Programação em Fortran - introdução

Prof. Paulo de Tarso R. Mendonça, Ph.D.
Universidade Federal de Santa Catarina - UFSC
Depto. Engenharia Mecânica, CP 476, Florianópolis, SC. CEP 88.040-900,
e-mail: mendonca@grante.ufsc.br

14 de Junho de 2013

Resumo

Este é um sumário dos principais comandos de programação Fortran. É um texto introdutório, onde se inclui apenas as operações mais simples. Detalhes sobre argumentos dos comandos são deixados para serem consultados nos manuais dos fabricantes dos compiladores que o leitor esteja usando. Por outro lado, supõe-se que o leitor tenha já feito um curso de um semestre em alguma outra linguagem de programação. Aqui são comentados apenas aspectos comuns a todas as versões de Fortran, o que corresponde ao Fortran 77, exceto por alguns tópicos adicionais característicos das versões 90 e posteriores. Não são consideradas programação gráfica, janelamento, programação orientada a objeto, paralelização, integração com outras linguagens e outras operações.

Conteúdo

1 Tipos de Variáveis e Constantes

São seis os tipos de dados em Fortran:[1]

- **Inteiros**, que podem ser armazenados em 1, 2 ou 4 bytes. Por default os inteiros são armazenados em 4 bytes. O número de bytes usado para representar o número indicam o máximo valor possivel de ser representado. Por exemplo, com 1 byte é possivel representar apenas números de -128 a 127, com 2 bytes números de -32768 a 32767, e 4 bytes os valores vão a pouco mais de 2 milhões. Por default, todas as variáveis iniciadas pelas letras i, j, k, l, m, n, são inteiras. As variaveis iniciadas pelas demais letras são reais.
- **Real**, que podem ser armazenados em 4 bytes, também chamados reais de precisão simples, e em 8 bytes, chamados reais de precisão dupla (double precision). Por default os reais são de precisão simples.
- **Complexos**, de 8 bytes, onde parte real e imaginária são precisão simples, e complexos de 16 bytes, onde cada parte é precisão dupla.
- **Lógico**, de 1, 2 ou 4 bytes. Os valores podem ser 0 (falso) ou 1 (verdadeiro). Mais detalhes nos manuais de fabricante dos compiladores.
- **Caracter**, que armazena um único caracter, ocupando um byte. Permite a formação de palavras usando o comando $Character*n$, onde n é um inteiro entre 1 e 32767, para indicar n caracteres.
- **Arranjos** ("array"). São vetores (um contador de índice) ou matrizes de até 7 contadores.

2 Estrutura dos Programas

Um programa Fortran é composto de um **programa principal** e um conjunto de **subrotinas** e de **functions**. Eventualmente um programa poderia ser construido apenas com o programa principal realizando todas as operações, sem a necessidade de subprogramas. Entretanto é mais eficiente a subdivisão indicada, onde o programa principal é montado de forma a ser o mais enxuto possivel, cuidando apenas do gerenciamento das operações através das chamadas aos subprogramas.

Cada parte da estrutura é delimitada por grupos definidos de comandos. A estrutura geral é a seguinte:

```
Exemplo1
c       Programa Principal
```

[1] Autor: Prof. Paulo de Tarso R. Mendonça, Ph.D., 2004, 2013, Depto.Eng.Mecânica, UFSC, mendonca@grante.ufsc.br.

```
      Program NomedoPrograma
      Implicit real*8 (a-h,o-z)
      Dimension a(3)
          .
c.    corpo do programa principal
          .
      a(1)=1.2
      a(2)=1.3D-3;          a(3)=2*a(2)
      call subrot1(a,b,s)       !  chamada aa subrotina subrot1
          .
c.    mais programa principal
          .
      stop
      end NomedoPrograma                      !  termino do programa principal
c     termino do programa principal
      Subroutine subrot1(x,y,t)
      Implicit real*8 (a-h,o-z)
      Dimension x(3)
          .
c.    corpo da subrotina subrot1
          .
      co=2          !  co eh uma variavel local aa rotina.
      y=x(1)+x(2)
      t=x(3)/co
          .
      return
      end
```

Todos os comandos em fortran são definidos por **linhas de comandos**. Existem dois tipos de formatação de texto, o **formato livre** e o **fixo** (*free* e *fixed*). O tipo de formato deve ser definido previamente à codificação e digitação. No formato fixo, o mais antigo, as linhas devem ser iniciadas em qualquer coluna a partir da sétima, inclusive. O comando deve estender-se apenas até a coluna 72 (ou 80 dependendo de ajuste de parâmentros do compilador). Linhas de comando mais longas podem ser formadas, continuando nas linhas seguintes, desde que as linhas subsequentes possuam um caracter qualquer na sexta coluna. Cada uma das linhas de comando do exemplo acima são contidas em uma única linha. No formato livre as linhas de comando podem se iniciar em qualquer coluna, e normalmente as linhas são ajustadas no compilador para serem mais longas. A continuação é feita com o uso de um caracter ao final de cada linha a ser continuada.

Espaços horizontais entre variáveis não são considerados. Também o Fortran não distingue entre caracteres maiúsculos e minúsculos na redação dos comandos. Os nomes das **variáveis** e das **subrotinas** são compostas por caracteres alfanuméricos, mas devem se iniciar por uma letra. O número máximo de caracteres no nome da variável é arbitrário até 256, porém o compilador distingue apenas até o oitava caracter. Comandos podem ser encadeados ao longo da mesma linha, separados por ";" como no exemplo do programa principal.

Linhas de comentario são identificadas para o compilador pelo caracter **c** na coluna 1, como no exemplo. Comentários também podem ser colocados na própria linha de comando, após um sinal "!" em qualquer coluna.

Todo programa ou subprograma possui um conjunto de comandos que forma um "**cabecalho**". No programa principal do exemplo o cabecalho é formado pelos comandos "implicit "e "dimension". **Implicit** é o comando para alterar o default das variaveis reais, isto é, ele faz com que todas as variaveis iniciadas por letras dentro dos **a-h** e **o-z** sejam reais de precisão dupla, 8 bytes, em vez de simples como no default. De fato, o comando implicit pode ser usado para redefinir as variaveis pela forma sua letra inicial. Por exemplo,

```
implicit real*8 (d-h), integer*2 (i-k), integer*4 (l-n)
implicit real*4 (o-y), complex*16 (a-c), character*5 z
```

Aqui as variaveis iniciadas pelas letras d até h serão todas de dupla precisão, as iniciadas por i, j e k serão inteiros de 2 bytes, etc.

Uma outra forma de identificar as variáveis é de forma individual, usando os comandos `real, integer*2, integer*4` etc., da seguinte forma:

```
integer*2 vigas, N/4, laminas(3)
```

Aqui as variáveis `vigas, N` e `laminas` são definidas como inteiras de 2 bytes. Além disto, os comandos podem ser usados para definir o próprio valor da variavel, no caso `N=4`. Os comandos `real, integer`, etc., também podem ser usados para definir as dimensões dos arranjos. No exemplo temos que `laminas` é um vetor de inteiros de 3 posições.

O segundo comando no programa do exemplo é o **dimension**. Ali está a definição de que a variável `a` é um arranjo, no caso um vetor de 3 posições. Uma matriz `B` de 3 linhas por duas colunas seria definida por exemplo como `dimension B(3,2)`. O termo (1,2) da matriz é referenciado simplesmente como `B(1,2)`.

No exemplo, o programa principal tem seu corpo constituido apenas por uma linha, em que é chamada uma **subrotina**, denominada `SUBROT1`. Esta chamada é sempre feita através do comando `CALL`. Observamos a subrotina em si, listada abaixo do programa principal. Ela deve ser iniciada por um comando `SUBROUTINE` seguida pelo nome dado à subrotina, e os argumentos de passagem. A rotina também pode ter seu cabecalho, definindo os tipos das variáveis. Os argumentos da de uma subrotina são de fato apontadores para os endereços de memória onde são armazenadas as variáveis indicadas. Isto significa que, os valores que os argumentos tinham no "CALL", são "transmitidos" automaticamente para a rotina. Na rotina estes podem ser por sua vez alterados e, uma vez de volta ao programa de chamada, estes valores estarão permanentemente alterados.

Pode-se fazer como no exemplo, onde se identificou o primeiro argumento, `a` no programa principal, como a "entrada de dados" para a rotina. Ela recebe estes valores com o nome local de `x` e calcula os resultados nas variáveis denominadas localmente de `Y` e `T`. Quando de volta ao programa principal teremos os valores de `B` e `T` dados por `B=A(1)+A(2) e T=A(3)/2`. Observe que a variável `CO` é local da rotina, um real de precisão dupla.

Note que no cabecalho da rotina devemos novamente identificar o tipo das variaveis e sua dimensão por `dimension`, mesmo que aparentemente ela tenha já sido identificada no programa de chamada. Isto permite uma grande versatilidade ao fortran. Por exemplo, poderiamos ter no programa principal `a` variável a definida como `REAL*4 A(4)`, isto é, um vetor de reais de

precisão simples de 4 posições. Na rotina o primeiro argumento poderia ter sido `INTEGER*4 X(2,2)`, isto é, a partir do endereço do início da variável, a rotina organiza os termos em forma de matriz. No caso teriamos o seguinte. No programa principal:

$$\texttt{A} = \left\{ \begin{array}{c} \texttt{a(1)} \\ \texttt{a(2)} \\ \texttt{a(3)} \\ \texttt{a(4)} \end{array} \right\},$$

enquanto que estes valores na subrotina seriam organizados na forma

$$\texttt{X} = \left\{ \begin{array}{cc} \texttt{a(1)} & \texttt{a(3)} \\ \texttt{a(2)} & \texttt{a(4)} \end{array} \right\},$$

isto é, `X(1,2)=A(3)`. De fato, a rotina não altera a forma de organização dos dados na memória. **Qualquer matriz é sempre armazenada em forma de coluna**. Apenas torna-se permissível ao usuário acessar os termos usando dois índices (no exemplo, o que poderiam ser mais), em vez de gerenciar pessoalmente o cálculo dos endereços dos termos na coluna. Outro efeito é que mudamos inclusive o tipo de variáveis. No programa principal era real e na rotina os recebemos como inteiros. A conversão é feita mantendo a contagem dos números de bytes de cada caso. Embora isto em geral seja apenas causa de erro em programas ainda sendo depurados, frequentemente a conversão acima é feita de forma proposital.

O término do programa principal é assinalado pelos comandos `stop` e `end`, enquanto que os subprogramas teminam em `RETURN` e `END` como no exemplo.

2.1 Função e Função Declaração

Além da subrotina, dois outros tipos de subprogramas existem, as chamadas `FUNCTION` e função declaração. A `FUNCTION` é um programa semelhante à subrotina, porém só retorna um único valor. Como uma função de várias variáveis, **a function pode ter diversos argumentos de entrada, mas retorna apenas o valor da função**. Por exemplo, podemos querer definir uma função de a e b por: $F(a,b) = sen(a) + sen(b) + 1$. Neste caso, poderiamos ter um subprograma `FUNCTION` de nome `SOMA` definido como:[2]

```
real function soma(r,s)
somaa = sin(r)
somab = sin(s)
soma = somaa + somab + 1
return
end
```

Note que poderiamos ter programado tudo com uma única linha em vez de três: `SOMA = sin(r)+sin(s)+1.` A função deve ter um nome. Observe que aquelas funções implícitas como seno, cosseno, etc. presentes em todas as linguagens, são de fato programadas internamente como `FUNCTION` na forma acima. Se numa outra rotina precisamos saber o valor de sen (0.1)+ sen $(0.15) + 1$, basta usarmos o comando com a nova função: `SOMA(0.1,0.15)`.

[2]Autor: Prof. Paulo de Tarso R. Mendonça, Ph.D., 2004, Depto.Eng.Mecânica, UFSC, mendonca@grante.ufsc.br.

A **função declaração** (*statement functions*), são funções definidas por uma única linha de comandos. Esta linha deve ser situada logo após o final do cabecalho do programa ou subprograma. Ela não é executada na sequencia de operações, mas apenas quando chamada. Só podem ser chamada de dentro do mesmo programa em que foi definida. Por exemplo, no exemplo 1, na subrotina poderiamos ter logo após o comando dimension: `FUNC1(p,q) = 2.0*P+Q**2,` isto é:

```
Subroutine subrot1(x,y,t)
Implicit real*8 (a-h,o-z)
Dimension x(3)
func1(p,q) = 2.0*p+q**2
    .
```

c. corpo da subrotina subrot1

```
    .
co=2
y=x(1)+x(2)
t=co*func1(x(2),x(3) )
    .
return
end
```

Neste caso teriamos a variável t calculada por: `2*(2*X(2) + X(3)**2)`. Os asteriscos simples e duplos implicam produto e potência.

A sequencia correta de comandos no cabecalho é a seguinte:

`Subroutine` ou `function` ou `program`
`implicit`
`common`
`dimension`
funções declaração
corpo do programa

`PROGRAM` é um comando adicional que pode ser usado para indicar o nome do programa principal. No exemplo 1 poderiamos ter como primeira linha: `program nome`. O comando `COMMON` é uma outra forma de transferir dados entre programas. Em lugar de passar como argumentos nas chamadas às rotinas, define-se um `common`, por exemplo:

```
common / nome1/ a(3),b
```

`NOME1` é o nome de identificação do `common`, dado pelo programador. Todas as rotinas que necessitarem acesso aos dados nos endereços `A` e `B` basta que tenham em seu cabeçalho um COMMON / NOME1/. Os argumentos das variáveis nas rotinas são nomes locais, não necessáriamente os mesmos `A` e `B` da definição. Também os tipos podem mudar. Entretanto, todas as rotinas num mesmo arquivo devem identificar o mesmo número de bytes. Isto é verificado na etapa de compilação apenas. Como não é verificado na etapa de linkagem, rotinas de diferentes arquivos podem ter comprimentos diferentes no mesmo `common`.

Tabela 1: Funções implícitas mais comuns. Alguns comandos não listados podem ser obtidos simplesmente usando os prefixos `d` e `c` para precisão dupla e complexo. Existem diveros outras funções para operações com números complexos.

Nome	argumento		Nome	argumento		Nome	argumento
exp	real*4		sin	real*4		ifix(x)	converte p/int*4
dexp	real*8		dsin	real*8		hfix(x)	idem p/int*2
cexp	complex*8		cos	real*4		nint(x)	arredonda p/int
cdexp	complex*16		dcos	real*8		sign(a,b)	tansfere sinal
			tan	real*4			
base e			dtan	real*8		min(a,.,.)	minimo do conj.
alog(	real*4		dcotan	real*8		max(a,.,.)	max.do conjunto
dlog	real*8		sinh	real*4			
clog	complex*8		cosh	real*4			
cdlog	complex*16		tanh	real*4			
qlog	real*16		dsinh	real*8			
base 10			abs	val.absol.			
alog10	real*4		dabs	idem, real*8			
			dble	converte p/real*8			
asin	real*4		dfloat	idem			
acos	real*4		iabs	va.absol. inteiro			
atan	real*4		mod(i,j)	resto de i/j			
dasin	real*8		amod(x,y)	resto de x/y			
			dmod(x,y)	idem real*8			

2.2 Funções Implícitas

Como toda linguagem de programação, Fortran possui um conjunto extenso de funções implicitas. A lista completa depende do fabricante de cada compilador e deve ser procurada nos manuais. Uma lista mais usual é dada na Tabela 1.

3 Comandos de Controle

Praticamente toda a programação em Fortran pode ser realizada com o uso de apenas três tipos de comandos: `DO`, `IF` e `GOTO`, além das chamadas a rotinas e funções, e comandos de leitura e impressão de dados. Em geral cada linha de comando pode ser numerada por um número de 1 a 9999. Este número deve ser único para cada linha dentro de um dado subprograma, e é colocado entre as colunas 1 a 5 do arquivo. Desta forma laços de cálculo podem ser facilmente identificados, sem necessidade de tentativa de localização visual através de uma indentação das linha na listagem do programa.[3]

[3]Autor: Prof. Paulo de Tarso R. Mendonça, Ph.D., 2004, Depto.Eng.Mecânica, UFSC, mendonca@grante.ufsc.br.

3.1 DO

O comando `DO` define a repetição do cálculo de um conjunto de linhas de comando. A estrutura vista no exemplo:

```
      j = 0
      DO 123 i = 5, 21, 6
          ...
          ! linhas de comando
          ...
          j = j + 1
          a(j) = 2*i
          ...
          ! mais linhas de comando
          ...
123 continue !  final do laco do DO
```

O caso acima só pode ser usada numa formatação fixa. Existe uma forma mais flexível, aplicavel tanto à formatação fixa quanto à livre, que é a seguinte:

```
      j = 0
      DO i = 5, 21, 6
          ...
          ! linhas de comando
          ...
          j = j + 1
          a(j) = 2*i
          ...
          ! mais linhas de comando
          ...
   ENDDO     ! final do laco do DO.
```

O comando é então na forma `DO N  k = m1, m2, m3`, onde N é o número da linha onde termina o laço de cálculo. No caso este número, 123, foi colocado num comando especial, o `continue.` Poderia ter sido colocado na última linha de comando do laço. `K` é o contador, que varia de `m1` até `m2`, de `m3` em `m3`. O contador e os limites são inteiros e sem sinal. No exemplo `A = {10,22,34}`. O argumento `m3` é opcional, tendo como 1 default.

O comando funciona pelo menos uma vez, percorrendo todo o laço. A quantidade de vezes que o laço é executado é $(m2 - m1)/m3) + 1$. Se $m2 < m1$ o laço é executado uma única vez. Isto porque primeiro é feito $i = m1$ e o laço executado, para só então ser feito o teste se $i < m2$.

Ninhos de `DO`. Os DO's podem claramente serem aninhados na forma:

```
      DO na  ka = ma1, ma2, ma3
          DO nb  kb = mb1, mb2, mb3
              DO nc  kc = mc1, mc2, mc3
                  .
nc                continue
```

```
nb         continue
na continue
```

Os números dos *labels* `na, nb` e `nc` podem ser omitidos, e o final de cada laço pode ser indicado simplesmente por ENDDO.

3.2 GOTO

Existem dois tipos de `goto`, ambos para transferir a execução para uma determinada linha de programa. Esta linha é identificada por um número inteiro n, e a linha deve pertencer ao mesmo subprograma. O primeiro tipo é o `goto` incondicional, que tem a forma:

```
goto n
```

A linha referida pode estar acima ou abaixo do comando. O segundo tipo é o chamado `goto` controlado. Sua forma é a seguinte:

```
goto (n1, n2,..,ni,..,nm),i
```

onde `n1, n2` etc. são números de linhas de comandos. `i` é um inteiro que deve ter valor $1 < i < m$. A transferência é feita para a `i`-ésima linha `ni`. Por exemplo, `goto(10,12,30),k`, fara uma transferência para a linha 12 se `k=2`, e para a linha 30 se `k=3`.

3.3 IF

If's são comandos de teste e de controle de execução. Os tipos são os seguintes. **IF aritimético**, que tem a forma

```
if (a) n1,n2,n3
```

`a` é uma expressão, cujo valor numérico define a ação do comando. se `a<0`, a execução do programa é transferida para a linha `n1`, se `a=0` para a linha `n2` e se `a>0` a transferência ocorre para a linha `n3`. É um comando equivalente a três `goto`'s.

O segundo tipo de if é o **if lógico**, que tem a forma:

```
if (expr) comando
```

`expr` é uma expressão lógica, com valor verdadeiro ou falso, e `comando` é um comando executavel qualquer, exceto um `DO`, mas deve ser uma única linha de programação. Por exemplo,

```
      if(a.le.0) go to 25
                 ....
                 ....            ! linhas de comandos
                 ....
  25  c = d + e
      if (a.eq.b) total = 2.0*a/c
      f = g/h
```

As expressõds lógicas são as seguintes `.and., .or., .not., .le., .lt., .ge., .gt., .eq.` Por exemplo, `if (a.lt.b.and.(c.eq.d.or.a.gt.f) ) a = a + b`.

3.4 if then else

O outro tipo de IF não faz uso de um *label* para delimitar as linhas envolvidas. Sua estrutura é a seguinte:[4]

```
IF expressão1 THEN
    bloco de comandos 1
ELSE IF expressão2 THEN
    bloco de comandos 2    !  Executa caso a expressão1 seja falsa.
ELSE
    bloco de comandos 3    !  Executa caso expressão1 e expressão2 sejam
falsas.
ENDIF
```

Uma construção mais usual é:

```
IF expressão1 THEN
    bloco de comandos 1
ENDIF
```

`Expressões 1, 2 e 3` são expressões lógicas, com resultados verdadeiro ou falso. Exceto as duas primeiras linhas e a última, as demais são opcionais. Se `bloco de comandos 1` for uma única linha o end `ENDIF` é desnecessário.

4 Arquivos, Entrada e Saida de Dados

4.1 Criação de Arquivos

Cada operação de entrada ou saida é realizada sobre um **equipamento lógico**, que pode ser um equipamento físico como um disco, uma impressora, o monitor, teclado ou um arquivo interno. Cada um destes equipamentos lógicos deve ser identificado por um número através de um **comando OPEN**. Por exemplo, para criar um arquivo denominado `dados.dat`, e associa-lo ao número 10, usamos o comando

```
OPEN (unit = 5, file = 'dados.dat')
```

Este comando, de fato, da forma acima, apenas reconhece um arquivo de nome já existente no disco, como é de se esperar num arquivo de dados. No caso, por exemplo de um arquivo onde desejamos colocar a saida de resultados calculados, é possível que o arquivo não exista, e deva então ser criado pelo programa, ou é possível que o arquivo já exista, talvez criado numa execução anterior do programa. Neste caso usa-se o comando

```
OPEN (unit = 6, file = 'saida1.sai', STATUS= 'unknown')
```

[4] Autor: Prof. Paulo de Tarso R. Mendonça, Ph.D., 2004, Depto.Eng.Mecânica, UFSC, mendonca@grante.ufsc.br.

Aqui o comando `STATUS` é usado com o valor `'unknown'` de forma que se o arquivo `'saida1.sai'` não existe ele é criado com o número 6, e se existe ele é usado, sobreescrevendo os valores anteriormente existentes, que são perdidos.

Alguns números são previamente definidos, mas infelizmente variam de fabricante para fabricante de compilador. Por exemplo:

Asterisco *	sempre indica o teclado e o video;
0	inicialmente representa o teclado e o monitor
5	inicialmente representa o teclado
6	inicialmente representa o monitor.

Os valores 0, 5 e 6 acima são para o compilador DIGITAL e representa valores de default. Estes números podem ser reutilizados via `open`, mas uma vez o arquivo seja fechado com **close**, o default acima volta a funcionar.

4.2 Entrada e Saida

Os principais comandos de entrada e saida de dados são `read`, `write` e `print`. Por exemplo, para escrever no video e no arquivo de saida `saida.sai` o texto bastante original: `este eh um teste`, usamos

```
write(*,*)  'este eh um teste.'
write(6,*)  'este eh um teste.'
```

Para mostrar no video pode-se usar o comando `print`, que sempre manda os dados para o video: `print 'este e um teste.'`.

A leitura de dados depende do tipo de dados e de arquivo. Para dados armazenados em caracteres ASCII, a leitura é feita linha por linha a cada execução de um comando `read`, de forma sequencial, desde a primeira linha até a última linha do arquivo, sem possibilidade de voltar a ler uma linha já lida. Este é o chamado **arquivo sequencial**, do tipo usado em dados do usuario. Por exemplo, consideremos que o arquivo de dados definido acima, o `dados.dat`, seja composto por duas linhas na forma:

```
2     0.35          1.4E-5
   5     -2.5D5
```

Poderiamos ler estes dados usando:

```
read(5,*) n1, a,b
read(5,*) n2, c
```

Neste caso teriamos que $n1 = 2$, $a = 0.35$, $b = 1.4 \cdot 10^{-5}$, $n2 = 5$ e $c = -2.5 \cdot 10^{5}$. Observe que deveriamos ter previamente definido $n1$ e $n2$ como inteiros, a e b como reais e c como real de precisão dupla.

4.3 Formatação

Se usamos um comando `write` para escrever dois números reais, por exemplo, eles serão escritos de forma padrão, com um espaçamento padrão entre eles, e o primeiro comecará na primeira

coluna da linha. Frequentemente desejamos que os arquivos possuam uma formatação apropriada, principalmente quando são relatórios que devem ter uma clareza de organização. Neste caso os comandos de `write` (e também `read`), podem ser usados em conjunto com o comando `format`. Cada comando `read` and `write` possui associado um comando format, embora um comando format possa ser usado por diversos comandos `read` and `write`.[5]

A forma de uso é a seguinte. Suponha que desejamos obter duas linhas no arquivo de saida, com a seguinte formatação:

```
diametro = 12.5             espessura =  2.5E-03
posicao = -1.35D 03        quantidade = 32
```

Suponhamos que os valores de diametro, etc., estejam armazenados nas variaveis `d`, `esp`, `pos` e `iq`. Para imprimir seriam usados os sequintes comandos

```
    write(6,21) d, esp, pos, iq
21 format('diametro = ',F4.1,'   espessura =  ',1PE8.1,/,'posicao = ',
   *    1PD9.2,' quantidade = ',I2)
```

Note que, previamente, as variaveis `d`, `esp`tinham sido definidas como reais de precisão simples e `pos` e `iq` como real de precisão dupla e inteiro respectivamente. No argumento do format as instruções de formatação devem ser adequadas ao tipo de variavel sendo formatada. Primeiramente, conteudo entre caspas, ' ', são impressos diretamente como caracteres ASCII. Isto permite também incluir **espacamento horizontal**. Outra forma de colocar espaçamento horizontal é com o uso de `nX`, isto é, ' ', com três espaços, é equivalente a `3X`. Outra forma de impressão direta de caracteres é com o comando `.nH,`. Ele transmite os próximos n caracteres para a unidade de saida.

Mudança de linha é feita com ,/, isto é, cada / faz o posicionamento mudar para o inicio da próxima linha. É possível usar por exemplo ,///, para três linhas, ou ,(3/),.

Reais em geral, precisão simples ou dupla, são formatados pelo comando ,`Fm.n`, onde `m` indica a quantidade total de digitos e `n` a quantidade de decimais. Isto deve incluir o sinal. Por exemplo, `-2.247` necessita de um formato `F6.3`. Note que na impressão o formato no format pode **truncar** o valor impresso da variavel. Por exemplo, caso o valor do diâmetro `d` fosse `0.001`. Neste caso, com o formato usado `F4.1`, teriamos impresso 0.

Uma forma mais eficiente de formatação de reais é com os comandos ,`iPEm,n,` e ,`iPDm,n,` para precisão simples e dupla respectivamente. Aqui os números são impressos em potência de 10. `m` é a quantidade total de posições ocupadas, incluindo o próprio `E`, o sinal do número e do expoente, mesmo que positivo. `n` é a quantidade de decimais. i é a quantidade de dígitos antes do ponto decimal.

Um **número inteiro** é formatado com o comando `In`, onde `n` é a quantidade de dígitos, incluindo o sinal.

Uma variável **caracter** é formatada por `,An,` onde `n` é a quantidade de caracteres da variável. Por exemplo, a variavel `tabela` de valor `'ano 2000'`.

```
      character*8
      write(6,22) tabela
```

[5]Autor: Prof. Paulo de Tarso R. Mendonça, Ph.D., 2004, 2013, Depto.Eng.Mecânica, UFSC, mendonca@grante.ufsc.br.

```
22 format(A8)
```

Isto produzirá como saida `ano 2000`.

Repetições. Note que todos os comandos acima podem ser repetidos, por exemplo, `,I3,I3,` pode ser colocado simplesmente como `,2I3,`.É possível agrupar e repetir formatos. Por exemplo: `,F6.3,2X,F6.3,2X,` pode ser colocado como `,2(F6.3,2X),`.

Vetores e matrizes. A leitura de vetores e matrizes é feita como segue. Por exemplo, para ler num vetor `iv(3)` uma linha com `2  5  45`, e em seguida escreve-lo os comandos seriam:

```
read(5,10) (iv(i), i=1,3)
write(6,10)(iv(i),i=1,3)
```

Uma matriz `im(2,3)` pode se escritas por exemplo como:

```
   write(6,11)((im(i,j),j=1,3),i=1,2)
11 format(3(I5,2X))
```

Caso seja usado `write(6,11)im,` sem os `do`'s implícitos, a matriz também é impressa completamente, embora numa formatação pré-definida. Sem dúvida os `do`'s podem ser explícitos, por exemplo:

```
   DO 21 i = 1,2
21 write(6,11)(im(i,j),j=1,3)
```

produz o mesmo efeito.

Listas com **formatação não definida**. É possivel ler uma linha de dados usando um read(n,*), isto é, sem um format associado. Neste caso os números na linha de dados são associados às variáveis do read simplesmente através dos espaçamentos entre os números e os tipos de variaveis definidas. por exemplo, uma linha no arquivo de dados 5 com os valores `2 -1.3` seria lida por: `read(5,*) j,d`, onde préviamente se teria `j` e `d` definidos como variáveis inteira e real respectivamente. Note que é necessário o espaço entre os números.

Para escrever no arquivo, `write(6,*) j,d` produz uma linha num formato padrão do compilador.

4.4 Arquivos de Acesso Direto

Os arquivos lidados na seção anterior eram do tipo sequencial, isto é, uma vez lida ou escrita uma linha, era impossivel acessa-la novamente ou às anteriores. Dispositivos físicos como video e teclado são do mesmo tipo. Existe um outro tipo de arquivos, denominados de acesso direto. Eles permitem oa acesso direto para leitura e escrita de regitros situados em qualquer posição. Um `registro` é um bloco de dados, de mesmo comprimento, identificados por um número, armazenados em sequência na memória, e que podem apenas ser acessados em sua primeira posição.

A definição de um arquivo de acesso direto é feita com o comando `open` já visto, na forma:

```
      OPEN(10,FILE='ROD1A.TMP',ACCESS='DIRECT',RECL=800,
     *        STATUS='unknown')
```

Aqui o nome do arquivo foi posto como `ROD1A.TMP`. O comando `ACCESS='DIRECT'` o definie como de acesso direto, e o comando `RECL=800` indica que cada registro contém 800 bytes. Em geral é necessária a construção de conjunto de rotinas para permitir uma manipulação confortavel dos dados na programação. Detalhes podem ser vistos nos manuais de progrmação.

5 Dimensionamento, Includes, alocação dinâmica

O comando `include` insere o conteudo de um arquivo texto especificado no ponto onde a linha de include foi localizada. Por exemplo, se colocamos num ponto de uma rotina o comando

```
include  'arqnome.001'
```

todo o conteudo do arquivo `arqnome.001` é "puxado" para esta rotina e tratado como se aquele conteudo estivesse nela, naquela posição.

Uma dificuldade para o programador ocorre nas versões tradicionais do Fortran quanto ao dimensionamento de matrizes. Sempre que se usa por exemplo um comando como o `dimension`, o tamanho indicado para a variavel deve ser um valor numérico, por exemplo `dimension a(20)`. Mas suponha que `a` seja uma área onde desejamos armazenar valores a serem lidos, e não sabemos a priori quantos serão estes valores. Uma possibilidade é usar um número n grande o suficiente para cobrir todas as possibilidades, por exemplo `a(1000)`. Quando da compilação e linkedição do programa se terá um arquivo executável que alocará 4000 bytes devido àquela variável. Eventualmente o usuário notará que para diversos conjuntos de dados bastaria valores da ordem de 230 variáveis, e outros 52. Mas o usuário do programa precisaria abrir a listagem do programa, alterar aquele dimensionamento e compilar tudo de novo e assim conseguir uma economia de memória requerida. Uma forma mais eficiente consiste no seguinte. Cria-se um arquivo, por exemplo o `arqnome.001`, com o conteúdo

```
n = 230
```

e no programa se colocaria o seguinte:

```
include  'arqnome.001'
dimension a(n)
```

Note que todo este problema só existe no programa principal.

5.0.1 Alocação dinâmica

Ocorre frequentemente a necessidade de dimensionar uma matriz, porém suas dimensões não são fixas, não são conhecidas a priori: em vez disso, essas dimensões devem ser lidas do arquivo de dados, ou obtidas durante a execução do programa. Nesse caso, a alocação dinâmica de memória é uma solução elegante, presente a partir do Fortran 90. A estrutura do programa é a seguinte:

```
c       Exemplo 2
c      Programa Principal
```

```
Program NomedoPrograma
Implicit real*8 (a-h,o-z)
real*8 B(:), C(:,:), D(:,:,:)              !  vetor B e matrizes C e D.
Dimension a(3)
allocatable B,C,D
    .
```

c. corpo do programa principal

```
    .
a(1)=1.2
a(2)=1.3D-3;          a(3)=2*a(2)
call subrot2(a,n1,n2,n3)       !  calcula ou le as dimensoes n1,n2,n3.
allocate (B(n2),C(n1,n2),D(n1,n2,n3) )          ! dimensiona as matrizes
    .
```

c. mais programa principal

```
    .
stop
end NomedoPrograma                          !  termino do programa principal.
```

O conjunto de comandos:

```
real*8 B(:), C(:,:), D(:,:,:)
allocatable B,C,D
allocate (B(n2),C(n1,n2),D(n1,n2,n3) )
```

pode ser usado em qualuer programa ou subprograma. `allocatable` é usado ao final do cabeçalho, e claramente `allocate` so pode ser usado quando se conhece as dimensões a serem aplicadas. Caso as áreas alocadas (A, B e C) sejam de uso local, após seu uso elas podem ser desalocadas para liberar espaço de memória, usando, por exemplo, `deallocate (B(n2))`. De qualquer forma, quando o processamento do programa termina, todas as áreas são desalocadas.

6 Exercícios

1. Definir uma `FUNCTION` para calcular $f(x) = x^2 + \sqrt{1+2x+3x^2}$. Use a função para calcular[6]

$$a_1 = \frac{6,9+y}{y^2+\sqrt{1+2y+3y^2}} \qquad a_3 = \frac{\operatorname{sen} y}{y^2+\sqrt{1+2x+3x^2}}$$
$$a_2 = \frac{21z+z^4}{z^2+\sqrt{1+2z+3z^2}} \qquad a_4 = \frac{1}{\operatorname{sen}^2 y+\sqrt{1+2\operatorname{sen} y+3\operatorname{sen}^2 y}}$$

calcule os valores e armazene-os num vetor. Faça um relatório com estes valores num arquivo e no video.

2. Escrever uma subrotina para calcular $r = \sqrt{a+b\,x+x^2}$, $s = \cos(2\pi\,x+a)\,e^{bx}$ e $t = ((a+bx)/2)^{l+1} - ((a-bx)/2)^{l-1}$, onde os argumentos da rotina são a, b, x e l.

[6]Autor: Prof. Paulo de Tarso R. Mendonça, Ph.D., 2004, 2013, Depto.Eng.Mecânica, UFSC, mendonca@grante.ufsc.br.

3. Dado um vetor $\mathbf{x}$ de 20 componentes, a) escrever uma rotina para calcular o vetor y cujas componentes estão em ordem reversa em relação às componentes de $\mathbf{x}$. (b) Escreva uma rotina para colocar em ordem crescente as componetes de $\mathbf{x}$, porém usando a mesma área de memória x.

4. Dada uma matriz quadrada simétrica $\mathbf{A}$(50,50), escrever uma rotina pra trocar a diagonal principal pela diagonal secundária a partir do termo (2,2) até o termo (49,49). Ajuste a rotina para admitir uma matriz de ordem arbitrária (n,n), onde n é um inteiro lido de um arquivo de dados, junto aos termos da própria matriz.

5. Escrever uma rotina para calcular o produto escalar entre dois vetores dados, $\mathbf{x}(n)$, $\mathbf{y}(n)$, com n também variável, dado como argumento.

6. Escreva uma rotina para calcular todas as raizes de um polinômio real de segundo grau. Os coeficientes devem ser lidos de um arquivo texto, e a resposta mostrada em outro arquivo.

7. Considere uma **matriz triangular superior**, isto é, uma matriz simétrica cujos termos armazenados são apenas aqueles acima da diagonal principal. Esses termos são armazenados em um vetor, coluna após coluna, da seguinte forma:

$$\mathbf{A} = \begin{bmatrix} A_{11} & A_{12} & A_{13} & \cdots \\ & A_{22} & A_{23} & \cdots \\ & & A_{33} & \cdots \\ & & & \ddots \end{bmatrix}_{N\times N} \quad \rightarrow \text{armazenados como: } \mathbf{V} = \begin{bmatrix} A_{11} \\ A_{12} \\ A_{22} \\ A_{13} \\ A_{23} \\ A_{33} \\ \vdots \\ A_{NN} \end{bmatrix}_{M} .$$

   O número de termos no vetor é obtido pela fórmula (facilmente deduzivel): $M = N(N+1)/2$. Um termo arbitrário A_{ij} de $\mathbf{A}$ pode ser localizado na posição m de $\mathbf{V}$ por: $m = (j-1)j/2 + i$. Por exemplo, o termo A_{23} está na posição $m = 5$ de $\mathbf{V}$.

   (a) Programe uma subrotina que receba uma matriz de ordem $N \leq 8$ armazenada na forma triangular superior num vetor $\mathbf{V}$, e imprima os termos na disposição de matriz superior.

   (b) Programe uma subrotina que receba uma matriz $\mathbf{A}$ de ordem N qualquer, armazenada na forma triangular superior num vetor $\mathbf{V}$, e um vetor $\mathbf{X}$ também de ordem N. Faça o produto $\mathbf{AX} = \mathbf{F}$, e imprima o resultado $\mathbf{F}$. Dica: use função declaração mm(i,j) = (j-1)*j/2+i. Em seguida, o produto é feito por:

```
      DO 10 i = 1,N                    ! corre termos de F
      F(i)=0.0d0
      DO 8 j=1,N                     ! corre coluns de A
8       F(i) = F(i) + V(mm(i,j)) * X(j)
10    Continue
```

(Esse fragmento tem um erro. Identifique-o. Lembre que a fórmula de recorrência da função declaração serve apenas para indicar os termos acima da diagonal da matriz.)

8. Programe um conjunto de rotinas para leitura de dados de elementos finitos a partir de um arquivo texto.

   – Uma rotina para leitura de dados de coordenadas nodais. Cada linha contém: "no coorX coorY coorZ ".

   – Uma rotina para leitura de dados de conectividade de elementos de até quatro nós. Cada linha contém: " elem noI noJ noK noL ".

   Cada rotina de leitura deve também enviar os valores lidos para compor um relatório num arquivo texto de saida. Os valores lidos devem permanecer a disposição no programa principal em matrizes de dimensões "coor(NNOS,3) " e "iconec(NELEM,5) ", onde NNOS e NELEM são os primeiros valores lidos no arquivo de dados, que serão em seguida utilizados para fazer a alocação dinâmica das matrizes de dados.