma certa sintaxe para ser considerado um programa Fortran válido. A seguir é apresentado um exemplo simplificado de programa.

```
      program circulo
      real r, area

c Este programa lê um número real r e imprime
c a área do círculo com raio r.

      write (*,*) 'Valor do raio:'
      read  (*,*) r
      area = 3.14159*r*r
      write (*,*) 'Area = ', area

      stop
      end
```

As linhas que começam com a letra “c” são comentários e não tem outra finalidade senão a de proporcionar maior legibilidade ao programa.

Organização do programa

Um programa Fortran geralmente consiste de um programa principal e eventualmente de vários subprogramas (procedimentos ou sub-rotinas). A estrutura do programa principal é:

```
program nome
declarações
comandos
stop
end
```

O comando stop é opcional e pode parecer supérfluo uma vez que o programa terminará de qualquer modo quando alcançar o fim, mas é recomendado encerrar a execução do programa com o comando stop para enfatizar o final do fluxo de execução.

Regras de posição de colunas

O Fortran 77 não é uma linguagem de formato livre, pois impõem certas restrições para a escrita do código fonte. As regras de posição de colunas são:

Coluna	conteúdo
1	"*" ou "!" para comentários
2 a 5	rótulo ou label (opcional)
6	continuação da linha anterior (opcional)
7 a 72	comandos
73 a 80	número de seqüência (opcional, raramente usado hoje em dia)

A quase totalidade das linhas do programa Fortran 77 inicia na posição 7 e termina antes de alcançar a coluna 72, ou seja, apenas o campo destinado aos comandos é utilizado.

Comentários

A linha que inicia com a letra "**c**" ou com asterisco na primeira coluna é um comentário. Ele pode aparecer em qualquer lugar no programa. Comentários bem escritos são cruciais para a legibilidade do programa.

Continuação

Ocasionalmente, um comando ocupa mais de uma linha de codificação e para denotar essa ocorrência é usado uma marca de continuação na posição 6.
Exemplo:

```
1234567                                                      7273    80
c O próximo comando ocupa duas linhas
      area = 3.14159265358979
     +       * r * r
```

Qualquer caractere pode ser usado no lugar do sinal "+" como caractere de continuação.

Espaços em branco

Espaços em branco são ignorados no Fortran 77. Se for removido todos os brancos, o programa também será considerado sintaticamente correto, no entanto, praticamente ilegível para as pessoas.

4. Variáveis e tipos de dados

Nomes de variáveis

Os nomes de variáveis em Fortran consiste de letras **a**-**z** e de dígitos **0**-**9**. O primeiro caractere deve obrigatoriamente ser uma letra. Não há distinção entre letras maiúsculas e minúsculas.

Tipos de dados

Cada variável deve ser expressa em uma declaração. Isto estabelece o tipo de dados da variável. Os tipos de dados mais comum são:

```
integer   lista de variáveis
real      lista de variáveis
double precision  lista de variáveis
character  lista de variáveis
logical  lista de variáveis
```

A lista de variáveis consiste de nomes de variáveis separadas por vírgulas. Cada variável deve ser declarada apenas uma vez.

Exemplos:

```
integer k, m
real numero, area
character nome*w
```

Onde **w** representa o número máximo de caracteres que a variável pode conter.

```
character*w var1,var2
```

As variáveis **var1** e **var2** possuem o mesmo tamanho **w.**

```
character (len = w) var1, (len=w2) var2
```

A variável **var1** tem tamanho **w** e **var2** tamanho **w2.**

Se o tipo da variável não for declarado, o Fortran 77 utiliza as regras implícitas para determinar o tipo. Isto significa que todas as variáveis que começam com as letras **i-n** (de i até n) são consideradas inteiras (***integer***) e todas as demais, reais (***real***). Recomenda-se declarar as variáveis de forma explícita a fim de evitar ou reduzir a probabilidade de erros que podem ocorrer na construção do programa. Quando não se deseja que nenhuma variável seja declarada implicitamente deve-se usar o comando **`implicit none`** no início do programa.

Variáveis inteira e de ponto flutuante

O Fortran 77 tem apenas um tipo de variável inteira ou de ponto fixo (*fixed point*). As variáveis inteiras ***integer*** ocupam 32 bits ou 4 bytes para armazenar seus conteúdos. As variáveis de ponto flutuante (*floating point*) são de dois tipos: ***real*** – 4 bytes e ***double precision*** - 8 bytes. Versões não padronizadas do Fortran usam a sintaxe real*8 para declarar uma variável de ponto flutuante de 8 bytes.

5. Constantes

Exemplos de constantes inteiras (integer):

:

```
0
-100
32767
+15
```

Exemplos de constantes reais (real):

```
1.0
-0.25
2.0E6
3.333E-1
```

O valor do elemento com notação "**E**" pode ser obtido multiplicando-se o número que antecede a letra **E** por **10** elevado ao número que segue a letra **E** . Assim, 2.0E6 equivale a dois milhões, enquanto 3.333E-1 é aproximadamente um terço.

Double precision

Para constantes cujos números são maiores que o permitido pelo tipo real, ou que requerem alta precisão, deve ser usado o tipo de dado **double precision**. A notação é a mesma que a da constante real, exceto a letra E que é substituída pela letra D.
Constantes de dupla precisão (**double precision**):

```
2.0D-1
1D99
```

2.0D-1 é uma constante de precisão dupla que representa um quinto. 1D99 equivale a um seguido de 99 zeros.

Constantes lógicas (logical)
A constante **logical** pode conter somente dois valores escritos entre dois pontos:

```
.TRUE.
.FALSE.
```

Constantes de caracteres (character)
A constate **character** consiste de uma seqüência de caracteres, conhecida como string, colocada entre apóstrofos.
Constantes de caracteres ou alfanuméricas (**character**):

```
'ABC'
'Vale tudo'
'O dia está lindo'
```

6. O comando parameter

Com o comando **parameter** pode-se definir uma constante somente uma vez no início do programa. Por exemplo, o programa que calcula a área do círculo, pode ser escrito como segue:

```
      program circulo
      real r, area
      parameter (pi = 3.14159)

* Este programa lê um número real r e imprime
* a área do circulo com raio r.

      write (*,*) 'Valor do raio:'
      read  (*,*) r
      area = pi*r*r
      write (*,*) 'Area = ', area

      stop
      end
```

A sintaxe do comando parameter é:

```
parameter (nomeDaCostante1 = valor, nomeDaConstante2 = valor, . . .)
```

As regras do comando **parameter** são:

- A "variável" definida no comando **parameter** não é propriamente uma variável mas uma constante cujo valor não pode ser modificado.
- O **parameter** deve ser definido antes do primeiro comando executável.

Uma das razões para o uso do comando **parameter** é no caso de modificação do valor da constante, que pode ser referenciada muitas vezes no programa, basta alterar seu valor apenas uma vez no comando **parameter**.

7. Atribuição e expressões aritméticas

Atribuição

A atribuição tem a seguinte forma:

```
nome-da-variável = expressão
```

A interpretação é a seguinte: O resultado da expressão à direita do sinal de igualdade é atribuído à variável à esquerda desse sinal. A expressão à direita pode conter outras variáveis, mas seus valores nunca são alterados. Por exemplo,

```
área = pi * r ** 2
```

o valor de pi ou de r não são modificados, apenas o conteúdo de área.

Expressões aritméticas

Expressões simples são da forma: operando operador operando
Exemplos:

```
x + y
x + 2 * y
```

Os valores podem ser atribuídos à variáveis como nos exemplos que seguem:

```
soma = x + y
resul = x * y / z
```

Operadores aritméticos

A precedência dos operadores aritméticos em Fortran 77 são: (da mais alta para a mais baixa prioridade):

```
**    {exponenciação}
*, /  {multiplicação, divisão}
+, -  {adição, subtração}
```

Todos os operadores são considerados da esquerda para a direita, exceto a exponenciação que tem precedência da direita para a esquerda. Caso se queira trocar a ordem de avaliação, podem ser utilizados parênteses.
Os operadores acima são todos operadores **binários**. Os operadores **unários**, "–" para negação, tem prioridade sobre os demais.
Deve-se tomar especial cuidado ao usar o operador de divisão, que tem significado diferente para valores inteiros e reais. Se os operandos são ambos inteiros, uma divisão inteira será realizada, caso contrário terá efeito uma divisão de aritmética real. Por exemplo, **3/2** é igual a **1**, enquanto que **3./2.** é igual a **1.5**.

Operador de strings

A concatenação ou junção de duas ou mais palavras pode ser realizada através do operador "//". Exemplo:

```
a1 = 'tele'
a2 = 'visão'
b = a1 // a2   → fica: b = 'televisão'
```

8. Conversão de tipos de dados

Quando diferentes tipos de dados aparecem na mesma expressão, a conversão de tipos tem lugar, explicita ou implicitamente.
Por exemplo,

```
real x
x = x + 1
```

seria convertido o número um inteiro para o número um real. Contudo, em expressões mais sofisticadas, é boa prática de programação efetuar a conversão de tipo de modo explícito. Para números, as seguintes funções estão disponíveis:

```
int
real
dble
ichar
char
```

As primeiras três têm significados óbvios. **ichar** converte um caractere para integer, ao passo que **char** faz exatamente o oposto.

Exemplo: Multiplicar duas variáveis reais x e y usando a precisão dupla (double precision) e armazenar o resultado na variável double precision **w**:

```
w = dble(x) * dble(y)
```

Note que isto é diferente de

```
w = dble(x * y)
```

9. Expressões lógicas

A expressões lógicas podem conter apenas o valor **.TRUE.** ou **.FALSE..** A expressão lógica pode ser constituída por expressões de comparação aritméticas, utilizando os operadores relacionais.

Operadores relacionais:

Operadores	significado
.LT. ou <	menor que
.LE. ou <=	menor ou igual
.GT. ou >	maior que

.GE. ou >=	maior ou igual
.EQ. ou ==	igual
.NE. ou /=	diferente

Exemplos:

```
15.NE.20   → resulta em TRUE
30.LT.20   → resulta em FALSE
```

Os símbolos < , <=, >, >=, == e /= não podem ser usados para comparação em Fortran 77 ANSI. No entanto, tais símbolos são aceitos em muitos compiladores Fortran 77 de hoje.
O símbolo = (igual) nesses compiladores é representado por dois iguais seguidos (= =) sem espaço entre eles.

As expressões lógicas podem ser combinadas pelos operadores lógicos.

Operadores lógicos:

.AND., .OR., .NOT.

Exemplos:

```
50.GT.5 .AND. 20.GT.30 → resulta em FALSE
50.GT.5 .OR. 20.GT.30  → resulta em TRUE
.NOT.5.GT.50           → resulta em TRUE
```

Variáveis lógicas e atribuições

O valor verdade pode ser armazenado em variáveis lógicas. A atribuição é análoga a atribuição aritmética.

Exemplo:

```
Logical a, b
a = .true.
b = a.and.3.lt.5/2
```

Prioridade de execução dos operadores: 1º operadores aritméticos, 2º operadores relacionais, 3º operadores lógicos na seguinte ordem: *not*, *and* e *or*.

A ordem de precedência é importante, como o último exemplo mostrado. A regra é que as expressões aritméticas são avaliadas primeiro, depois os operadores relacionais e por fim os operadores lógicos. Desse modo, **b** receberia **.FALSE.** no exemplo acima.
As variáveis lógicas são raramente usadas em Fortran. Todavia, as expressões lógicas são bastante empregadas em declarações condicionais como na instrução **if**.

10. Estrutura condicional

Uma parte importante de qualquer linguagem de programação são os comandos condicionais. É muito comum o uso da instrução **if** em Fortran que pode ser empregada de formas diferentes:

```
if (expressão lógica) comando executável
```

Deve ser escrita em uma linha. O exemplo a seguir determina o valor absoluto de x:

```
if (x .LT. 0 )  x = -x
```

Se mais de um comando tenha de ser escrito dentro do **if**, então a seguinte sintaxe deve ser usada:

```
if (expressão lógica) then
     comandos
endif
```

A estrutura geral do comando **if** tem o seguinte aspecto:

```
if (expressão lógica) then
     comandos
elseif (expressão lógica) then
     comandos
:
:
else
     comandos
endif
```

O fluxo de execução é de cima para baixo. As expressões condicionais são avaliadas em seqüência até o sistema encontrar uma que seja verdade. O comando associado a expressão lógica verdade é executado e o controle salta para o próximo comando após o **endif**.

if aninhados

A instrução **if** pode ser aninhada em vários níveis. Para garantir a legibilidade, é importante escrever as instruções de modo indentadas.

Exemplo:

```
if (x .GT. 0) then
      If (x .GE. y) then
            write(*,*) 'x  positivo e x >= y'
      else
            write(*,*) 'x positivo, mas x < y'
      endif
elseif (x .LT. 0) then
      write(*,*) 'x negativo'
else
      write(*,*) 'x = 0'
endif
```

No entanto, a construção de muitos níveis de ifs aninhados devem ser evitados em favor da legibilidade e manutenibilidade do programa.

if aritmético

A instrução **if** aritmético ou também conhecida como **if** com deslocamento é utilizada com uma expressão aritmética cuja forma geral do comando é:

```
if (expressão aritmética) label1, label2, label3
```

onde label1, label2 e label3 são os números das linhas das instruções escritos nas colunas 2 a 5 do programa. Se o resultado da expressão aritmética for menor que zero o fluxo do programa é desviado para label1, se o resultado for zero o fluxo de execução é desviado para o label2 e se maior que zero, para o label3.

Exemplo:

```
      k = 3
      if (k – 2) 10, 20, 30
10    write(*,*) 'k menor que zero'
20    write(*,*) 'k igual a zero'
30    write(*,*) 'k maior que zero'
```

Resultado:

```
            k maior que zero
```

11. Estrutura de repetição

Para repetir a execução de instruções são utilizadas sentenças de iteração ou *loop*. As pessoas familiarizadas com outras linguagens de programação, certamente já lidaram ou ouviram falar em *for-loops, while-loops* e *until-loops.* O Fortran 77 tem apenas uma forma de construção de *loop*, chamada *do-loop*. O *do-loop* corresponde ao que é conhecido como *for-loop* em outras linguagens. Outra construção de *loop* pode ser simulada utilizando os comandos if e goto.

do-loops

O do-loop é utilizado para efetuar simples contagem. A seguir, é mostrado um exemplo simples que calcula e exibe a soma dos números inteiros de 1 a n:

```
      program somaSerie
      implicit none
      integer i, n, soma
      write(*,*)'Insira um número inteiro: '
      read(*,*)n
      soma = 0
      do 10 i = 1, n
            soma = soma + i
 10   continue
      write(*,*) 'soma = ', soma
      stop
      end
```

O número 10 é um rótulo ou *label*. O programador é responsável pela atribuição de número único para cada *label* em cada programa. Lembrar que as colunas 2 a 5 são reservadas para designar *labels*. O valor numérico dos *labels* não tem significado, portanto qualquer número pode ser usado. Tipicamente, muitos programadores fazem incrementos de 10 em 10 aos *labels*.

A variável definida no comando ***do*** é incrementada de 1 em 1 por padrão, Entretanto, pode-se definir qualquer outro valor inteiro para o incremento. O segmento de programa abaixo imprime os números pares entre 1 e 10 em ordem decrescente.

```
      program numPar
      implicit none
      integer i
      do 20 i = 10, 1, -2
            write(*,*) 'i =', i
 20   continue
      stop
      end
```

A forma geral do comando do-loop é a que segue:

```
       do label  var =  expr1, expr2, expr3
            comandos
label continue
```

var é uma variável do tipo integer (geralmente chamada de índice do loop)
expr1 especifica o valor inicial de var
expr2 é o limite terminal do loop e
expr3 é o incremento ou passo do loop.

Nota: A variável do comando do-loop nunca deve ser modificada por outro comando dentro do loop, pois causaria uma grande confusão.
Muitos compiladores Fortran 77 permitem que do-loops sejam encerrados com o comando enddo. A vantagem é que o comando label pode então ser omitido. A contrução do loop com enddo é largamente usada, mas não é um componente do Fortran 77 ANSI.

while-loops

O modo mais intuitivo para escrever o while-loop é

```
do while (expr lógica)
     comandos
enddo
```

Os comandos serão repetidos contanto que a condição no comando while seja verdadeira. Ainda que esta sintaxe seja aceita por muitos compiladores ela não faz parte do Fortran 77 ANSI. O modo correto é usar if e goto:

```
label if (expr lógica) then
           comandos
           goto label
      endif
```

A seguir, é mostrado um exemplo que calcula e imprime todas as potências de dois que são menores ou iguais a 100:

```
      program potencia2
      implicit none
      integer n
      n = 1
10    if (n .le. 100) then
            n = 2*n
            write (*,*) n
            goto 10
      endif
      stop
      end
```

until-loops

Se o critério de avaliação é no fim ao invés de ser no começo, ele é geralmente denominado de until-loop, cujo pseudocódigo é:

```
do
      comandos
until (expr lógica)
```

Isto pode ser implementado em Fortran 77 através das instruções if e goto:

```
label continue
      comandos
      if (expr lógica) goto label
```

12. O comando goto

Para avançar ou recuar na estrutura do programa pode ser utilizado o comando **goto** ou **go to**, escrito com as sílabas unidas ou separadas.
Sintaxe:

```
goto label
```

onde *label* é o número da linha para a qual o fluxo de execução é desviado.

O comando **goto** tem uma sintaxe alternativa denominada goto implícito:

```
goto (label1, label2, ..., labeln) variável
```

Exemplo:

```
      i = 3
      go to (50,60,70) i
 50   write(*,*) 'Valor de i = 1'
 60   write(*,*) 'Valor de i = 2'
 70   write(*,*) 'Valor de i = 3'
```

Resultado do exemplo:

```
Valor de i = 3.
```

13. Comandos básicos de entrada e saída

Uma parte importante de qualquer programa de computador é a manipulação de entrada e saída de dados. Nos exemplos mostrados, já foram usadas as duas formas mais comuns: **read** e **write**. O I/O (Input/Output) em Fortran pode ser de construção bastante complexa, no entanto, aqui serão descritos apenas algumas formas mais simples.

Read e write

Read é usado para entrada e **write** para saída de dados Uma forma simples é:

```
read (unidade, formato) lista-de-variaveis
write(unidade, formato) lista-de-variaveis
```

A unidade é definida por um número associado a entrada ou saída padrão do sistema. O formato corresponde ao número do rótulo (label) do comando format que será descrito mais adiante.

É possível simplificar esses comandos utilizando o símbolo asterisco (*) para alguns argumentos como fora feito em todos os exemplos anteriores.

```
read (*,*) lista-de-variáveis
write(*,*) lista-de-variáveis
```

O primeiro comando (read) fará a leitura de dados a partir da entrada considerada padrão do sistema e associa os valores às variáveis constantes na lista de variáveis e o segundo (write) efetua a gravação na unidade assinalada como padrão de saída de dados.

Exemplos

Segue um segmento de código de um programa Fortran:

```
integer m, n
real x, y

read(*,*) m, n
read(*,*) x, y
```

A leitura é realizada através da entrada padrão. Um arquivo de dados consiste de um conjunto de registros. Nos exemplos apresentados, cada registro contem um número (inteiro ou real). Os registros são separados por espaços ou vírgulas. Uma entrada válida para o programa acima mencionado seria:

```
-1  100
-1.0 1e+2
```

Ou poderiam ser colocadas vírgulas separadoras:

```
-1, 100
-1.0, 1e+2
```

Observe que a entrada de dados no Fortran 77 é sensível a linha, assim sendo é importante ter a quantidade correta de elementos (registros) em cada linha. Por exemplo, se os dados de entrada forem informados todos em uma única linha como

```
-1, 100, -1.0, 1e+2
```

então às variáveis m e n seriam atribuídos os valores –1 e 100, respectivamente, porém os últimos dois valores seriam descartados, deixando x e y sem valores ou indefinidos.

Outras versões

Para uma lista simples de entrada e saída é possível usar uma sintaxe alternativa

```
read  *, lista-de-variaveis
print *, lista-de-variaveis
```

que tem o mesmo significado dos comandos **read** e **write** descritos anteriormente. Esta versão permite leitura e gravação na unidade padrão de entrada e saída uma vez que o asterisco (*) ajusta-se ao formato.

14. O comando Format

Até agora foi mostrado o formato livre de entrada e saída. Geralmente o programador necessita de um formato específico de entrada e saída para descrever valores de diferentes tipos (inteiros, reais, caracteres, etc.). Para essa finalidade o Fortran 77 dispõem do comando **Format** que é usado tanto para entrada como para saída de dados.

Sintaxe

```
      write(*, label) lista-de-variáveis
label format(formato-codigo)
```

Um exemplo simples pode demonstrar o funcionamento. Uma variável inteira contendo 4 algarismos e um número real em notação de ponto flutuante com 3 decimais, seus formatos poderiam ser assim codificados:

```
     write(*, 900) i, x
 900 format (I4,F8.3)
```

O *label* 900 do formato foi escolhido arbitrariamente, porem é prática comum numerar os comandos ***format*** com números de valores mais altos do que os *labels* de controle de fluxo. Após a palavra-chave ***format*** segue o formato entre parênteses. I4 para o tipo inteiro com a extensão de 4 dígitos e F8.3 para que o número seja impresso segundo a notação de ponto flutuante com a largura de 8 e com 3 casas decimais.

O comando ***format*** pode ser codificado em qualquer lugar dentro da unidade de programa. Há dois estilos de programação: O comando ***format*** tanto pode suceder ao comando read/write ou todos os comandos *format* podem ser agrupados no fim do programa.

Códigos de format

Os códigos de ***format*** mais comuns são:

A	textos
D	números de dupla precisão, notação de expoente
E	números reais, notação de expoente
F	números reais, formato de ponto fixo
I	inteiros
X	espaços horizontais
/	nova linha
$	inibe o avanço de linha

A letra F (e semelhantemente D, E) tem a forma geral ***Fw.d***, onde ***w*** é uma constante inteira representando a largura do campo e ***d*** é uma constante inteira que representa a quantidade de dígitos a direita do ponto decimal.

Para inteiros apenas a largura do campo é especificada, cuja sintaxe é ***Iw***. Da mesma maneira, cadeia de caracteres podem ser especificadas como ***Aw***, porém a largura do campo geralmente é truncada.

Se o número ou série de caracteres não preencher por inteiro a largura do campo, serão adicionados espaços. Geralmente o texto é alinhado à direita, mas a regra exata varia dentre os diferentes códigos de formato.

Para espacejar na linha é usado o código ***n*X**, onde ***n*** significa a quantidade de espaços. Se ***n*** for omitido, é assumido como sendo 1 (n=1). Para o salto de linha é usado o código / (barra). Cada barra corresponde a uma nova linha. É importante dizer que o comando **read** ou **write** predefine avanço de linha.

Alguns exemplos

O seguinte trecho de código Fortran,

```
      x = 0.025
      write(*,100) 'x=', x
100   format (A,F5.3)
      write(*,110) 'x=', x
110   format (A,F4.2)
      write(*,120) 'x=', x
120   format (A,E8.2)
      write(*,130) 'x=', x
130   format (A,E7.1)
```

produz os seguintes resultados:

```
x=0.025
x=0.03
x=0.25E-01
x=0.3E-01
```

Neste exemplo para cada comando write foi definido um comando format diferente. Mas é perfeitamente possível usar o mesmo comando format para diversos write. Aliás, esta é uma das principais vantagens de usar o comando format. Este recurso é conveniente quando for preciso imprimir tabelas, por exemplo e deseja-se que cada linha tenha o mesmo formato.

Em vez de especificar o formato em comandos format separados, pode-se definir o format diretamente no comando read/write. Por exemplo, o comando

```
write (*,'(A, F8.3)') 'Resposta - x = ', x
```

é equivalente a

```
     write (*,990) 'Resposta - x = ', x
 990 format (A, F8.3)
```

O texto também pode ser digitado no comando format como no seguinte exemplo:

```
     write (*,999) x
 999 format ('Resposta - x = ', F8.3)
```

15. O comando Data

O comando **data** é outro meio para entrada de dados declarado no momento em que o programa é escrito. É similar ao comando de atribuição. A sintaxe é:

```
data lista-de-variáveis/ lista-de-valores /, ...
```

onde o três pontos significam repetição do modelo. Exemplo:

```
data m/10/, n/20/, x/2.5/, y/2.5/
```

Pode também ser escrito como segue:

```
data m,n/10,20/, x,y/2*2.5/
```

Equivale às atribuições:

```
m = 10
n = 20
x = 2.5
y = 2.5
```

O comando **data** é declarado somente uma vez antes do início dos comandos de execução do programa Por isso, o comando **data** é usado principalmente no programa principal e não em sub-rotinas.

O comando **data** pode também ser usado para inicializar vetores e matrizes. O exemplo a seguir mostra como zerar uma matriz no programa:

```
real A(10,20)
data A/ 200 * 0.0/
```

Alguns compiladores inicializam automaticamente as matrizes, mas nem todos o fazem. Naturalmente, as matrizes podem ser inicializadas com outros valores diferentes de zero. Pode-se também inicializar os elementos da matriz individualmente:

```
integer v(5)
real B(2,2)
data v/10,20,30,40,50/, B/1.0,-3.7,4.3,0.0/
```

16. Vetores e Matrizes

Muitos cálculos científicos usam vetores e matrizes. O tipo de dado Fortran usado para a representação de tais objetos é a matriz ou *array*. O array de uma dimensão corresponde ao **vetor**, ao passo que o array bidimensional corresponde a **matriz**. Para o completo entendimento de como isso funciona em Fortran 77, deve-se conhecer não só a sintaxe empregada, mas também como esses objetos são armazenados na memória.

Matriz unidimensional

A matriz unidimensional ou **vetor** é exatamente uma seqüência linear de elementos armazenados consecutivamente na memória. Por exemplo, a declaração

```
real a(20)
```

declara "a" como um array de tamanho 20. Isto é, "a" consiste de 20 números reais armazenados contiguamente na memória. Por convenção, arrays Fortran são indexados de 1 em diante. Assim sendo, o primeiro elemento no array é designado por a(1) e o último por a(20). Não obstante, pode-se definir uma faixa de índices arbitrariamente, por exemplo:

```
real b(0:19), estranho(-162:237)
```

b é similar ao exemplo anterior, exceto o índice que vai de 0 a 19. Estranho é um array que admite 400 elementos, apurados como segue: 237-(-162)+1 = 400.

O tipo dos elementos de um array pode ser qualquer dos tipos de dados básicos. Exemplos:

```
integer i(10)
logical a(0:1)
double precision x(100)
```

Cada elemento de um array pode ser considerado como uma variável isolada. Faz-se referência a um elemento do array a por **a(i)**. A seguir, é exibido um exemplo de código que armazena os 10 primeiros números quadrados no array de nome quadra e, em seguida os exibe:

```
      program potencia
      implicit none
      integer i, quadra(10)
      do 100 i = 1, 10
            quadra(i) = i**2
            write(*,'($,i3,a)')quadra(i),', '
```

```
 .100 continue
      stop
      end
```

Um erro comum em Fortran é aquele em que o programa tenta acessar elementos do array que estão fora dos limites ou indefinidos. Isto é da responsabilidade do programador uma vez que o compilador não detecta tais erros.

Matriz bidimensional

As matrizes tem essenciais aplicações em álgebra linear. São usualmente representadas por arrays de duas dimensões. Por exemplo, a declaração

```
real A(3,5)
```

define um array bidimensional de 3 x 5 = 15 números reais. É conveniente imaginar o primeiro índice como o índice de linha e o segundo como índice de coluna. Assim, entende-se como mostrado na seguinte ilustração:

```
(1,1)  (1,2)  (1,3)  (1,4)  (1,5)
(2,1)  (2,2)  (2,3)  (2,4)  (2,5)
(3,1)  (3,2)  (3,3)  (3,4)  (3,5)
```

Arrays bidimensionais também podem ter índices definidos dentro de faixa numérica abitrária. A sintaxe geral para a declaração é:

```
nome (indInferior1 : indSuperior1, indInferior2 : indSuperior2)
```

O tamanho total do array é:

```
tamanho = (indSuperior1-indInferior1+1)*(indSuperior2-indInferior2+1)
```

É muito comum em Fortran declarar arrays maiores do que o tamanho da matriz desejada. (Isso porque Fortran não permite alocação dinâmica de memória).
Exemplo:

```
      program matrizbi
      implicit none
      real a(3,5)
      integer i,j
c
c    Nesse exemplo são usados os elementos de até 3 por 3.
c
      do 20 j = 1, 3
            do 10 i = 1, 3
            a(i,j) = real(i)/real(j)
            write(*,*)a(i,j)
 10   continue
 20   continue
      stop
```

```
      end
```

Os elementos na submatriz A(1:3,4:5) ficam indefinidos. Neste caso, não é assumido que esses elementos sejam inicializados com valor zero pelo compilador. (Alguns compiladores poderão zerar, mas não todos).

Iteração implícita

Por exemplo, uma matriz bidimensional (3 x 4) de números inteiros pode ser impressa, codificando-se como segue:

```
      program loopImplicito
      implicit none
      integer a(3,4),i,j
      data a/1,2,3,4,5,6,7,8,9,10,11,12/
      do 10 i = 1, 3
            write(*,1000) (a(i,j), j=1,4)
 10   continue
1000  format (I3)
      stop
      end
```

Nas instruções acima, verifica-se uma iteração explícita referente as linhas da matriz e uma iteração implícita que se refere as colunas, índice j.

Freqüentemente, o comando format envolve repetições, por exemplo:

```
950 format (2X, I3, 2X, I3, 2X, I3, 2X, I3)
```

Para este tipo de codificação, há uma notação resumida, como:

```
950 format (4(2X, I3))
```

É possível fazer repetições sem posicionar explicitamente quantas vezes o formato será repetido. Supondo que se deseja imprimir os primeiros 50 elementos de um vetor, com 10 elementos em cada linha., pode-se proceder como segue:

```
      program repete
      implicit none
      integer x(80),i
      data x/80*100/
      write(*,1010) (x(i), i=1,50)
1010  format (10I6)
      stop
      end
```

O comando format declara que dez números serão impressos. Mas a descrição do comando diz haver 50 números a serem impressos. Contudo, depois de serem impressos os primeiros

dez elementos, o mesmo comando **format** é automaticamente utilizado para os próximos dez e assim por diante.

Armazenamento de matrizes de duas dimensões

O Fortran armazena arrays como uma seqüência linear contígua de elementos. É conveniente saber que arrays de duas dimensões são armazenados em colunas. No exemplo acima, o elemento (1,2) seguirá o elemento (3,1). Sucedem os demais da segunda coluna, depois a terceira coluna, e assim por diante.

Considerar o exemplo no qual foi utilizado apenas até 3 por 3 elementos do array definido com o tamanho de 3 x 5, A(3,5). Os 9 elementos considerados serão armazenados nas primeiras nove localizações de memória, as outras 6 últimas localizações que foram inicialmente reservadas não serão usadas.

Matriz multidimensional

O Fortran 77 permite arrays de até sete dimensões. A sintaxe e o formato de armazenamento são análogos aos de duas dimensões.

O comando dimension

Há uma forma alternativa para declarar arrays em Fortran 77. Os comandos

```
real A, x
dimension A(10,20)
dimension x(50)
```

são equivalentes a

```
real A(10,20), x(50)
```

O comando ***dimension*** é hoje um estilo considerado antiquado.

17. Subprogramas

Quando um programa é maior que umas poucas centenas de linhas, torna-se difícil sua compreensão. Programas que resolvem problemas de engenharia geralmente possuem dezenas de milhares de linhas. O único modo para tratar códigos extensos é utilizar uma abordagem modular e dividir o programa em pequenas unidades denominadas subprogramas.

Um subprograma consiste num trecho de código para solucionar um subproblema bem definido. Extensos programas, geralmente têm que resolver o mesmo subproblema com diferentes dados. Em vez de duplicar código, resolve-se esta questão através da adoção de subprogramas. O mesmo subprograma pode ser chamado muitas vezes para operar com diferentes dados de entrada.

O Fortran utiliza dois tipos de subprogramas, denominados funções e sub-rotinas.

Funções

Funções Fortran são totalmente similar as funções matemáticas: Ambos aceitam um conjunto de argumentos de entrada (parâmetros) e retornam um valor de mesmo tipo. Na discussão precedente foi falado sobre subprogramas definidos pelo usuário. O Fortran 77 também possui funções incorporadas.

Um exemplo básico ilustra o uso de uma função:

```
x = cos(pi/3.0)
```

cos refere-se a função coseno, assim x receberá o valor 0.5 (se **pi** estiver corretamente definido; o Fortran 77 não possui constantes incorporadas). Há muitas funções embutidas no Fortran 77. Algumas das mais comuns são:

```
abs   valor absoluto
min   valor mínimo
max   valor máximo
mod   resto da divisão
sqrt  raiz quadrada
sin   seno
cos   coseno
tan   tangent
atan  arco tangente
exp   exponencial (natural)
log   logaritmo (natural)
```

Exemplos:

```
real x, numAbs
integer y, z, resto
x = -1
y = 5
z = 2
numAbs = abs(x)
resto = mod(y,z)
```

Em geral, a função sempre tem um tipo. A maioria das funções internas mencionadas acima, contudo, são genéricas. Conforme mostrado no exemplo acima, **pi** e **x** podem ser do tipo ***real*** ou ***double precision***. O compilador verificará o tipo e o uso da versão correta de **cos** (***real*** ou ***double precision***). Infelizmente, o Fortran não é realmente uma linguagem polimórfica, de maneira que, em geral deve-se atentar para os tipos das variáveis e suas funções.

Agora, serão verificadas as funções definidas pelo usuário.

Uma função é semelhante a um programa, mas ela pode retornar um valor. Uma função pode utilizar argumentos, como constantes, variáveis ou expressões passadas para ela por um programa de chamada. Caso uma função não tenha argumentos, sua instrução ***function*** deve incluir um conjunto de parênteses vazio. Uma função retorna um valor atribuindo um valor ao seu nome em uma ou mais instruções do programa.

No exemplo a seguir, a função **Celsius** calcula o número de graus Celsius a partir dos graus Fahrenheit. Quando a função for chamada a partir do programa principal, uma variável que contém o valor do argumento será passada para a função. O resultado do cálculo é retornado para o programa de chamada.

```
program chama
real temp
write(*,*) 'Insira graus fahrenheit: '
read (*,*) temp
write(*,*) 'Temperatura: ', celsius(temp), ' graus C.'
stop
end

real function Celsius(fah)
Celsius = (fah - 32) * 5 / 9
return
end
```

Observar que a estrutura da função envolvida assemelha-se a do programa principal. As diferenças são:

- Funções tem um tipo. Este tipo deve também ser declarado no programa chamador.
- O valor de retorno será armazenado na variável que tem o mesmo nome da função.
- Funções são terminadas pelo comando return em vez de stop.

A sintaxe geral de uma função em Fortran 77 é:

```
tipo function nome (lista-de-variáveis)
declarações
comandos
return
end
```

A função deve ser declarada com o mesmo tipo da variável declarado no programa chamador. A função é chamada simplesmente usando o nome da função e a lista de parâmetros entre parênteses.

Sub-rotinas

Um função essencialmente retorna um valor. Pretendendo retornar dois ou mais valores (ou ocasionalmente nenhum), deve-se desenvolver uma sub-rotina. A seguir é mostrado a sintaxe de uma sub-rotina:

```
subroutine nome (lista-de-argumentos)
declarações
comandos
return
end
```

Observe-se que a sub-rotina não tem nenhum tipo especificado e conseqüentemente não será declarado no programa chamador.

A seguir é apresentado um exemplo de uma sub-rotina que tem por objetivo trocar dois valores inteiros:

```
      subroutine itroca (a, b)
      integer a, b
c Variaveis locais
      integer tmp
      tmp = a
      a = b
      b = tmp
      return
      end
```

Observar que há dois blocos de declaração de variáveis. O primeiro, declara os parâmetros de entrada e saída, isto é, variáveis comuns à ambos chamador e chamado. O segundo bloco declara variáveis locais, ou seja, aquelas que só podem ser usadas dentro do subprograma. Pode ser usado o mesmo identificador de variável em diferentes subprogramas que o compilador reconhecerá e fará a necessária distinção.

Chamada-por-referência

O Fortran 77 utiliza o paradigma denominado chamada-por-referência. Isto significa que em vez de passar os valores dos argumentos para a função/sub-rotina (chamada-por-valor), são passados os endereços de memória dos argumentos. Um pequeno exemplo, a seguir, mostra a diferença:

```
program chama
      integer m, n
c
      m = 1
      n = 2
      call itroca(m, n)
      write(*,*) m, n
      stop
      end
```

O resultado deste programa é “2 1”, exatamente o que era esperado. Entretanto, se o Fortran 77 estivesse usando chamada-por-valor o resultado teria sido “1 2”, ou seja, as variáveis m e n não seriam permutadas. A razão para isto é que apenas os valores de m e n foram copiados da sub-rotina itroca, e igualmente se a e b fosse trocado dentro da sub-rotina os novos valores não teriam sido retornado ao programa principal.

Bibliografia

- PEREIRA FILHO, Jorge da Cunha. Introducao a programacao FORTRAN. Rio de Janeiro : Campus, 1981. 339p.
- FARRER, Harry et al.Algoritmos estruturados. 2.ed. Rio de Janeiro : LTC, c1989. 259p.
- FORBELLONE, André Luiz Villar; EBERSPACHER, Henri Frederico. Lógica de programação : a construção de algoritmos e estruturas de dados. 2.ed. São Paulo : Makron Books, 2000. 197p.
- PACITTI, Tercio.Do Fortran a Internet.2. ed.Pearson Education do Brasil Ltda.
- FARRER,Harry, FARIA Eduardo Chaves, CAMPOS FILHO,Frederico Ferreira.Fortran Estruturado.LCT

Eletrônica

- http://www.strath.ac.uk/CC/Courses/fortran.html
- http://www.projetoforce.hpg.ig.com.br/ (Site do compilador Fortran FORCE)