Apostila de FORTRAN

Rita Valéria Andreoli

João Carlos Carvalho

INPE

dezembro/2001

Apresentação

No documento final do I-EPGMET (Encontro dos Alunos de Pós-Graduação em Meteorologia), realizado em 2000, expressa-se claramente que os alunos gostariam de ter cursos introdutórios dos principais softwares e linguagens de programação utilizados pela comunidade de Meteorologia do INPE. Desde então, essa demanda permaneceu presente, mas adormecida. Recentemente, em setembro, com a retomada das discussões sobre a necessidade de cursos, propôs-se a elaboração de apostilas simples, básicas, voltadas para o usuário que nada ou muito pouco conhece do software ou da linguagem de programação. Dessa proposta nasceu o **Projeto Apostila**.

O Projeto Apostila é composto de 5 grupos. Cada grupo foi responsável pela elaboração de uma apostila sobre um software ou uma linguagem de programação. Os grupos são: Rosane Chaves e Daniel Andres (Grads); Rita Valéria Andreoli e João Carlos Carvalho (Fortran); Marcelo Zeri (Matlab); Pablo Fernandez e Emanuel Giarolla (Latex); e Hélio Camargo e Marcos Oyama (Unix). Os grupos, durante 2 meses, trabalharam intensamente (sem se descuidar das suas dissertações e teses) para produzir as apostilas. A apostila que você está recebendo é fruto do esforço de um dos grupos.

Gostaríamos de agradecer a todos os colegas que revisaram a versão "0" das apostilas: Alexandra Amaro de Lima, Antônio Marcos Mendonça, Edna Sousa, Elizabeth Reuters, Everson dal Piva, José Francisco de Oliveira Júnior, Marcos Yoshida, Maurício Bolzan, Patrícia Moreno Simões, Paulo Marcelo Tasinaffo, Raimundo Moura, Rodrigo de Souza e Wantuir Aparecido de Freitas. As sugestões, críticas e os comentários apresentados foram de grande valia. Muito obrigado!

Gostaríamos, também, de agradecer a Anísio Moliterno por disponibilizar a área pública da fractal para os arquivos de exemplos das apostilas.

As apostilas não têm o objetivo de competir com os cursos que são oferecidos pelo CPTEC ou INPE, mas complementar. **A idéia é que o usuário, após estudar a apostila, possa caminhar sozinho, consultando manuais ou os colegas; ou seja, torne-se independente.** A apostila é uma ponte, não o fim. Recomenda-se aos leitores da apostila que façam os cursos oferecidos pelo CPTEC ou INPE: sempre temos algo a aprender! Além disso, no futuro, as apostilas podem servir de base para cursos ministrados por alunos - instrutores.

Espera-se que, no futuro, outros Projetos Apostila sejam realizados, melhorando e atualizando as apostilas existentes. Além disso, outras apostilas (p.ex. Fortran 90), poderão ser elaboradas.

Boa leitura!

Instalação dos exemplos

Peça a alguém do Suporte (ou algum colega que conheça um pouco de UNIX) para instalar os exemplos na sua área. As instruções são as seguintes:

1. Transfira (via ftp, como usuário anônimo) o arquivo **`instala_apostila`** da área pública da **fractal** (**`/pub/software/apostila`**) para o `home` do usuário.

(em negrito está o que você deve digitar; θ é a tecla Enter)

```
nevasca:/home/fulano>ftp fractal θ
Name (fractal:fulano):anonymous θ
Guest login ok...
Password:fulano@cptec.inpe.br θ        #ATENÇÃO: não irá aparecer na tela!
Guest login ok...
ftp>cd pub/software/apostila θ
ftp>asc θ
ftp>get instala_apostila θ
ftp>quit θ
nevasca:/home/fulano>
```

2. Abra a permissão de execução de `instala`.

```
nevasca:/home/fulano>chmod u+x instala_apostila θ
```

3. Execute `instala_apostila` (e entre com as informações pedidas durante a instalação).

```
nevasca:/home/fulano>instala_apostila θ
```

Explicação dos exemplos

A apostila de programação em FORTRAN contém vários exemplos, que são disponibilizados ao usuário. Esses exemplos, estão na forma original apresentada na apostila, e encontram-se no arquivo prog_for.tar.

1) adiciona.f - *este programa soma números de 1 a100, utilizando o comando DO, encerrado pelo comando enddo;*
2) adiciona1.f - *executa a mesma operação do programa anterior, mas utiliza o comando CONTINUE, para encerrar o laço do comando DO;*
3) array2.f - *calcula média e desvio padrãode uma série de dados fornecida a partir do tecaldo*;
4) calcula_pi.f - *calcula o valor de pi, utilizando uma função intrínseca*;
5) contagem.f - *faz contagem regressiva a partir de um número fornecido via teclado;*
6) contagem1.f - *permite observar quando estão sendo executadas as primeira e segunda intreações na estrutura de laços aninhados (comando DO);*
7) equacao.f - *calcula a equação de segundo grau*;
8) equação1.f - *calcula a equação de segundo grau, e suas raízes, utilizando a estrutura do IF aritmético;*
9) equacao2.f - *calcula a equação de segundo grau, e suas raizes utilizando o comando IF expandido.*
10) ler_arq.f - *lê séries temporais a partir de arquivos em disco, faz a soma e gera um arquivo de saída;*
11) notas.f - *fornece um conceito, dependendo, da nota fornecida via teclado*;

12) rad.f - *calcula o nível de radiação utilizando o comando DO WHILE*;
13) radiacao.f - *mostra um exemplo de formatação*;
14) raio.f - *calcula a área do circulo de raio r*
15) read.f - *lê um arquivo do disco não formatado, e escreve com um formato específico*
16) sqrt.f - *calcula raiz quadrada de um número, utilizando uma funçaõ intrínseca*
17) teste.f - *exemplo simpes utilizando a estrura do IF lógico simples*
18) soma1.f, soma2.f, soma3.f, soma4.f e soma5.f - *são exemplo simples, utilizando alguns comandos básicos.*
19) os arquivos *.dat - *são séries temporais, utilizadas em alguns exemplos*.

SUMÁRIO

1. INTRODUÇÃO

O **FORTRAN** (*FORmula TranSLATION*) é uma linguagem de alto nível, desenvolvida entre 1954 e 1958 por John Backus e colaboradores. Como o próprio nome diz, ele permite uma tradução quase direta de fórmulas, por meio de uma simbologia de variáveis e operadores algébricos, sendo assim por excelência uma linguagem voltada para problemas que possam ser formulados matematicamente, em particular nos campos da física, da engenharia, da estatística e da própria matemática. Apesar de ele permitir a elaboração de códigos extremamente sofisticados para a resolução de problemas de grande complexidade, o amplo sucesso do FORTRAN nos meios acadêmicos e científicos deve-se ao uso de uma terminologia simples (em inglês) - OPEN, READ, STOP, etc - aliada à codificação de cima para baixo (top-down approach), linha por linha, aproximando-se bastante do procedimento manual para a resolução desses problemas. Assim, a idéia é expressar de maneira simples o problema a ser resolvido. Vamos começar com um exemplo simples.

Imagine que o nosso problema é fazer a soma de três valores. O primeiro passo é, então, tentar descrever as etapas a serem seguidas para obter essa soma. A etapas a serem seguidas seriam: ler os valores, somar os valores, imprimir seu resultado e finalizar. Vamos agora tentar descrever esse problema em linguagem Fortran, seguindo essas etapas.

```
C23456789                !linha de comentário
      program soma       !inicia o programa chamado soma
      integer k,n,m      !declara que k,n,m são variáveis inteiras
      n=1                !atribui o valor 1 à variável n
      m=3                !atribui o valor 3 à variável m
      k=m+n              !soma n e m, e o resultado é armazenado em k
      print*,k           !escreve na tela a variável k
      stop               !para a execução do programa
      end                !fim do programa soma
```

Como podemos notar acima, uma estrutura básica que inclui o comando program (dá um nome ao programa), declaração de variáveis, cálculo, e os comandos stop e end, é seguida para a elaboração de um programa em Fortran . Além disso, podemos notar que alguns critérios de formatação são usados para editar o programa. O programa pode ser editado em qualquer editor de texto (como, nedit, textedit, etc).

Formatação

Os seguintes critérios devem ser seguidos para se escrever um programa em FORTRAN no modo de formulário fixo:

- colunas 1 a 5: são usadas escrever os rótulos (*label*) ou números de comando. Estes números devem ser inteiros e estar totalmente contido nestas colunas. Não podem se repetir e não precisam estar em ordem crescente. Serão usados para que outros comandos possam identificar aquela linha;

- coluna 6: qualquer caractere diferente de 0 (zero) nesta coluna indica que o que vem a seguir é continuação da linha anterior ou da última linha que não seja um comentário (próximo item); podem existir até 19 linhas de continuação;
- colunas 7 a 72: comandos;
- colunas 73 a 80: ignoradas pelo compilador;
- como as palavras - comando em FORTRAN são todas em inglês, letras acentuadas, cê – cedilha (ç) e til NÃO são aceitas.

Conhecendo, agora, a convenção de colunas utilizada para a elaboração de um programa em FORTRAN, tente editar o programa dado na introdução, em seu editor de texto de preferência. Com o programa já editado a partir da sétima coluna (não esqueça), salve seu programa com a extensão .f (por exemplo soma1.f). O próximo passo é a compilação e execução desse programa.

Compilação e Execução

A compilação de um programa em FORTRAN numa workstation, sob unix é feita no terminal, via teclado, não se usa o mouse. No prompt do terminal digite:

`f77 -o` <nome do programa executável> <nome do programa fonte>.f <enter>

Assim, considerando seu programa soma1.f, a compilação será:

`f77 -o` soma1.exe soma1.f <tecle enter>

Quando você executar o comando de compilação irá aparecer no prompt a seguinte mensagem:

soma1.f:
MAIN soma:

Se houver erro em seu programa, aparecerá mensagem de erro no terminal e não será criado o executável.

A execução será realizada a seguir. No prompt do terminal digite:

<nome do programa executável> <enter>

ou seja,

soma1.exe <tecle enter>

Agora que já conhece os comandos de compilação e execução faça um teste. Para verificar isto, inclua em seu programa um erro. Você pode introduzir um outro operador além da soma para gerar esse erro (k=*n+m). Agora é só compilar novamente o programa e verá a mensagem de erro.

Seguindo nosso exemplo 1 (soma1.f), alguns conceitos básicos são apresentados, a seguir. Eles referem-se a estrutura básica.

2. CONCEITOS BÁSICOS

1. *Comentários*: não são interpretados pelo computador; um bom programa deve conter muitos para que fique o mais claro possível. Em FORTRAN a letra c ou os caracteres * e ! na primeira coluna indicam que toda a linha é um comentário. Na

linha de comentário é permitido o uso de qualquer caracter, especial ou não. Note que no programa soma1.f, a primeira linha que define as colunas inicia-se com a letra c. Muitos comentários também são colocados em cada linha, após o caracter !.

2. *Constantes*:
   - Numéricas: podem conter qualquer valor real, inteiro ou complexo. A parte decimal é separada da inteira por um ponto'.'. Os zeros antes e depois do ponto decimal podem ser omitidos, se não forem significativos. O expoente decimal é indicado pela letra '*e*' ou '*E*', e deve vir entre o número e seu expoente sem o uso de espaços entre eles. Números negativos assim como a parte exponencial quando for negativa deve vir precedida do sinal menos '-'. O sinal '+' é opcional em ambas as partes. Os números imaginários devem vir entre parênteses e a parte real deve ser separada por uma vírgula da parte imaginária.

     Ex: 0 , -5 , 4 (inteiras)
     2.5 , 1.0E+05 , -2.55E-02 (reais)
     5+2i $\Rightarrow$ (5.0,2.0) (complexa)

   - Alfanuméricas: (são as '*strings*', seqüências de letras e/ou números) podem conter qualquer seqüência de caracteres. Deve vir entre aspas " " ou apóstrofos ' '. As aspas têm preferência sobre os apóstrofos; portanto, um valor literal pode conter apóstrofos, desde que seu valor venha entre aspas. Não é permitido o uso de caracteres especiais e letras acentuadas.

3. *Maiúsculas e Minúsculas*: o FORTRAN não faz qualquer distinção entre letras maiúsculas e minúsculas. É permitido inclusive o uso do nome da variável escrita de formas diferentes no mesmo programa. EX.: VAR = var = Var.

4. Os programas podem conter no início o seu nome (*program* nome_do _programa), e devem terminar com a palavra '*end*'. Outra forma de parar o programa, e esta pode ser usada em qualquer parte dele, é usando a palavra '*stop*'. O programa terminará independentemente de haver mais comandos na seqüência.

COMANDO STOP – termina a execução do programa. Embora não seja comum, ele pode aparecer no meio do programa.

COMANDO END - indica o final do programa devendo ser obrigatoriamente a sua última linha.

COMANDO PAUSE – interrompe temporariamente a execução do programa, que é retornada a partir daquele ponto assim que o usuário fornecer um comando adequado.
Esse comando pode ser incluído no exemplo anterior para verificarmos seu uso.

```
C23456789                  !linha de comentário
      program soma         !inicia o programa chamado soma
      integer k,n,m        !declara que k,n,m são variáveis inteiras
      n=1                  !atribui o valor 1 à variável n
      m=3                  !atribui o valor 3 à variável m
      pause ´ digite a letra c para continuar ´
      k=m+n                !soma n e m, e o resultado e armazenado em k
      print*,k             !escreve na tela a variável k
      stop                 !para a execução do programa
      end                  !fim do programa soma
```

COMANDO PARAMETER

Este comando define um nome para uma constante. A diferença entre um valor com '*parameter*' e uma variável comum é que com '*parameter*' o valor não pode ser modificado em nenhuma parte do programa ou ser lido através de um comando de leitura 'read'. Vamos então introduzir o comando parameter em nosso programa soma1.f e definir um segundo programa (soma2.f)

```
C23456789
      program soma         !inicia o programa chamado soma
      parameter(raio=3)    !define um nome simbólico para uma constante
      integer k,n,m        !declara que k,n,m são variáveis inteiras
      integer pi           !declara que o valor de raio é inteiro
      n=1                  !atribui o valor 1 à variável n
      m=3                  !atribui o valor 3 à variável m
      k=m+n+pi             !soma n , m e raio, e o resultado é armazenado em k
      print*,k             !escreve na tela a variável k
      stop                 !para a execução do programa
      end                  !fim do programa soma
```

Um outro programa considerando o comando parameter, é mostrado abaixo (raio.f)

```
c O programa calcula a area do circulo com raio r
      program circle
      real r, area, pi
      parameter (pi = 3.14159)
      write (*,*) 'Give radius r:' !escreve na tela o valor de r
      read  (*,*) r                !lê o valor de r
      area = pi*r*r                !calcula a área e atribui o valor em area
      write (*,*) 'Area = ', area
      stop
      end
```

Como já vimos acima, sempre declaramos que tipo de variáveis estamos utilizando. Assim, uma descrição dos tipos de variáveis é feita. Em seguida são apresentados os operadores aritméticos.

3. DECLARAÇÃO DE VARIÁVEIS

As variáveis podem ser inteiras, reais, alfanuméricas ou literais. A declaração de uma variável deve vir antes que ela seja usada, através dos comandos de especificação explícita, tais como: integer, real, complex, character. Se isto não ocorrer, o compilador assumirá as variáveis que começam com as letras I até N como inteiras (*INTEGER*4)* e todas as outras como reais (*REAL*4)*. Isso é chamado declaração implícita. Quando não se deseja declarar implicitamente nenhuma variável usa-se o comando *"implicit none"* .

Para se declarar variáveis que sejam matrizes e vetores deve-se indicar suas dimensões logo após o nome da variável; entre parênteses, e separadas umas das outras por vírgula. Ex.: a(4,3) indica uma matriz a de 4 linhas por 3 colunas.

Tipos de Variáveis:

Inteiras (INTEGER): são aquelas às quais se atribuem valores inteiros.

Reais (REAL): são aquelas às quais se atribuem valores reais.

Alfanuméricas (CHARACTER): são as palavras ou expressões
CHARACTER NOME*w - w representa o número máximo de caracteres que a variável pode conter dentro do programa.

Lógicas (LOGICAL): As variáveis lógicas são "nomes" aos quais se atribuem valores de constantes lógicas. Assim, podem assumir os valores .TRUE. (VERDADEIRO) ou .FALSE. (FALSO) , ou somente T e F.

4. OPERADORES

Atribuição

A variável ou identificador que estiver à esquerda do sinal de atribuição '=' recebe o valor da expressão, constante ou variável que estiver à direita. Veja que em nosso exemplo (soma1.f) as variáveis m, n e k recebem valores através desse operador.

Operadores Literais

Uma função útil para variáveis literais é a concatenação, ou a junção de duas ou mais palavras. Em FORTRAN a concatenação é feita pelo operador '//'.

Ex:

```
a = 'meteor'
b = 'ologia'
c = a//b          => c = 'meteorologia'
```

Operadores Aritméticos

Executam operações aritméticas comuns.

FORTRAN	Matemática Tradicional	Significado
+	+	Soma
-	-	Subtração
*	X	Multiplicação
/	÷	Divisão
**	a^p	Potenciação

Dois operadores não podem aparecer lado a lado. Ex: A*-B(errado) ; A*(-B) (certo).
Quando uma variável inteira recebe o resultado de uma divisão com resto, este resto é desprezado ou seja o valor é truncado. Alguns exemplos de expressões aritméticas são:

```
C = A**2 + B**2
D = E**(1/2)
q=(a+b)/(c+d)
```

Uma observação importante, é que o quociente entre dois números inteiros resulta sempre num inteiro mais próximo; igual ou menor ao valor verdadeiro, pelo processo de truncamento puro e simples. Assim, quando consideramos uma expressão do tipo D=E**(1/2), o resultado da divisão (1/2) será 0 (zero). Para contornar esse problema, escrevemos os valores da divisão como reais, acrescentando um ponto após cada número da divisão; a expressão torna-se:

```
D = E**(1./2.)
```

Operadores Relacionais

Comparam variáveis, constantes ou expressões e retornam '.TRUE.' ou '1' se a comparação for verdadeira, '.FALSE.' ou '0' se a comparação for falsa.

FORTRAN	Matemática Tradicional	Significado
.LT.(less than)	<	MENOR QUE
.LE.(less than or equal to)	≤	MENOR OU IGUAL QUE
.EQ.(equal to)	=	IGUAL A
.NE.(not equal to)	≠	DIFERENTE DE
.GT.(greater than)	>	MAIOR QUE
.GE.(greater than or equal to)	≥	MAIOR OU IGUAL QUE

Exs: A.NE.B NOME.EQ.'JOAO' B**2-4.*A*C .LT. 0.

Operadores Lógicos

São usados quando é necessário mais de uma condição relacional ou quando é preciso inverter seu resultado. Alguns exemplos da estrutura básica são mostrados.

FORTRAN	Significado
.AND.	Junção
.OR.	Disjunção
.NOT.	Negação

Ex: L1 .AND. L2 ; (A .LT. B) .AND. (C .GT. B)

Dois operadores lógicos e/ou relacionais não podem aparecer lado a lado
X .NOT. .EQ. Y (errado) ; X .NE. Y (certo)

Prioridade

Operador	Prioridade
**	1ª
*	2ª
/	2ª
+	3ª
-	3ª
.EQ.	4ª
.NE.	4ª
.GT.	4ª
.GE.	4ª
.LT.	4ª
.LE.	4ª
.NOT.	5ª
.AND.	6ª
.OR.	7ª

O uso de parênteses pode ser feito para trocar a ordem de prioridade. As expressões são calculadas de dentro para fora.

```
                (5º) (6º)      (2º)       (4º)       (8º)
                  ↑   ↑         ↑          ↑          ↑
Ex:      (A + B - C * ( X - Y **2 ) / 3.0 ) * Z + W
                       ↓         ↓          ↓
                      (3º)      (1º)       (7º)
```

Com o conhecimento dos tipos de variáveis e operadores aritméticos podemos incorporar novos comandos ao nosso programa inicial. Tente incluir em seu primeiro programa outras operações. No exemplo abaixo, a soma de variáveis reais e a concatenação de strings são incluídas (exemplo3, soma3.f).

```
C23456789
      program soma              !inicia o programa chamado soma
      integer k,n,m             !declara que k,n,m são variáveis inteiras
      real a,b,c                !declara que as variáveis a,b,c são reais
      character*2 estado        !declara que estado é string com 2 caracteres
      character*3 codigo        !declara que estado é string com 3 caracteres
      character*5 tudo          !declara que estado é string com 5 caracteres
      n=1                       !atribui o valor 1 à variável n
      m=3                       !atribui o valor 3 à variável m
      k=m+n                     !soma n e m, e o resultado é armazenado em k
      a=2.5                     !atribui o valor 2.5 a variável a
      b=1.0                     !atribui o valor 1.0 a variável b
      c=a+b                     !soma a e b, e o resultado é armazenado em c
      print*,k,c                !escreve na tela a variável k e c
      estado = ´SP´             ! atribui o string SP à variável estado
      codigo=01a                ! atribui o string 01a à variável código
      tudo= estado//codigo      !concatena estado e código  e atribui a tudo
      print*, tudo              !escreve na tela a variável tudo
      stop                      !para a execução do programa
      end                       !fim do programa soma
```

O exemplo seguinte (equacao.f) faz a resolução de uma equação de 2º.

```
C234567
C este programa calcula equacao de segundo grau
      program equacao           !inicia o programa equação
      integer b,a,c             !declara que as variáveis a,b,c são inteiras
      real delta
      a=4
      b=5
      c=1
      delta=b**2-4.0*A*C        !calcula a equação e resultado é atribuído a delta
      write(*,*)delta           !escreve na tela o valor de delta
      stop
      end
```

Tente, agora, montar alguns programas simples.

5. FUNÇÕES INTRÍNSECAS

Existem várias funções predefinidas em FORTRAN, que podem ser usadas em qualquer parte do programa. Aqui algumas dessas funções são apresentadas.

Funções Trigonométricas

Nome	Definição	Tipo de Argumento	Tipo da Função
SIN (x)	Seno (radianos). Se x for complexo, a parte real é assumida como valor em radianos.	Real ou complexo.	REAL*4
ASIN (x)	Arcoseno (radianos). Retorna valores na faixa [-π /2, π /2]	Real, |x| .le. 1	REAL*4
COS (x)	Coseno (radianos) Se x for complexo, a parte real é assumida como valor em radianos.	Real ou complexo	REAL*4
ACOS (x)	Arcocoseno (radianos)) Retorna valores na faixa [0, π]	Real, |x| .le. 1	REAL*4
TAN (x)	Tangente (radianos)	Real	REAL*4
ATAN (x)	Arcotangente (radianos). Retorna valores na faixa [-π /2, π /2]	Real	REAL*4
SINH (x)	Seno Hiperbólico (radianos)	Real	REAL*4
COSH (x)	Coseno Hiperbólico (radianos)	Real	REAL*4
TANH (x)	Tangente Hiperbólica (radianos)	Real	REAL*4

Outras Funções

Nome	Definição	Tipo de Argumento	Tipo da Função
ALOG10 (x)	Logaritmo de x na base 10	Real	Real
ALOG (x)	Logaritmo neperiano de x (x > 0)	Real	Real
EXP (x)	o número e (base dos logaritmos neperianos) elevado a x	Real	Real
ABS (x)	valor absoluto de x	Real	Real
IABS (x)	valor absoluto de x	Inteiro	Inteiro
IFIX (x)	Conversão de real para inteiro, truncando	Real	Inteiro
FLOAT (x)	Conversão de inteiro para real	Inteiro	Real
DBLE (x)	Converte para dupla precisão	Real	Real (dupla precisão)
CMPLX (x)	Converte para o tipo complexo	Real	Complexo

SIGN (x,y)	Fornece valor positivo de x se y ≥ 0 e negativo de x se y < 0	Real	Real
MOD (x,y)	resto da divisão de x por y	Inteiro	Inteiro
AMOD (x,y)	resto da divisão de x por y	Real	Real
SQRT (x)	raiz quadrada de x (x ≥ 0)	Real	Real

Como usá-las? Nos exemplos a seguir mostramos dois exemplos simples. Um calcula o valor de π (calcula_pi.f) e o outro a raiz quadrada de um número qualquer (sqrt.f).

```
C234567
c     Calcular o número pi.
c     Pi é o arco cujo cosseno é -1.
      program p_calcpi
      real pi
      pi=acos(-1.0)            !calcula o valor de pi utilizando a função acos
      write(*,*) 'Pi = ',pi
      stop
      end
```

```
C Este programa calcula a raiz quadrada de um numero
c utilizando a funcao intrinseca SQRT
      program funcoes
      implicit none
      real x, y
      print *,'entre com um numero real'
      read *, x
      y=sqrt(x)           !calcula a raiz quadrada de x e armazena em y
      print *,'a raiz quadrada do numero dado ', y
      stop
      end
```

6.COMANDOS DE CONTROLE

Como já foi dito, uma das características da linguagem FORTRAN é o processamento de cima para baixo, linha por linha. Entretanto, essa ordem pode ser alterada quando utilizamos algumas condições para que os cálculos sejam realizados. Isso pode ser feito utilizando o comando IF.

Estrutura do IF

O IF ("se") é utilizado quando queremos optar por várias alternativas dependendo do resultado ou do dado fornecido. Há duas formas de sintaxe mais comuns:

IF com uma alternativa (estrutura condicional simples) – A sintaxe do comando é:

```
IF (condição) then
    Bloco 1
end if
```

Neste caso, se a condição for verdadeira, o bloco1 é executado. Caso seja falsa, ele pula e prossegue com o programa. Vamos voltar ao nosso primeiro exemplo e incluir uma condição para que ele execute a soma (exemplo, soma4.f).

```
C23456789
      program soma          !inicia o programa chamado soma
      integer k,n,m         !declara que k,n,m são variáveis inteiras
      n=1                   !atribui o valor 1 à variável n
      m=3                   !atribui o valor 3 à variável m
      if(m .ne. 0)then      !se o valor de m for diferente de zero, então
      k=m+n                 !soma n e m, e o resultado e armazenado em k
      endif                 !finaliza o bloco
      print*,k              !escreve na tela a variável k
      stop                  !para a execução do programa
      end                   !fim do programa soma
```

IF com duas alternativas **(estrutura condicional composta) -** Aqui, se a condição for verdadeira, o bloco1 é executado; se não, o bloco2 é que será executado. A sintaxe do comando é:

```
IF (condição)then
  Bloco1
else
  Bloco2
end if
```

Como ficaria agora nosso exemplo? (soma5.f)

```
C23456789
      program soma          !inicia o programa chamado soma
      integer k,n,m         !declara que k,n,m são variáveis inteiras
      n=1                   !atribui o valor 1 à variável n
      m=3                   !atribui o valor 3 à variável m
      if(m .ne. 0)then      !se o valor de m for diferente de zero, então
      K=m+n                 !soma n e m, e o resultado é armazenado em k
      else                  !se a condição não for verdadeira
      k=n                   !o valor de n é atribuído a k
      endif                 !finaliza o bloco
      print*,k              !escreve na tela a variável k
      stop                  !para a execução do programa
      end                   !fim do programa soma
```

IF lógico simples **-** executa ou não o comando a sua frente (na seqüência, na mesma linha) dependendo de sua condição. Este comando pode ser de qualquer tipo, atribuição, escrita, leitura (teste.f). A sintaxe nesse caso é:

```
IF (condição) comando
```

```
C234567
      program teste
      real x,y
      x=5
      y=4
      if(x .gt. y) y=x    !se valor de x  maior que y, o valor de x é atribuído a y
      print*,x,y
      stop
      end
```

IF aritmético

Uma maneira de se deslocar num programa, usando uma expressão no lugar de uma variável é a seguinte:

```
IF (exp. Numérica) r1,r2,r3
```

Exp. numérica não pode ser complexa. r1, r2, r3 são rótulos. Se o valor de 'exp. numérica' for menor que zero o comando vai para a linha com o rótulo (label) r1, quando for igual a zero o comando vai para linha com o rótulo r2 e quando for maior que zero vai para r3. Retornando ao exemplo equacao.f , e podemos incluir o comando if aritmético (exemplo equacao1.f)

```
C234567
C este programa calcula equacao de segundo grau
      program equacao          !inicia o programa equação
      real b,a,c,x, x1,x2      !declara que as variáveis a,b,c são inteiras
      real delta
      a=4
      b=5
      c=1
      delta=b**2-4.0*A*C       !calcula a equação; resultado é atribuído a delta
      write(*,*)delta          !escreve na tela o valor de delta
      if (delta) 10, 20, 30    !se delta menor que zero executa a linha com
c                               label 10, se delta igual a zero executa a linha
c                               com label 20, se delta maior que zero executa a
c                               linha com label 30
  10  print*,´delta < 0 ; no real roots´
      go to 100                !a execução é desviada para o comando stop
  20  x=-b/(2.0*a)
      print*,´delta = 0 ; two equal real roots´
```

```
      print *,´               x1=x2= ´,x
      go to 100               !a execução é desviada para o comando stop
  30  sd=sqrt(delta)
      x1=0.5*(-b-sd)/a
      x2=0.5*(-b+sd)/a
      print*,´delta > 0; two different real roots ´
      print*,´                x1= ´,x1
      print*,´                x2= ´,x2
100   stop
      end
```

Além do comando If aritmético, neste programa nós usamos, pela primeira vez, a formatação dos label. Note que os números 10, 20 e 30 estão definidos entre as colunas de 1 a 5. O uso dos apóstrofos também foi incluído quando desejamos escrever na tela uma mensagem.

IF expandido **–** neste caso, várias condições são testadas. A estrutura a sintaxe é:

```
IF (condição ) then
  bloco1
else if (condição2) then
  bloco2
   :
else if (condição n) then
  blocon
else
  bloco (n+1)
end if
```

Um exemplo neste caso pode ser visto abaixo (notas.f)

```
c Exemplo de programa usando o comando IF em sua forma
c expandida
c23456
      program resultadonota
      implicit none      !todas as variáveis devem ser declaradas explicitamente
      real nota          !declara a variável nota como real
      print *, 'qual foi a nota da sua prova?' !escreve na tela a nota
      read *, nota       !lê a nota
      if (nota .ge. 9.00) then  !se a nota for maior que 9, então executa o
c                                comando a seguir; se não, testa a nova condição
      print *,'muito bem!'
      else if (nota .ge. 7.00) then
      print *,'voce foi bem!'
      else if (nota .ge. 5.00) then
      print *,'voce tem que melhorar um pouco!'
      else
```

```
      print *,'farelo!!!!!!'
      end if                    !finaliza o bloco if
      stop
      end
```

Um segundo exemplo é rescrevermos o programa equacao1.f utilizando a estrutura de if estendido (equacao2.f).

```
C234567
C este programa calcula equacao de segundo grau
      program equacao           !inicia o programa equação
      real b,a,c,x, x1,x2       !declara que as variáveis a,b,c são inteiras
      real delta
      a=4
      b=5
      c=1
      delta=b**2-4.0*A*C        !calcula a equação; resultado é atribuído a delta
      write(*,*)delta           !escreve na tela o valor de delta
      if (delta .lt. 0)then      !se a condição1 for verdadeira, então executa a
c                                linha abaixo e finaliza o bloco if

      print*,´delta < 0 ; no real roots´

      else if(delta .eq. 0)then     !se a condicao1 for falsa a condição2
c                                   é testada; se for verdadeira, executa os
c                                   comandos das linhas seguintes e finaliza.
c                                   Se for falsa, uma condição3 é testada
      x=-b/(2.0*a)
      print*,´delta = 0 ; two equal real roots´
      print *, ´          x1=x2= ´,x

      else if (delta .gt. 0)then   !testa a terceira condição
      sd=sqrt(delta)
      x1=0.5*(-b-sd)/a
      x2=0.5*(-b+sd)/a
      print*,´delta > 0; two different real roots ´
      print*,´                x1= ´,x1
      print*,´                x2= ´,x2
      endif                     !finaliza o bloco if
      stop
      end
```

COMANDO DO (faça)

Quando o mesmo comando precisa ser executado várias vezes até que se atinja uma certa condição ou um número certo de repetições, o melhor é usar as estruturas de repetição. Estas estruturas são bem simples e podem economizar várias linhas de comando.

Laços simples

```
      do  i = ibeg,iend,incr
       (comandos a serem executados)
      end do
```

'i' é uma variável inteira que recebe inicialmente o valor 'ibeg', a seqüência de comandos se repete, e o valor de 'i' aumenta de 'incr' a cada vez que o comando volta para a linha do 'do'. A repetição só para quando o valor de 'i' é maior que 'iend'.
A palavra chave 'end do' pode ser escrita como 'enddo'. 'ibeg', 'iend' e 'incr' podem ser constantes ou variáveis inteiras ou reais, positivas ou negativas. Quando 'incr' for igual a 1 ele pode ser omitido. O exemplo a seguir pode ajudar a entender o comando do (adiciona.f).

```
C234567
c     somar os números inteiros de 1 a 100.
      program p_add100
c     inicializa um acumulador.
      ia=0          !atribui o valor zero para variável que irá acumular os valores
      do ic=1,100,1       !para os valores de 1 a 100 com intervalo de 1, faz
         ia=ia+ic         !adiciona o valor de ic à soma (ia)
      end do              !encerra o loop
      write(*,*) ia
      stop
      end
```

Outra forma para encerrar um laço do comando DO , é utilizando o comando CONTINUE. A estrutura básica do comando é:

```
          do n i = 1,10
             (comandos do bloco)
      n   continue
```

"n" label (número inteiro) referente ao comando a CONTINUE que identifica o final da seqüência de comandos a ser executada várias vezes. O exemplo anterior utilizando o comando continue será (adiciona1.f):

```
c234567
c     somar os numeros inteiros de 1 a 100.
      program p_add100
c     inicializa um acumulador.
      ia=0          !atribui o valor zero para variável que irá acumular os valores
      do 10 ic=1,100,1    !para os valores de 1 a 100 com intervalo de 1, faz
         ia=ia+ic         !adiciona o valor de ic à soma (ia)
  10  continue            !enquanto ic <100, o comando do é executado
      write(*,*) ia
      stop
      end
```

Outro Exemplo: (contagem.f)

```
c este programa faz contagem regressiva a partir de um numero
fornecido
      program contagem    ! início do programa
      implicit none       ! todas variáveis devem ser declaradas explicitamente
      integer interv, tempo, inicio      ! declara as variáveis do tipo inteiras
      parameter (interv=-1)   ! a constante -1 é representada pela variável interv
      print *, 'entre com um numero inteiro'   ! escreve na tela tudo
c                                                que estiver entre aspas
      read *, inicio         ! lê o numero digitado e o atribui a variável inicio
      print *, 'começou a contagem regressiva'
      do 10 tempo=inicio, 1, interv    ! executa o comando dentro do laço
c                                        até que a variável inicio seja igual a l
      print *, tempo
   10 continue
      print *, 'bummmm!!'
      stop
      end                      ! fim do programa
```

Laços aninhados

A situação em que temos um DO dentro de outro DO é chamado de laços aninhados, ou seja, o laço interno será executado o número de vezes determinado pelo laço externo. Este tipo de estrutura é fundamental quando trabalhamos com matrizes. No exemplo abaixo (contagem1.f) o comando no interior do laço será executado 50 vezes.

```
      do i=1,10                    !(externo)!
         do j=1,50                 !(interno)!
            (comando)
         end do                    !(interno)!
      end do                       !(externo)!
```

```
c este programa permite observar quando está sendo!
c executado o primeiro e o segundo do!
      program loop
      implicit none
      integer m, n
      parameter (m=3, n=2)
      integer i, j
      print *,' i j '
      do 10 i=1,m                  ! inicio do laço externo
      print *, 'primeiro do ', i
      do 20 j=1,n                  ! inicio do laço interno
      print *, 'segundo do ',i,j
      20 continue                  ! fim do laço interno
      10 continue                  ! fim do laço externo
      stop
      end
```

COMANDO "DO WHILE"

Este comando é utilizado para executar um bloco repetidas vezes enquanto a condição for verdadeira. A partir do momento em que a condição passa a ser falsa, este sai do laço continuando a execução do restante do programa. a sintaxe do comando é:

```
DO WHILE (condição)
 (comandos do bloco)
END DO
```

O programa abaixo mostra a utilização do comando DO WHILE (rad.f).

```
!Este programa calcula o nivel de radiação e o grau de segurança!
!o programa utiliza o comando do while, que é utilizado para
!executar um bloco repetidas vezes enquanto a condição for
!verdadeira. A partir do momento em que a condicao passa a ser
!falsa, este sai do loop continuando a execucao do restante do
!programa.
      program radiacao
      implicit none           !todas as variáveis devem ser declaradas explicitamente
      real radseg, fatseg, nivrad, radmin ! declaração das variáveis do tipo
reais
      parameter (radseg=0.466, fatseg=10.0) !atribui nomes às variáveis
      integer dia                  ! declaração das variáveis do tipo inteira
      dia=0
      print *,'entre com o nivel de radiação do dia'
      read *, nivrad
      print *, 'n.dias radiacao '
      radmin=radseg/fatseg
      do while (nivrad .gt. radmin) ! executa as instruções dentro do laço ate que
a
c                                   condição seja satisfeita
      if (nivrad .gt. radseg) then
      print *,dia,nivrad,' inseguro'
      else
      print *, dia, nivrad, ' seguro'
      end if
      dia=dia+3
      nivrad=nivrad/2.0
      end do                       ! fim do comando do while
      end
```

---

7. COMANDOS E FORMATOS DE ENTRADA E SAÍDA

Os comandos de entrada e saída, na sua forma mais simplificada, possuem a seguinte estrutura:

COMANDO READ – permite a entrada de dados via teclado ou de um arquivo em disco.

read (unidade, formato) lista_de_parâmetros

- unidade: se número inteiro, especifica o canal (arquivo em disco)
  se * ,indica entrada de dados via teclado
- formato: se número inteiro, especifica rótulo de formato
  se * ,indica formato livre

```
c23456
      program ler
      integer a,b,c
      print*,entre com os valores de a b c´
      read(*,*)a,b,c
      sum=a+b+c
      stop
      end
```

COMANDO WRITE – permite a saída de dados para o monitor ou para um arquivo em disco.

write (unidade, formato) lista_de_parâmetros

As mesmas especificações para unidade e formato, dadas acima, são utilizadas para o comado write. Podemos notar que, nos exemplos utilizados anteriormente, esses comandos eram expressos considerando a entrada de dados via teclado e saída na tela. Ou seja, nossa unidade e formato eram definidos por ´*´.

```
c23456
      program escrever
      integer a,b,c
      print*,entre com os valores de a b c´
      read(*,*)a,b,c
      sum=a+b+c
      write(*,*)a,b,c,sum
      stop
      end
```

Uma outra maneira de escrever os arquivos de saída é através do comando "PRINT' . Esse comando permite a saída de dados exclusivamente para o monitor. Sua sintaxe é:

Print formato, lista_de_parâmetros

Onde lista_de_parâmetros representa os dados que serão impressos, e devem vir separados por vírgula. esta lista pode conter variáveis ou expressões alfanuméricas; estas últimas devem vir entre apóstrofos ' '.

```
c23456
      program ler
      integer a,b,c
      print*,entre com os valores de a b c´
      read(*,*)a,b,c
      sum=a+b+c
      print*,´sum =´,sum
      stop
      end
```

As unidades ‘6’ e ‘*’ se não forem definidas dentro do programa, serão consideradas como o terminal (‘*write*’ ou ‘*print*’). Da mesma forma as unidades ‘5’ ou ‘*’ são definidas como o teclado (‘*read*’). O comando ‘print’ imprime sempre os resultados na unidade definida como o terminal.

CANAIS DE ENTRADA E SAÍDA

- Teclado e monitor

No comando READ(*,formato), os dados de entrada devem ser digitados no teclado, conforme a especificação do formato, se for o caso; e separados por vírgula ou espaço em branco, para formato livre.

Nos comandos WRITE(*,formato) e PRINT, os dados serão exibidos no monitor conforme a especificação de formato, se for o caso; ou em formato aleatório, escolhido pelo computador, para formato livre.

O exemplo a seguir calcula a média e desvio padrão a partir de dados fornecidos via teclado (array.f)

```
c este programa calcula a media e o desvio padrao de uma
c serie de dados fornecidos a partir do teclado
      program conjloop
      integer maxitm
      parameter (maxitm = 8)
      integer i
      real x(maxitm), averag, stdev, sum, sumsqr
      print *, 'entre', maxitm, 'numeros, um por linha:'
      do 10 i=1, maxitm
      read *, x(i)
   10 continue
      sum = 0.0
      sumsqr = 0.0
      do 20 i=1,maxitm
      sum = sum + x(i)
      sumsqr = sumsqr + x(i) ** 2
   20 continue
      averag=sum/real(maxitm)
      stdev=sqrt(sumsqrt/real(maxitm) - averag ** 2)
```

```
      print *
      print 15, 'o valor medio e', averag
      print 15, 'o desvio padrao e', stdev
   15 format (1x, a, f8.1)
      print *
      print *, 'tabela de diferença entre x(i) e a media'
      print 25, 'i', 'x(i)', 'diferença'
   25 format (1x, a4, 3x, a8, 3x, a14)
      do 30 i=1, maxitm
      print 35, i, x(i), x(i)-averag
   35 format (1x, i4, 3x, f8.1, 3x, f14.1)
   30 continue
      stop
      end
```

- ***Arquivos em disco***

Como especificado anteriormente, o 1º argumento nos comandos READ ou WRITE especifica o canal de entrada/saída. Assim, se a "unidade" for um número (inteiro), é feita uma associação entre a lista de variáveis a ser lida e o nome do arquivo correspondente através dos comandos OPEN e CLOSE.

COMANDO OPEN (abrir)

OPEN (UNIT=número da unidade,FILE=´nome_arquivo´,STATUS=´tipo_arquivo´)

Ex:

OPEN (UNIT=1,FILE=´dados´,STATUS=´OLD´)
OPEN (UNIT=2,FILE=´dados´,STATUS=´NEW´)

- O número da unidade deve ser um número inteiro não associado a um outro arquivo.
- O tipo de arquivo deve ser 'NEW' para um arquivo de saída novo , 'OLD' para um arquivo já existente, ou 'UNKNOWN' para arquivo desconhecido. Em geral não se usa o NEW porque os compiladores dão mensagem de erro quando o arquivo já existe e você quer ignorar o que foi escrito anteriormente nesse arquivo. Então se usa o UNKNOWN para arquivos novos também.

COMANDO CLOSE (fechar)

Um arquivo pode também ser fechado. Isto fará com que o FORTRAN coloque uma marca de fim de arquivo naquele ponto, esta marca pode ser identificado por outro comando ou função .

close (unidade,**status=**'estado')
ou
endfile unidade

Onde status='estado' é opcional. Estado pode ser 'keep' que mantém o arquivo na memória (esta é a opção assumida quando status='estado' é omitida), ou 'delete' que apaga o arquivo da memória. Arquivos fechados podem ser reabertos em qualquer parte do programa. O exemplo a seguir (ler_arq.f) mostra a leitura e escrita de arquivos em disco.

```
c234567
      program ler_arq
c   este programa le series temporais
      real var1(104),var2(104),x(104),y(104),z(104)
c*************************************************************
c                        arquivos de entrada
c*************************************************************
      open(10,file='serie1.asc',status='old')
      open(11,file='serie2.asc',status='old')
c*************************************************************
c                    arquivo de saida
c*************************************************************
       open(12,file='soma.asc',status='new')
c*************************************************************
c                     leitura de dados
c*************************************************************
         do l=1,104
           read(10,*)var1(l)   !ler a var1 contida no arq referente a unidade 10
           read(11,*)var2(l)   !ler var2 contida no arq referente na unidade 11
            x(l)=var1(l)       !atribui o valor da variável para x
            y(l)=var2(l)       !atribui o valor da variável para y
            z(l)=x(l)+y(l)
          write(12,*)z(l)      !escreve o arq_saida na unidade 12

      enddo
      close(10)
      close(11)
      close(12)
      stop
      end
```

FORMATOS

A maioria dos exemplos mostrados consideram que os arquivos de entrada e saída têm uma formatação livre, definida pelo caracter *. É um recurso útil para se evitar erros. Os formatos servem para que os dados sejam impressos ou lidos de uma forma especifica, determinado pelo programador. Os formatos são compostos por uma seqüência de especificações que determinarão como os dados serão processados. Cada uma dessas especificações devem vir separadas por vírgulas. Pode-se ainda imprimir constantes numéricas e alfanuméricas, sendo que esta última deve vir entre apóstrofos ‘ ‘.

A forma de se declarar os formatos é a seguinte:

r **format** (especificação1, especificação2, ...)

Onde r é um numero inteiro, e representa o rótulo do ‘format’. Um mesmo ‘format’ pode ser usado por vários comandos de escrita e leitura. Strings devem vir entre apóstrofos duplos (‘’string’’) nesse formato.

Formato	**Uso**
Im m: número de posições reservadas, inclusive para o sinal	Valores Inteiros Ex: valor formato 12345 I5
Fm.n m: número total de posições reservadas, incluindo o ponto decimal n: número de casas decimais	Valores Reais Ex: 1234.5678 F9.4
Em.n m: número total de posições, inclusive o sinal e o ponto decimal da mantissa, símbolo E, e o sinal e os dois dígitos do expoente n: número de casas decimais da mantissa.	Valores Reais com expoente Ex: 0.1234E+04 E10.4
mX m número de espaços em branco	Deixa m espaços em branco
A[m] m: número de caracteres	Seqüência de Caracteres Ex: meteorologia A(12)

Caso as strings sejam maiores que o espaço reservado à elas, serão tomados apenas os m primeiros caracteres. Se forem menores, elas serão alinhadas a direita e os outros espaços deixados em branco.

Nome = ‘belo horizonte’	
read (*, ‘(a10)’) nome write (*, ‘(a15)’) nome	=> belo horiz => belo horiz

A seguir, dois exemplos (read.f e radiacao.f, respectivamente) serão apresentados considerando algumas especificações de formato.

---

c23456

```
     program read
c  Este programa le os dados do arquivo "sen_cos.dat" que
c possui formatacao livre e imprime para a tela usando o
c comando FORMAT
      integer nmax, u
      parameter (nmax=1000, u=20)
      real x(nmax), y(nmax), z(nmax)
      open (u, FILE='sen_cos.dat', STATUS='OLD')  !abre o arquivo de c
                                                  dados

      read(u,*) n                  !lê o numero de pontos
```

```
      if (n.GT.nmax) then      !se n>nmax então, executa os comandos do if
      write(*,*) 'Erro: n = ',n,'e maior que nmax =', nmax
      stop                     !para a execução
      endif                    !finaliza o bloco if

      do 10 i= 1, n            !para valores de 1 a n, faça
         read(u,*) x(i), y(i), z(i)      !lê os valores de x, y,z
   10 enddo
      close (u)                !fecha a unidade u

      do 20 i= 1, n            !para valores de 1 a n, faça
         write(*,100) x(i), y(i), z(i) !escreve os valores de x, y, z para
c                                       o formato definido
   20 enddo
  100 format (3(F10.4))        !especificação do formato. Neste caso, o número c
                          3 representa o número de vezes que a
c                                 especificação de formato deve se repetir
      stop
      end
```

```
!programa que mostra um exemplo de formatacao usando o comando
format
      program radiacao
!este programa calcula o grau de contaminação de uma certa!
!area e o tempo necessario para que esta que com um grau!
!de segurança!
      real radseg, fatseg
      parameter (radseg=0.466, fatseg=10.0)
      integer dia
      real nivrad, radmin
      dia=0
      print *,'entre com o nivel de radiação do dia'
      read *, nivrad
      print 15, 'n. dias', 'radiacao', 'condicao'
   15 format (1x,a7,2x,a11,2x,a8)
      radmin=radseg/fatseg
      do while (nivrad .gt. radmin)
      if (nivrad .gt. radseg) then
      print 25, dia, nivrad, 'inseguro'
   25 format (1x, i7, 2x, f11.7, 2x, a8)
      else
      print 35, dia, nivrad, 'seguro'
   35 format (1x, i7, 2x, f11.7, 2x, a8)
      end if
      dia = dia + 3
      nivrad = nivrad/2.0
      end do
      end
```

REFERÊNCIAS BIBLIOGRÁFICAS

Esta apostila foi preparada levando em consideração, o objetivo principal, que é fornecer subsídios à um público sem nenhum conhecimento prévio do assunto; razão pela qual a abordagem é iniciada a partir de conceitos básicos da linguagem FORTRAN, tais como as convenções para a definição de variáveis e as regras para operações aritméticas simples. Além disso, teve-se a preocupação de reunir um conteúdo mínimo que permitisse ao usuário a aquisição de informações suficientes para a elaboração de um programa em FORTRAN de complexidade simples a média. Não obstante, acreditamos que outros tópicos, como, funções e subrotinas, não apresentados nesta apostila, poderão ser úteis na geração de programas mais sofisticados, como também na compreensão de códigos escritos por outros autores. Assim, algumas referências são apresentadas, pois acreditamos que essas poderão servir de complemento ao material reunido nesta apostila, em particular, para aqueles interessados em aplicações mais específicas de programação em FORTRAN.

1. Shibata, C. S. **Fortran Básico:** Notas de aula referentes ao curso organizado pelo setor de treinamento. INPE. Primeira edição, julho de 1996.

Esta apostila serve como guia introdutório ao uso de Fortran, com uma abordagem ampla do assunto. É um material compreensível para um usuário não especializado, e suficientemente bom para a resolução de problemas simples.

2. Farrer, H. ; Becker, C. G.; Faria, E. C.; Campos, F.F.; Matos, H. F.; Santos, M. A.; Maia, M. L**. Programação estruturada de computadores: FORTRAN ESTRUTURADO**. Editora Guanabara Koogan S.A., p.194, 1992.

É um livro bastante claro e didático, e contém vários exemplos e exercícios propostos.

3. Hehl, M. E. **Linguagem de programação estruturada: FORTRAN 77.** Editora McGraw-Hill, 1987.

É um livro não muito didático, indicado para usuários que já possuem algum conhecimento prévio sobre o assunto.

4. Press, W. H. et al. **Numerical Recipes in Fortran: the Art of Scientific Computing**. Cambridge Univ. Press, Cambrigde, 1986.

Livro em nível mais avançado. Traz um breve desenvolvimento teórico do problema, seguido do código numérico correspondente.