Programação estruturada no Fortran 90 - 4

Alexandre Diehl

Departamento de Física – UFPel

Unidade de programa do tipo MÓDULO:

- Usado para agrupar dados globais.
- Usado para agrupar tipos derivados e suas operações associadas.
- Usado para agrupar blocos de interface.
- Usado para agrupar rotinas internas.

→ Não pode ser usado separadamente, pois não tem uma parte executável.

→ Quando declarado no mesmo arquivo do Programa Principal, devem ser declarados antes do início do Programa Principal.

→ Podem ser declarados dentro de um arquivo .f90 em separado do Programa Principal (ou Sub-Programas) que usam os Módulos.

Formas de definição de um MÓDULO:

```
MODULE <nome módulo>
        <declaração de variáveis>
[CONTAINS
        <rotinas de módulo>]
END MODULE <nome módulo>
```

Formas de uso de um MÓDULO:

- Dentro do Programa Principal (ou Sub-Programas) devemos usar a instrução

USE <nome módulo>

→ A instrução USE não é executável e deve ser colocada logo após o cabeçalho da unidade de programa (PROGRAM, FUNCTION, SUBROUTINE ou MODULE), antes de qualquer declaração de variáveis ou comandos executáveis.

```
Module Globais
  implicit none
  integer :: N, iseed
  integer :: tentativa_maxima
  real :: L, sigma
  real, allocatable :: x(:), y(:)
end Module Globais

program insere_particulas
  USE Globais
  integer :: i
  read*,N, sigma
  read*,L
  read*,tentativa_maxima
  read*,iseed
  allocate(x(N),y(N))
  call inicializar()
  do i = 1,N
     print*,x(i),y(i)
  end do
CONTAINS
 [...]
```

Declaração de objetos Globais:

- Todas as variáveis declaradas no Módulo são visíveis no Programa Principal, em função da inclusão da declaração USE.
- Todas as variáveis declaradas no Módulo são visíveis dentro dos Sub-Programas, uma vez que foram definidas a partir da instrução CONTAINS. Neste caso, não é necessário a inclusão da declaração USE.
- A inclusão do implicit none dentro do Módulo vale para todas as unidades que fazem uso do Módulo.

```
SUBROUTINE inicializar()
    integer :: i = 1, j, tentativa = 1
    real :: distancia
    logical :: overlap
    do while (i <= N)
        x(i) = ran2(iseed)*L;    y(i) = ran2(iseed)*L
        j = 1
        overlap = .false.
        do while (j < i .and. (.not.overlap))
            overlap = look_overlap(x(i),x(j),y(i),y(j))
            j = j + 1
        enddo
        if (overlap) then
            tentativa = tentativa + 1
            if (tentativa > tentativa_maxima) then
                write (*,'(A)') 'tentativas excedidas. Pare!'
                stop
            endif
        else
            tentativa = 1;      i = i + 1
        endif
    enddo
 end SUBROUTINE inicializar
```

```
logical function look_overlap(x1,x2,y1,y2)
  implicit none
  real :: x1, x2, y1, y2
  real :: distancia

  look_overlap = .false.
  distancia = sqrt((x1-x2)**2+(y1-y2)**2)

  if (distancia < sigma) then
     look_overlap = .true.
  endif

end function look_overlap
```

- A Função ran2 deve ser inserida dentro do bloco CONTAINS, pois é usada dentro da Subrotina de inicialização das coordenadas das partículas.

```
Module Globais
  implicit none
  integer :: N, iseed
  integer :: tentativa_maxima
  real :: L, sigma
  real, allocatable :: x(:), y(:)
end Module Globais

program insere_particulas
  USE Globais
  integer :: i
  read*,N, sigma
  read*,L
  read*,tentativa_maxima
  read*,iseed
  allocate(x(N),y(N))
  call inicializar()
  do i = 1,N
     print*,x(i),y(i)
  end do
end program insere_particulas
```

Declaração de objetos Globais:

- Usando Sub-Programas internos (**sem** CONTAINS) as variáveis declaradas no Módulo são visíveis com a inclusão da declaração USE.

  → num único arquivo .f90 a ordem das unidades de programa deve ser: Módulos, Programa Principal, Sub-Programas.

- Usando Sub-Programas Externos (declarados em arquivos .f90 separados) as variáveis declaradas no Módulo são visíveis com a inclusão da declaração USE.

  → Os arquivos .f90 com as unidades de programa devem ser reunidos na **etapa de compilação**.

```
SUBROUTINE inicializar()
   USE Globais
   integer :: i = 1, j, tentativa = 1
   real :: distancia, ran2
   logical :: overlap, look_overlap
   do while (i <= N)
      x(i) = ran2(iseed)*L;    y(i) = ran2(iseed)*L
      j = 1;          overlap = .false.
      do while (j < i .and. (.not.overlap))
         overlap = look_overlap(x(i),x(j),y(i),y(j))
         j = j + 1
      enddo
      if (overlap) then
         tentativa = tentativa + 1
         if (tentativa > tentativa_maxima) then
            write (*,'(A)') 'tentativas excedidas. Pare!'
            stop
         endif
      else
         tentativa = 1;       i = i + 1
      endif
   enddo
 end SUBROUTINE inicializar
```

```
logical function look_overlap(x1,x2,y1,y2)
  USE Globais, ONLY : sigma
  implicit none
  real :: x1, x2, y1, y2
  real :: distancia
  look_overlap = .false.
  distancia = sqrt((x1-x2)**2+(y1-y2)**2)
  if (distancia < sigma) then
     look_overlap = .true.
  endif
end function look_overlap
```

- A declaração USE com o qualificador ONLY limita o acesso somente a certos objetos dentro de um Módulo.

  USE <nome módulo>, ONLY : <lista only>

  → Os demais objetos definidos no Módulo não são visíveis na unidade de programa, inclusive o implicit none.

MÓDULOS

Usando Sub-Programas Externos, cada um definido num arquivo .f90 em separado, a compilação é feita como:

```
ca/f90$
ca/f90$ gfortran inicializa.f90 look_overlap.f90 ran2.f90 insere_particulas.f90
```

- O arquivo insere_particulas.f90 contém o Programa Principal e o Módulo Globais.

  → Um arquivo com a extensão .mod será criado, onde as informações contidas no Módulo são armazenadas. É nele que as unidades de programa que usam o Módulo irão buscar as informações.

  → O executável a.out será criado, na ausência de erros de compilação.

- Para criar um executável com outro nome (além do padrão a.out), devemos usar a opção **-o <nome do executável>**:

  gfortran <arq1>.f90 ... <arqn.f90> -o <nome do executável>

  → A execução é feita como no padrão:

  ./<nome do executável>

Usando Sub-Programas Externos, cada um definido num arquivo .f90 em separado, a compilação pode ser feita como:

- Primeiro criamos os arquivos .o (objetos) dos respectivos arquivos .f90 associados com os Sub-Programas (e, eventualmente, com os Módulos definidos num arquivo .f90 em separado), usando a opção de compilação **-c <sub>.f90**:

  gfortran -c <prog>.90

  → Um Módulo definido dentro de um arquivo .f90 em separado, quando compilado com a opção -c, irá produzir dois arquivos: um .mod (gerado antes) e outro objeto .o.

- Uma vez criados os arquivos objeto .o, reunimos-os na etapa de compilação com o Programa Principal:

```
a/f90$
a/f90$ gfortran -c globais.f90
a/f90$ gfortran -c inicializa.f90
a/f90$ gfortran -c ran2.f90
a/f90$ gfortran -c look_overlap.f90
a/f90$ gfortran globais.o ran2.o inicializa.o look_overlap.o insere_particulas.f90
```

Rotinas de Módulos:

- Usadas quando Sub-Programas (Funções e Subrotinas) são definidas dentro de Módulos.
- Podem ser chamadas usando o comando CALL ou fazendo referência ao nome da Função, desde que a unidade de programa que fazem uso do Módulo tenha a instrução USE.
- Uma rotina de Módulo pode invocar outras rotinas de Módulo contidas no mesmo Módulo:

  → as variáveis declaradas no Módulo antes da palavra-chave CONTAINS são acessíveis a todas as rotinas do Módulo.

  → variáveis declaradas localmente em um determinado Sub-Programa de módulo são inacessíveis aos outros Sub-Programas.

Rotinas de Módulos:

```
MODULE INTEG
  IMPLICIT NONE
  REAL :: ALFA ! Variavel global.

CONTAINS

  REAL FUNCTION FF(X)
    REAL, INTENT(IN) :: X
    FF= EXP(-ALFA*X*X)
    FF= FF*X2(X)
    RETURN
  END FUNCTION FF

  REAL FUNCTION X2(Y)
    REAL, INTENT(IN) :: Y
    X2= Y**2
    RETURN
  END FUNCTION X2

END MODULE INTEG
```

- A Função FF usa a Função X2.
- A variável X definida na Função FF é local nesta.
- A variável Y definida na Função X2 é local nesta.
- A variável ALFA é global, podendo ser usada nas duas Funções.
- Para o uso deste Módulo, um Programa Principal deve ser criado, que chame o Módulo usando a instrução USE INTEG.

Rotinas de Módulos:

```
module mytrig
  implicit none
  real, parameter :: pi = 3.1415926
  real, parameter :: degree180 = 180.0
  real, parameter :: r_to_d = degree180/pi
  real, parameter :: d_to_r = pi/degree180

CONTAINS

  real function DegreeToRadian(degree)
    implicit none
    real, intent(in) :: degree
    DegreeToRadian = degree * d_to_r
  end function DegreeToRadian

  real function mySin(x)
    implicit none
    real, intent(in) :: x
    mySin = sin(DegreetoRadian(x))
  end function mySin

end module mytrig
```

Rotinas de Módulos:

Programa Principal

```
program degree_trig
  USE mytrig
  real :: begin = -180
  real :: final = 180.0
  real :: step = 30.0
  x = begin
  write(*,'(7X,A,15X,A)')'x','sin(x)'
  do
     if (x > final) exit
     write(*,*)x,mySin(x)
     x = x + step
  end do
end program degree_trig
```

Execução típica

```
diehl@lua:~/Documents/ensino/UFPEL/prog_c
      #x                 sin(x)
 -180.000000       8.74227766E-08
 -150.000000      -0.500000060
 -120.000000      -0.866025388
 -90.0000000       -1.00000000
 -60.0000000      -0.866025448
 -30.0000000      -0.500000000
  0.00000000        0.00000000
  30.0000000       0.500000000
  60.0000000       0.866025448
  90.0000000        1.00000000
  120.000000       0.866025388
  150.000000       0.500000060
  180.000000      -8.74227766E-08
diehl@lua:~/Documents/ensino/UFPEL/prog_c
```

TAREFA 6:

Considere um sistema com N partículas de diâmetro $\sigma = 1.0$. Construa um programa em FORTRAN 90 para inserir as N partículas numa caixa CÚBICA de lado $L = 10.0$ de forma aleatória.

As seguintes condições devem ser verificadas no programa: a) durante a inserção das partículas, a separação entre quaisquer duas partículas não pode ser menor do que σ. b) Caso não seja possível inserir as N partículas, o programa deve ser interrompido e uma mensagem de erro deve ser informada ao usuário. Caso seja possível inserir as N partículas, o programa deve: c) escrever as coordenadas x, y e z num arquivo de saída, usando saída formatada com descritor de edição ES, d) atribuir velocidades com mesmo módulo e direções aleatórias a cada uma das partículas, e) calcular a energia cinética total do sistema, sabendo que todas as partículas têm a mesma massa m, f) calcular a velocidade do centro de massa do sistema. Use Módulos e Sub-Programas no código.

Data limite de entrega: até as 23:55 do dia 23/02/2018

Forma de envio: através do ambiente AVA da disciplina, enviar: arquivos .f90 usados, arquivos de entrada (com os parâmetros usados no programa) e um arquivo em PDF com a transcrição do código.