

Introdução ao Fortran 90 - 4

Alexandre Diehl

Departamento de Física – UFPel

Arquivos de dados externos: acesso E/S

O **Fortran 90** permite que a **entrada (E)** e a **saída (S)** de dados seja feita a partir de um (ou vários) **arquivo externo** ao programa.

Para o **acesso E/S** em um arquivo externo, é necessário que o programador:

- Identifique o **nome do arquivo** a ser usado como dispositivo de **E/S**
- Informe o **tipo de acesso e uso** que será feito do arquivo
- Associe **instruções individuais** de **E/S** com o(s) arquivo(s) em uso
- Quando as **operações** de **E/S** estiverem **concluídas**, é necessário **instruir ao sistema** que **não é mais necessário acessar o arquivo**.

O **acesso E/S** será feito através dos comandos

READ ([UNIT=]<unidade>, <lista>) e **WRITE** ([UNIT=]<unidade>, <lista>)

Abertura de arquivos: comando **OPEN**

Permite que se **associe (conecte)** um **arquivo externo** a uma **unidade**, para realizar um acesso **E/S**

```
OPEN ([UNIT=] <unidade> [, <op-list> ])
```

<unidade>: **número inteiro (entre 1 e 100)** que identifica o arquivo externo de E/S a ser aberto

- evite o uso do **número 5**, por se tratar da unidade de **entrada padrão (teclado)**
- evite o uso do **número 6**, por se tratar da unidade de **saída padrão (monitor)**

<op-list>: lista opcional de especificadores que caracterizam o arquivo externo a ser aberto

- FILE = <fln>
- STATUS = <status>
- ACTION = <act>
- ACCESS = <acc>
- POSITION = <pos>
- Outros ...

Abertura de arquivos: comando **OPEN**

Usando **unidades** lógicas **pré-conectadas**

```
OPEN ([UNIT=] <unidade>)
```

Unidades pré-conectadas: 5 (entrada via **teclado**) e 6 (saída no **monitor**)

```
real :: b  
open ( unit=5)  
read (5,*)b      ! leitura do teclado  
open ( unit=6)  
write (6,*) ' _dado_=_ ',b ! escrita no monitor  
END
```

Os comando acima podem ser substituídos por

READ*, ou **READ (*,*)** e **PRINT***, ou **WRITE (*,*)**

Abertura de arquivos: comando **OPEN**

Usando **unidades** lógicas **externas** via **arquivos de dados**

```
OPEN ([UNIT=] <unidade>, FILE= <fln>)
```

<fln>: expressão de caracteres que fornece o **nome do arquivo**

```
real :: b  
open (unit=10, file='input.dat')  
read (10,*)b      ! leitura do arquivo  
open (unit=20, file='dados.dat')  
write (20,*)'_ldado_=',b ! escrita no arquivo  
END
```

O **arquivo de entrada** **input.dat** deve conter a mesma estrutura dos **READ** do programa

```
10.0      ! dado de entrada
```

Abertura de arquivos: comando **OPEN**

Usando **unidades** lógicas **externas** via **arquivos de dados**

```
OPEN ([UNIT=] <unidade>, FILE= <fln>)
```

<fln>: expressão de caracteres que fornece o **nome do arquivo**

```
real :: b  
open (unit=10, file='input.dat')  
read (10,*)b      ! leitura do arquivo  
open (unit=20, file='dados.dat')  
write (20,*)'_ldado_=_',b ! escrita no arquivo  
END
```

O **arquivo de saída dados.dat** será criado no diretório onde o programa é executado

```
dado =      10.0000000
```

Abertura de arquivos: comando **OPEN**

Usando **unidades** lógicas **externas** via **arquivos de dados**

```
OPEN ([UNIT=] <unidade>, FILE= <fnl>)
```

<fnl>: expressão de caracteres que fornece o **nome do arquivo**, caso o arquivo esteja num **diretório distinto do de trabalho**

```
real :: b
open (unit=10, file='../input.dat') ! um diretorio abaixo
read (10,*)b ! leitura do arquivo
open (unit=20, file='/home/usuario/teste/dados.dat')
write (20,*)'_dado_=',b ! escrita no arquivo
END
```

O **arquivo de saída dados.dat** será criado no diretório **teste** indicado no caminho

```
dado = 10.0000000
```

Abertura de arquivos: comando **OPEN**

Usando **unidades** lógicas **externas** via **arquivos de dados**

```
OPEN ([UNIT=] <unidade>, FILE= <fln>, STATUS= <status>)
```

<status>: expressão de caracteres que fornece o **status** do **arquivo**

```
real :: b
open (unit=10, file='input.dat', status='OLD')
read (10,*)b      ! leitura do arquivo
open (unit=20, file='dados.dat', status='NEW')
write (20,*)'_dado_=_',b ! escrita no arquivo
END
```

- status **OLD**: arquivo **já existe**; se **não existir**, ocorrerá um **erro de execução**
- status **NEW**: arquivo **será criado** pelo programa; se **já existir**, ocorrerá um **erro de execução**

Abertura de arquivos: comando **OPEN**

Usando **unidades** lógicas **externas** via **arquivos de dados**

```
OPEN ([UNIT=] <unidade>, FILE= <fln>, STATUS= <status>)
```

<**status**>: expressão de caracteres que fornece o **status** do **arquivo**

```
real :: b  
open (unit=10, file='input.dat', status='OLD')  
read (10,*)b      ! leitura do arquivo  
open (unit=20, file='dados.dat', status='REPLACE')  
write (20,*)' _dado_=_',b  ! escrita no arquivo  
END
```

- status **REPLACE**: se o arquivo **não existe**, ele **será criado**; se **já existir**, este **será eliminado** e um **novo arquivo é criado** com o **mesmo nome**
- status **SCRATCH**: o arquivo **é temporário** e será **deletado** quando este for **fechado** com o comando **CLOSE** ou na **saída** da unidade de programa

Abertura de arquivos: comando **OPEN**

Usando **unidades** lógicas **externas** via **arquivos de dados**

```
OPEN ([UNIT=] <unidade>, FILE= <fln>, ACTION= <act>)
```

<act>: expressão de caracteres que indica como o arquivo será usado

```
real :: b
open (unit=10, file='input.dat', status='OLD', ACTION='READ')
read (10,*)b      ! leitura do arquivo
open (unit=20, file='dados.dat', ACTION='WRITE')
write (20,*)' _dado_=_',b ! escrita no arquivo
END
```

- action 'READ': os comandos **WRITE**, **PRINT** e **ENDFILE** não devem ser usados; se usados, uma mensagem de erro de compilação será apresentada
- action 'WRITE': o comando **READ** não pode ser usado; se usado, uma mensagem de erro de compilação será apresentada
- action 'READWRITE': não há restrição de uso

Fechamento de arquivos: comando **CLOSE**

Usando **unidades** lógicas **externas** via **arquivos de dados**

CLOSE ([UNIT=] <unidade>)

Permite que se desassocie um arquivo externo a uma unidade

```
real :: b
open (unit=10, file='input.dat', status='OLD')
read (10,*)b      ! leitura do arquivo
open (unit=20, file='dados.dat', status='NEW')
write (20,*)'LdadoL=L',b ! escrita no arquivo
close (unit=10)
close (20)
END
```

Uma vez fechado o arquivo, a unidade que estava sendo usada pode ser associada na abertura de outro arquivo dentro do programa

Fim de arquivo de entrada: opção `IOSTAT=<ios>`

Usado quando **não sabemos o número de dados** a serem lidos do arquivo de entrada

```
READ ([UNIT=]<unidade>, [FMT=]<formato>, IOSTAT=<ios>)
```

`<ios>`: variável inteira que armazena o status do processo de leitura

- `<ios> = 0`: quando o comando é **executado sem erros**
- `<ios> > 0`: quando ocorre um **erro na execução** do comando
- `<ios> < 0`: quando uma condição de **final de arquivo** é detectada

Fim de arquivo de entrada: opção `IOSTAT=<ios>`

Tarefa: Considere o arquivo `input.dat`, com os dados listados em duas colunas:

```
0.0 0.0
1.0 2.0
2.0 4.0
3.0 8.0
```

Queremos fazer um programa que execute os seguintes passos:

- Ler o arquivo para saber quantas linhas deverão ser lidas
- Usar esta identificação para **alocar de forma dinâmica** os **vetores x e y** que armazenarão os dados
- Ler os dados do arquivo
- Fazer a saída dos dados lidos através do monitor

Fim de arquivo de entrada: opção IOSTAT=<ios>

```
integer :: i, ndata, stat
real, dimension (:), allocatable :: x, y
open (unit=10, file='input.dat', status='old', action='read')
ndata = 0
do
  read(10,*, iostat=stat)
  if (stat < 0) exit
  ndata = ndata + 1
end do
close (unit=10)
print*, '_numero_de_dados_=' , ndata
allocate(x(ndata), y(ndata))
open (unit=10, file='input.dat', status='old', action='read')
do i = 1, ndata
  read (10,*)x(i),y(i)
  print*,x(i),y(i)
end do
close (10)
END
```

Operação com matrizes

Tarefa: Considere a **Matriz B**, com **forma (5,4)**, armazenada num arquivo

Arquivo **matrizB.dat**

$$B = \begin{pmatrix} 1 & 4 & 0 & 10 \\ -2 & 4 & 3 & 0 \\ 5 & 3 & 0 & 11 \\ 7 & 1 & 2 & 9 \\ 11 & 1 & 4 & 4 \end{pmatrix}$$

```
1 4 0 10
-2 4 3 0
5 3 0 11
7 1 2 9
11 1 4 4
```

Queremos fazer um programa que execute os seguintes passos:

- Ler o arquivo **matrizB.dat**, linha por linha, armazenando os dados nos elementos **B(i,j)** da matriz
- Fazer a saída dos dados lidos através do monitor, no formato da matriz **B** original
- Fazer a saída dos dados lidos num arquivo, numa forma sequencial

Operação com matrizes

```
integer , parameter :: linhas = 5, colunas = 4
integer :: i, j, stat
integer , dimension(linhas , colunas) :: B
open (unit=10, file='matrizB.dat', status='old', action='read')
do i = 1,linhas
    read (10,*, iostat=stat)(B(i,j), j = 1,colunas)
end do
do i = 1,linhas
    print '(4(I2,1X))', (B(i,j), j = 1,colunas)
end do
close (unit=10)
END
```

A saída no monitor será como mostrado abaixo

```
1  4  0 10
-2 4  3  0
 5  3  0 11
 7  1  2  9
11  1  4  4
```


Operação com matrizes

```
integer , parameter :: linhas = 5, colunas = 4
integer :: i, j
integer , dimension(linhas , colunas) :: B
open (unit=10, file='matrizB.dat', status='old', action='read')
do i = 1,linhas
    read (10,*)(B(i,j), j = 1,colunas)
end do
close (unit=10)
open(unit=20,file='dados.dat', status='new', action='write')
write (20, '(A)') '#_Matriz_B_lida:'
write (20,*)
write(20, '(20(I3,1x))')B
close(20)
END
```

A saída no arquivo dados.dat será como mostrado abaixo

```
# Matriz B lida:
  1  -2   5   7  11   4   4   3   1   1   0   3   0   2   4  10
0  11   9   4
```

Operação com matrizes

A **entrada formatada** exige que os dados sejam apresentados exatamente como nos formatos descritos no comando **READ**

```
(...)  
read (10, '(3(F5.1,1x))')(a(i), i = 1, 3)  
read (10, '(3(F5.1,1x))')(b(i), i = 1, 3)  
(...)
```

Conteúdo do arquivo associado à unidade 10 →

```
-10.0  2.5  0.5  
4.5    -10.2  1.5
```

Conteúdo do arquivo associado à unidade 10:

```
-10.0  2.5  0.5  
4.5    -10.2  1.5
```

Operação com matrizes

Tarefa: Usando dois vetores \vec{a} e \vec{b} ,

$$\vec{a} = -10.0\vec{i} + 2.5\vec{j} + 0.5\vec{k} \quad \text{e} \quad \vec{b} = 4.5\vec{i} - 10.2\vec{j} + 1.5\vec{k},$$

lidos a partir de um arquivo, faça um programa que calcule e imprima o produto vetorial $\vec{c} = \vec{a} \times \vec{b}$ destes vetores. Para este cálculo, use a convenção para o produto vetorial,

$$c_i = \sum_{j=1}^3 \sum_{k=1}^3 \varepsilon_{ijk} a_j b_k,$$

onde c_i é a componente i do vetor \vec{c} , e ε_{ijk} é o tensor de Levi-Civita, que assume os seguintes valores, dependendo das possíveis combinações para os índices (i, j, k) ,

$$\varepsilon_{ijk} = \begin{cases} 1 & \text{se } (1, 2, 3) \text{ ou } (2, 3, 1) \text{ ou } (3, 1, 2) \\ -1 & \text{se } (3, 2, 1) \text{ ou } (1, 3, 2) \text{ ou } (2, 1, 3) \\ 0 & \text{se } i = j \text{ ou } i = k \text{ ou } j = k \end{cases}$$

Operação com matrizes

```
integer , parameter :: dim = 3
real :: a(dim), b(dim), c(dim), eijk
integer :: i, j, k
open(unit=10, file='vetores.dat')
read (10, '(3(F5.1,1x))')(a(i), i = 1, dim)
read (10, '(3(F5.1,1x))')(b(i), i = 1, dim)
write (10, '( "vetor_a:" )')
write (10, '(5x,3(F5.1,1x))')(a(i), i = 1, dim)
write (10, '( "vetor_b:" )')
write (10, '(5x,3(F5.1,1x))')(b(i), i = 1, dim)
! (... bloco para produto vetorial ...)
write (10, '( "produto_vetorial_c = a x b:" )')
write (10, '(5x,3(F5.2,1x))')(c(i), i = 1, dim)
close (10)
END
```

Operação com matrizes

```
! (... bloco para produto vetorial ...)  
c = 0.0  
do i = 1, dim  
  do j = 1, dim  
    do k = 1, dim  
      if ((i == 1 .and. j == 2 .and. k == 3) .or. &  
          (i == 2 .and. j == 3 .and. k == 1) .or. &  
          (i == 3 .and. j == 1 .and. k == 2)) then  
        eijk = 1.0  
      else if ((i == 3 .and. j == 2 .and. k == 1) .or. &  
              (i == 1 .and. j == 3 .and. k == 2) .or. &  
              (i == 2 .and. j == 1 .and. k == 3)) then  
        eijk = -1.0  
      else  
        eijk = 0.0  
      end if  
      c(i) = c(i) + eijk*a(j)*b(k)  
    end do  
  end do  
end do  
(...)
```

Operação com matrizes

Arquivo `vetores.dat` após a execução do código

```
-10.0 2.5 0.5  
4.5 -10.2 1.5  
vetor a:  
    -10.0 2.5 0.5  
vetor b:  
    4.5 -10.2 1.5  
produto vetorial c = a x b:  
    8.85 17.25 90.75
```