

Programação estruturada no Fortran 90 - 1

Alexandre Diehl

Departamento de Física – UFPel

Programação estruturada: unidades de programa

A programação estruturada em **Fortran 90** implica no uso de **unidades de programa**, do tipo:

- **Programa principal**
- **Subprogramas:** **Funções** e **Subrotinas**
- Módulos

Cada **unidade de programa** contém um **conjunto completo** e consistente de **tarefas** que podem ser **escritas e compiladas individualmente**.

Apenas o **Programa Principal** pode ser executado individualmente.

Subprogramas: Funções e Subrotinas

Em **Fortran 90** existem dois tipos de subprogramas: **Funções** e **Subrotinas**

- Uma **Função**, quando chamada, **retorna um único valor** calculado dentro da unidade de programa, **associado com o nome** da **Função**
- Uma **Subrotina**, quando chamada, pode **retornar diversos valores** calculados dentro da unidade de programa, **associados com os argumentos** usados na chamada da **Subrotina**

Subprograma do tipo FUNÇÃO

Um subprograma do tipo **FUNÇÃO** tem a seguinte sintaxe

```
[tipo] FUNCTION nome-da-função (arg1, arg2, ..., argn)
    IMPLICIT NONE
    [campo de declaração de variáveis]
    [campo de execução]
END FUNCTION nome-da-função
```

- A opção **tipo** identifica o tipo de dado associado com a **Função**. Pode ser **INTEGER**, **REAL**, **LOGICAL**, etc
- A **Função** é nomeada pelo identificador **nome-da-função**. É ele que devemos usar nas operações envolvendo a **Função**.
- Os **argumentos** da **Função** são identificados por uma lista de **identificadores mudos** **arg1**, **arg2**, ..., **argn**. Eles são usados na unidade de programa para produzir o resultado a ser extraído da **Função**.

Subprograma do tipo FUNÇÃO

Um subprograma do tipo **FUNÇÃO** tem a seguinte sintaxe

```
[tipo] FUNCTION nome-da-função (arg1, arg2, ..., argn)
    IMPLICIT NONE
    [campo de declaração de variáveis]
    [campo de execução]
END FUNCTION nome-da-função
```

- Uma **Função** é uma unidade de programa que recebe dados através de uma **lista de argumentos mudos**, realiza operações com estes argumentos e **retorna um resultado associado com o nome** da **Função**.
- No campo de execução da **Função** devemos ter uma ou mais declarações do tipo

nome-da-função = expressão

onde o resultado da **expressão** será salvo no identificador do nome da **Função**.

Subprograma do tipo FUNÇÃO

Um subprograma do tipo **FUNÇÃO** tem a seguinte sintaxe

```
[tipo] FUNCTION nome-da-função ()  
    IMPLICIT NONE  
    [campo de declaração de variáveis]  
    [campo de execução]  
END FUNCTION nome-da-função
```

- Uma **Função** não precisa ter uma **lista de argumentos mudos**.
- Neste caso, a Função deve ser definida como

```
[tipo] FUNCTION nome-da-função ()
```

e chamada de uma unidade de programa como

```
write(*,*)nome-da-função ()
```

Subprograma do tipo FUNÇÃO

Um subprograma do tipo **FUNÇÃO** tem a seguinte sintaxe

```
[tipo] FUNCTION nome-da-função (arg1, arg2, ..., argn)
    IMPLICIT NONE
    [campo de declaração de variáveis]
    [campo de execução]
    RETURN
END FUNCTION nome-da-função
```

- Se um dos comando dentro da **Função** é uma instrução do tipo **RETURN**, a execução da **Função** é encerrada e o programa segue a partir do ponto em que a **Função** foi chamada.
- Podem existir mais de uma instrução **RETURN** dentro da **Função**, isoladamente ou em combinação com instruções do tipo **IF**

IF (x < 0) **RETURN**

Subprograma do tipo FUNÇÃO

As **Funções** podem ser **internas** ou **externas**

- **Funções internas** (com **CONTAINS**):

→ São definidas dentro de uma **unidade de programa**, a partir da instrução **CONTAINS**.

```
program teste
  implicit none
  [campo de declaracao de identificadores]
  [campo de execucao]
  CONTAINS
    [function 1]
    [function 2]
end program teste
```

- Todas as **Funções** são **listadas a partir** do **CONTAINS**.
- O **Programa Principal** é **finalizado após** a declaração de todas as **Funções**.

Subprograma do tipo FUNÇÃO

Função interna com CONTAINS

```
program testa_function
  implicit none
  write (*,*)GetNumber()

CONTAINS
  real function GetNumber()
    implicit none
    real :: Input_Value
    do
      write (*,*) '_digite um numero positivo: '
      read (*,*)Input_Value
      if (Input_Value > 0.0) exit
      write (*,*) 'Erro: tente novamente'
    end do
    GetNumber = Input_Value
  end function GetNumber
end program testa_function
```

Subprograma do tipo FUNÇÃO

As Funções podem ser **internas** ou **externas**

- Funções internas (**sem CONTAINS**):

→ Todas as Funções devem ser **iniciadas e finalizadas** separadamente, **após a finalização** do Programa Principal.

```
program teste
  implicit none
  [campo de declaracao de identificadores]
  [campo de execucao]
end program teste
[function 1]
[function 2]
```

- As Funções devem ser **declaradas** no campo de identificadores do Programa Principal.

Subprograma do tipo FUNÇÃO

Função interna: **sem** CONTAINS

```
program testa_function
  implicit none
  real :: GetNumber
  write (*,*)GetNumber()
end program testa_function

real function GetNumber()
  implicit none
  real :: Input_Value
  do
    write (*,*) '_digite um numero positivo: '
    read (*,*)Input_Value
    if (Input_Value > 0.0) exit
    write (*,*) 'Erro: tente novamente'
  end do
  GetNumber = Input_Value
end function GetNumber
```

Declaração de argumentos e identificadores em Funções

- Os argumentos de uma Função podem ser declarados com a opção de intenção **INTENT**:
 - usado para se especificar como é passado um argumento para um subprograma e como ele é retornado.
- Para uma Função a opção de intenção que pode ser usada é o **INTENT(IN)**:
 - neste caso o valor do argumento não pode ser alterado pela Função
- A sintaxe a ser usada é a seguinte
INTENT(IN) :: <lista de argumentos mudos da função>
- Sem a opção de intenção **INTENT(IN)** os valores dos argumentos mudos podem mudar pelos cálculos realizados dentro da Função

Declaração de argumentos e identificadores em Funções

- Forma correta de uso da opção intenção **INTENT(IN)**:

```
program intention
  implicit none
  integer :: a = 10, b = 20
  write(*,*) Add(a,b)
```

CONTAINS

```
integer function Add(x,y)
  implicit none
  integer, intent(in) :: x, y
  Add = x + y
end function Add
end program intention
```

- Como usamos a opção de intenção **INTENT(IN)**, os argumentos mudos **x** e **y** não podem ter seus valores alterados.

Subprograma do tipo FUNÇÃO

Declaração de argumentos e identificadores em Funções

- Forma errada de uso da opção intenção **INTENT(IN)**:

```
program intention
  implicit none
  integer :: a = 10, b = 20
  write (*,*) Add(a,b)
CONTAINS
  integer function Add(a,b)
    implicit none
    integer , intent(in) :: a, b
    b = a + b
    Add = b
  end function Add
end program intention
```

- Os nomes dos argumentos mudos não precisam ser os mesmos daqueles usados no programa que chama a Função. Apenas os tipos devem ser os mesmos.
- Como usamos a opção de intenção **INTENT(IN)**, se tivéssemos usado **a** e **b** como argumentos mudos, daria um erro de compilação.

```
diehl@lua:~/Documents/ensino/UFPEL/prog_comput_fisica/f90$ gfortran teste.f90
teste.f90:14.4:

   b = a + b
   1
Error: Dummy argument 'b' with INTENT(IN) in variable definition context (assignment) at (1)
diehl@lua:~/Documents/ensino/UFPEL/prog_comput_fisica/f90$
```

Declaração de argumentos e identificadores em Funções

IDENTIFICADORES LOCAIS ou INTERNOS (com CONTAINS)

- São identificadores **declarados dentro** da **Função** e têm **validade apenas dentro** do **escopo** da **Função**, ou seja, não podem ser usados fora deste escopo.

IDENTIFICADORES GLOBAIS (com CONTAINS)

- São identificadores **declarados no programa principal** e têm **validade em todo o escopo** do programa, inclusive dentro da **Função**.
- **Não precisam ser transferidos** para a **Função** através dos seus **argumentos** ou qualquer outra instrução do tipo **COMMON**.

Declaração de argumentos e identificadores em Funções

```
program testa_escopo
  implicit none
  real, parameter :: pi = 3.1415926
  integer :: m, n
  ...
CONTAINS
  integer function func1 (k)
    implicit none
    integer, intent(in) :: k
    real :: f, g
    ...
  end function func1
  real function func2(u,v)
    implicit none
    real, intent(in) :: u, v
    integer :: i, j
    ...
  end function func2
end program testa_escopo
```

- Variáveis **globais**

- pi
- m
- n

- Variáveis **locais** de func1

- k
- f
- g

- Variáveis **locais** de func2

- u
- v
- i
- j

Subprograma do tipo FUNÇÃO

Declaração de argumentos e identificadores em Funções

- Um **identificador global** é visível em todo o programa

```
implicit none
integer :: a = 1, b = 2, c = 3
write(*,*) Add(a)
c = 4
write(*,*) Add(a)
write(*,*) Mul(b,c)
CONTAINS
integer function Add(q)
  implicit none
  integer, intent(in) :: q
  Add = q + c
end function Add
integer function Mul(x, y)
  implicit none
  integer, intent(in) :: x, y
  Mul = x * y
end function Mul
end
```

- a, b e c são **globais**
- O primeiro **Add(a)** retorna o valor 4
- O segundo **Add(a)** retorna o valor 5
- Mul(b,c)** retorna o valor 8

Assim, duas chamadas sucessivas da **Função Add(a)** produzem resultados diferentes, mesmo usando os mesmos argumentos mudos: **efeito lateral**.

Declaração de argumentos e identificadores em Funções

- Um **identificador global** é visível em todo o programa

```
program Global
  implicit none
  integer :: a = 10, b = 20
  write(*,*) Add(a,b)
  write(*,*) b
  write(*,*) Add(a,b)
CONTAINS
  integer function Add(x,y)
    implicit none
    integer, intent(in) :: x, y
    b = x + y
    Add = b
  end function Add
end program Global
```

- a, b são **globais**
- O primeiro **Add(a,b)** retorna o valor 30
- O primeiro **Add(a,b)** muda b para 30
- O primeiro **write(*,*)** mostra o valor 30
- O segundo **Add(a,b)** retorna o valor 40

Declaração de argumentos e identificadores em Funções

- Identificador local com o mesmo nome de um identificador global

```
PROGRAM escopo
  IMPLICIT NONE
  INTEGER :: i, Max = 5
  DO i = 1, Max
    Write (*,*) Sum(i)
  END DO
CONTAINS
  INTEGER FUNCTION Sum(n)
    IMPLICIT NONE
    INTEGER, INTENT(IN) :: n
    INTEGER :: i, s
    s = 0
    ...
    Sum = s
  END FUNCTION Sum
END PROGRAM escopo
```

Um identificador local pode ter o mesmo nome de um identificador global

- O identificador inteiro *i*, declarado dentro do programa principal, é global.
- O identificador inteiro *i*, declarado dentro da Função Sum(n), é local.
- Assim, qualquer mudança de *i* dentro da Função não irá alterar o valor da variável global *i* do programa principal.

Subprograma do tipo FUNÇÃO

Declaração de argumentos e identificadores em Funções

Sem o uso do **CONTAINS**, o conceito de identificadores **GLOBAIS** deve ser revisto:

```
implicit none
integer :: a = 1, b = 2, c = 3
integer :: Add, Mul
write(*,*) Add(a,c)
c = 4
write(*,*) Add(a,c)
write(*,*) Mul(b,c)
end
integer function Add(q,c)
  implicit none
  integer, intent(in) :: q, c
  Add = q + c
end function Add
integer function Mul(x,y)
  implicit none
  integer, intent(in) :: x, y
  Mul = x * y
end function Mul
```

- As variáveis só têm validade no escopo da unidade de programa onde foram definidas.
- Para serem usadas em unidades de programa distintas, as **variáveis devem ser transferidas** para a **Função** através dos seus **argumentos mudos**.

Declaração de argumentos e identificadores em Funções

- O atributo **SAVE**:

→ usado para se **preservar o valor de variáveis**, declaradas em uma **Função**.
Desta forma, o valor anterior desta variável está acessível quando a **Função** é invocada novamente.

```
program testa_save
  integer :: i
  i = SUB()
  i = SUB()
  i = SUB()
CONTAINS
  integer FUNCTION SUB()
    integer, SAVE :: a = 0
    a = a + 1
    print *, 'a= ', a
  end function SUB
end program testa_save
```

- O **valor inicial** de **a** é 0.
- Na **primeira chamada** da **Função** o valor de **a** é modificado para 1.
- Na **segunda chamada** da **Função** o valor de **a** é modificado para 2.
- Na **terceira chamada** da **Função** o valor de **a** é modificado para 3.

Declaração de argumentos e identificadores em Funções

- O atributo **DATA**:
 - usado **inicializar variáveis** declaradas em uma unidade de programa.

```
program testa_data
  real :: x, y
  DATA x, y / -1.0, 6.0 /
  print*,x,y
  print*, 'FUNC_=_',FUNC()
CONTAINS
  real FUNCTION FUNC()
    integer :: j
    DATA j / 100 /
    print*,j
    print*, 'x_=_',x
    FUNC = 2.0*x
  end function FUNC
end program testa_data
```

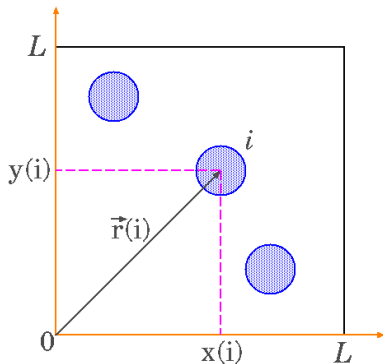
→ as variáveis inicializadas com o atributo **DATA** podem ter seus valores modificados.

```
diehl@lua:~/Documents/ensino/UFPEL/prog_comput_fisica/f9
-1.00000000    6.00000000
FUNC =          100
x =    -1.00000000
-2.00000000
```

Obs.: Uso inadequado da Função!

Subprograma do tipo FUNÇÃO

Exemplo: Programa para inserir N partículas numa caixa quadrada de lado L , com as posições fixadas de forma aleatória.



Subprograma do tipo FUNÇÃO

Rotina para gerar uma sequência de **números pseudoaleatórios**: **ran2.f90** (Numerical Recipes, <http://www.nrbook.com/a/bookf90pdf.html>)

```
REAL FUNCTION ran2 (idum)
  INTEGER idum, idum2
  INTEGER IM1, IM2, IMM1, IA1, IA2, IQ1, IQ2, IR1, IR2, NTAB, NDIV
  REAL AM, EPS, RNMX
  PARAMETER (IM1=2147483563, IM2=2147483399, AM=1./IM1,
             IMM1=IM1-1, IA1=40014, IA2=40692, IQ1=53668, &
             IQ2=52774, IR1=12211, IR2=3791, NTAB=32, &
             NDIV=1+IMM1/NTAB, EPS=1.2e-7, RNMX=1.-EPS)
  INTEGER j, k, iv (NTAB), iy
  SAVE iv, iy, idum2
  DATA idum2/123456789/, iv/NTAB*0/, iy/0/
  [... A ...]
```

A argumento mudo **idum** deve ser um **inteiro negativo** na **primeira chamada** de **ran2**.

Subprograma do tipo FUNÇÃO

Rotina para gerar uma sequência de **números pseudoaleatórios**: **ran2.f90** (Numerical Recipes, <http://www.nrbook.com/a/bookf90pdf.html>)

```
[... A ...]
if (idum <= 0) then
  idum = MAX(-idum,1)
  idum2 = idum
  do j = NTAB+8,1,-1
    k = idum/IQ1
    idum = IA1*(idum-k*IQ1) - k*IR1
    if (idum.lt.0) idum = idum + IM1
    if (j <= NTAB) iv(j) = idum
  enddo
  iy = iv(1)
endif
k = idum/IQ1
idum = IA1*(idum-k*IQ1) - k*IR1
[... B ...]
```

A argumento mudo **idum** deve ser um **inteiro negativo** na **primeira chamada** de **ran2**.

Subprograma do tipo FUNÇÃO

Rotina para gerar uma sequência de **números pseudoaleatórios**: **ran2.f90** (Numerical Recipes, <http://www.nrbook.com/a/bookf90pdf.html>)

```
[ ... B ... ]  
if (idum < 0) idum = idum + IM1  
k = idum2/IQ2  
idum2 = IA2*(idum2-k*IQ2) - k*IR2  
if (idum2 < 0) idum2 = idum2 + IM2  
j = 1 + iy/NDIV  
iy = iv(j) - idum2  
iv(j) = idum  
if (iy < 1) iy = iy + IMM1  
ran2 = MIN(AM*iy ,RNMX)  
return  
END FUNCTION ran2
```

Use **ran2** para inserir as N partículas na caixa quadrada. Use alocação dinâmica, entrada de dados a partir de arquivos e a saída das coordenadas das partículas num arquivo.

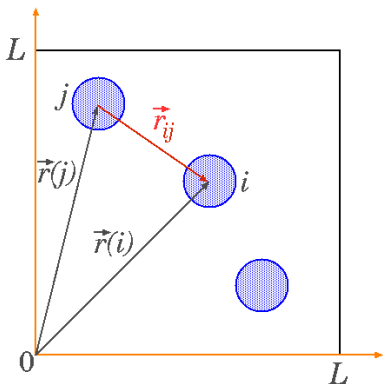
Subprograma do tipo FUNÇÃO

```
program insere_particulas
  implicit none
  integer :: i, N, iseed
  real :: L
  real, allocatable :: x(:), y(:)
  print*, 'quantas partículas?'
  read*, N
  allocate (x(N), y(N))
  print*, 'qual o tamanho L da caixa?'
  read*, L
  print*, 'digite um inteiro negativo:'
  read*, iseed
  do i = 1, N
    x(i) = L*ran2(iseed)
    y(i) = L*ran2(iseed)
    print*, x(i), y(i)
  end do
CONTAINS
  real FUNCTION ran2(idum)
  [ ... ]
  end FUNCTION ran2
end program insere_particulas
```

```
diehl@lua:~/Documents/ensino/UFPEL/pr
quantas partículas?
4
qual o tamanho L da caixa?
10.0
digite um inteiro negativo:
-2234
3.07763863      2.06435397E-02
3.64427662      9.00589752
3.82275534      8.20053482
5.61893940      2.34738731
diehl@lua:~/Documents/ensino/UFPEL/pr
```

Subprograma do tipo FUNÇÃO

Exemplo: Programa para calcular a distância centro-a-centro de N partículas numa caixa quadrada de lado L , com as posições fixadas de forma aleatória.



$$\vec{r}_{ij} = \vec{r}(i) - \vec{r}(j)$$

distância centro-a-centro:

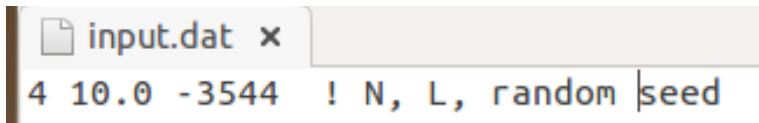
$$|\vec{r}_{ij}| = \sqrt{(x(i) - x(j))^2 + (y(i) - y(j))^2}$$

Subprograma do tipo FUNÇÃO

```
program insere_particulas
  implicit none
  integer :: i, j, N, iseed
  real :: L, distancia
  real, allocatable :: x(:), y(:)
  read*,N, L, iseed
  allocate(x(N),y(N))
  do i = 1,N
    x(i) = L*ran2(iseed)
    y(i) = L*ran2(iseed)
  end do
  do i = 1,N-1
    do j = i+1,N
      distancia = sqrt((x(i)-x(j))**2+(y(i)-y(j))**2)
      write(*,'(A,I3,2x,A,I3,2x,A,F6.3)') 'i=',i,'j=',j,&
        'distancia=',distancia
    end do
  end do
CONTAINS
  real FUNCTION ran2(idum)
  [ ... ]
  end FUNCTION ran2
end program insere_particulas
```

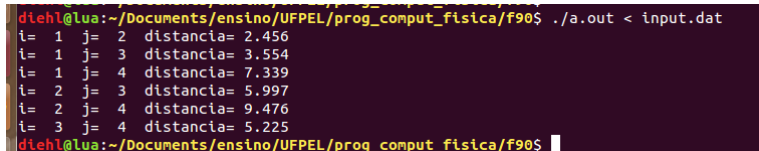
Subprograma do tipo FUNÇÃO

Formato do arquivo de entrada input.dat:



```
input.dat x
4 10.0 -3544 ! N, L, random seed
```

Execução e saída do código:



```
diehl@lua:~/Documents/ensino/UFPEL/prog_comput_fisica/f90$ ./a.out < input.dat
i= 1 j= 2 distancia= 2.456
i= 1 j= 3 distancia= 3.554
i= 1 j= 4 distancia= 7.339
i= 2 j= 3 distancia= 5.997
i= 2 j= 4 distancia= 9.476
i= 3 j= 4 distancia= 5.225
diehl@lua:~/Documents/ensino/UFPEL/prog_comput_fisica/f90$
```