

Curso de Introdução a Programação com Fortran

Paulo Yoshio Kubota
Outubro, 2014

Curso de Introdução a Programação com Fortran

- Porque Fortran?
 - Linguagem desenvolvida para realizar operações em ponto flutuante com eficiência.
 - Vários software científicos escrito em fortran (bem, muitas vezes em Fortran 77).
 - Abstração de alto nível (em relação a C); por exemplo, manipulação de matrizes.
 - Grande número de bibliotecas numéricas construído para Fortran.
 - Compiladores (especialmente os livres) são fáceis de encontrar em diferentes ambientes.
 - Algumas coisas mais fáceis de implementar em relação ao C.

Curso de Introdução a Pragração com Fortran

EMENTA: Conceitos Básicos:

- Definições da Linguagem.
- Tipos intrínsecos de variáveis (INTEGER, REAL, COMPLEX, CHARACTER, LOGICAL).
- Variáveis escalares e Inicialização.
- Expressões.
- Atribuições.
- Arrays e Ponteiros:
- Funções Intrínsecas
- Expressões com Arrays.
- Atribuições a Arrays e Ponteiros.
- Expressões e Atribuições com Ponteiros.
- Tipos definidos pelo usuário.
- Declarações, inicialização, expressões e atribuições.
- Comandos para Controle de Fluxo: If, Do.
- Estrutura de Programas: Funções, Subrotinas, Módulo, Programa Principal.
- Passagem de Argumentos a Funções e Subrotinas.
- Usos de Módulos. Encapsulamento.
- Comandos e Formatos de Entrada/Saída: Open, Close, Read, Print, Write.
- Compilação e Execução. Exercícios – FORTRAN

Histórico

FORmula TRANslation invented 1954–8
by John Backus and his team at IBM
FORTRAN 66 (ISO Standard 1972)
FORTRAN 77 (1980)
Fortran 90 (1991)
Fortran 95 (1996)
Fortran 2003 (2004)
Fortran 2008 (ongoing)

<http://ufpel.edu.br/~rudi/grad/ModComp/Apostila/html/Apostila.html#Apostilach7.html>

Compiladores

INTEL (ifort, ifc) <http://software.intel.com/en-us/intel-compilers>

GFORTTRAN (gfortran) <http://gcc.gnu.org/wiki/GFortran>

G95 (g95) <http://www.g95.org/>

PORTLAND (pgf90) <http://www.pggroup.com/>

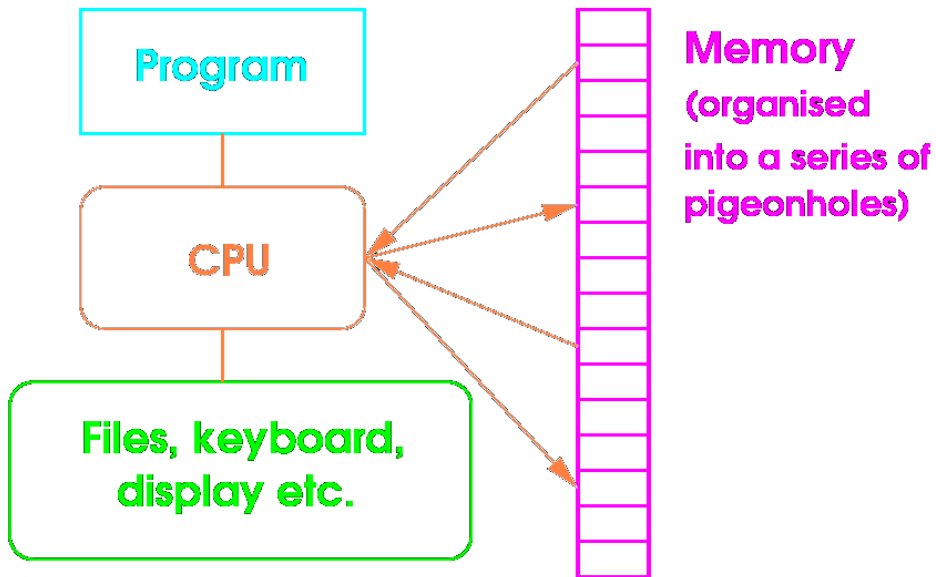
.....

.....

Fortran Programming Model

- Programa-Fonte.
- Programa-Objeto.
- Programa executável.

```
PROGRAM nome  
  IMPLICIT NONE  
  CONTAINS  
END PROGRAM nome
```



Para compilar um programa
(Gerar e rodar o executável)

```
ifort -c name.f90  
name.o
```

```
ifort -o name.exe name.o  
./name.exe
```

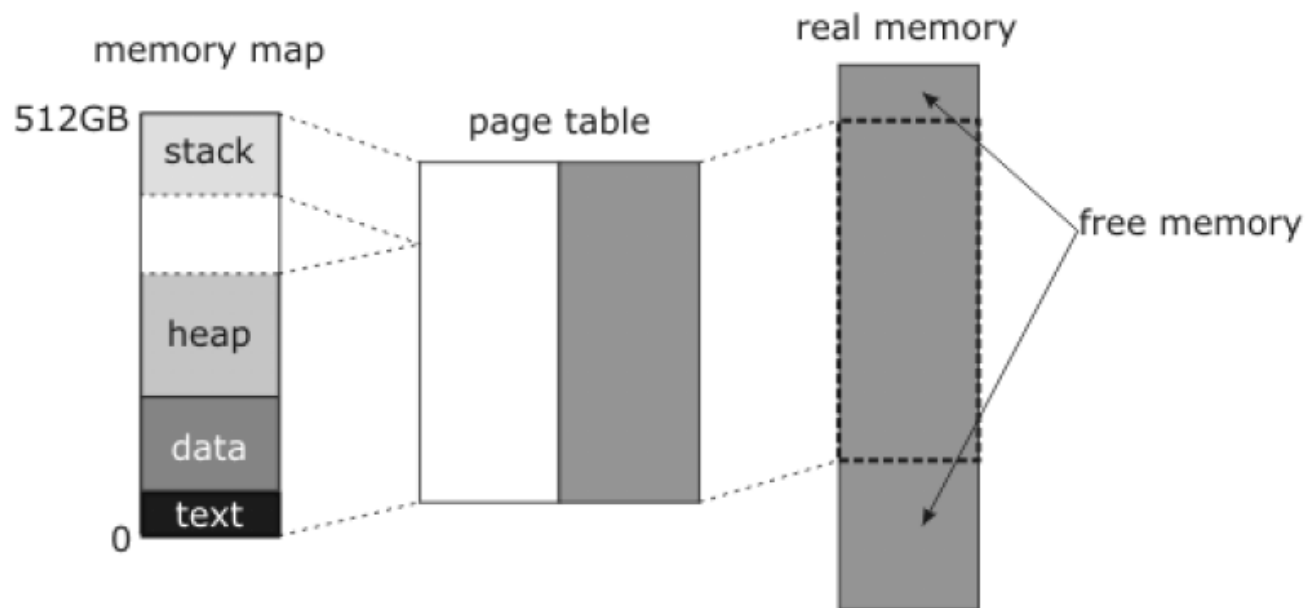
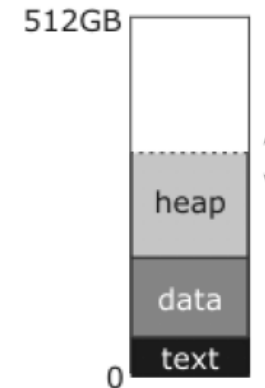
OU

```
ifort -o name.exe name.f90  
./name.exe
```

```
ifort name.f90  
./a.exe
```

Fortran Programming Model

- Programa-Fonte.
- Programa-Objeto.
- Programa executável.



Formato (Estrutura) do Código

- Um Programa Fortran 90 tem a seguinte forma:
- *nome do programa* é o nome do programa (**Main**)
- *parte-especificação*, *parte-execução*, and *parte-subprograma* São opcionais.
- Embora **IMPLICIT NONE** seja também opcional, Ele é **Essencial** para escrever programas seguros.

```
PROGRAM nome_do_programa
  IMPLICIT NONE
  [parte-especificação]
  [parte-execução]
CONTAINS
  [parte-subprograma]
END PROGRAM nome_do_programa
```


Formato (Estrutura) do Código

PROGRAM Nome_programa

! Este é um Comentário

IMPLICIT NONE

type e definição de variáveis

.

.

executable statements

.

.

STOP

CONTAINS *! Comentário*

definição de subrotinas local

.

END PROGRAM Nome_programa

SUBROUTINE sub(p1,p2)

IMPLICIT NONE

definição de procedure e parameter

definição de type e variable local

.

.

executable statements

.

.

RETURN

CONTAINS *! Comentário*

definição de subrotinas local

.

.

END SUBROUTINE sub

FUNCTION func(p1,p2)

IMPLICIT NONE

definição de funções e seus parametros

definição de type e variable local

.

executable statements

.

.

func=... *! assign a return value*

RETURN

CONTAINS *! Comment again*

local subroutine definitions

.

.

END FUNCTION func

F90 Comentários do Programa

- Inicia-se um comentário com um !
- O que segue após ! será ignorado
- Isto é similar a // em C/C++

exercicio001

! Este é um Exemplo

!

PROGRAM Nome_do_Programa

IMPLICIT NONE

INTEGER :: Year !Variável Year

READ(*,*) Year ! Lê o valor de Year através do terminal

Year = Year + 1 ! Adiciona 1 ao Year

PRINT*, 'Year=', Year

END PROGRAM Nome_do_Programa

F90 Continuação de linhas

- Fortran 90 não é completamente format-free!
- Uma nova **ação** deve começar com uma nova linha.
- Se uma **ação** é muito longa para ajustar uma nova linha a ação deve ser *continuada*.
- O caracter de continuação de linha é **&**, que não faz parte da ação.

```
Total = Total + &  
Amount * Payments  
! Total = Total + Amount*Payments
```

```
PROGRAM &  
ContinuationLine  
! PROGRAM ContinuationLine  
END PROGRAM ContinuationLine
```

Alfabetos do F90

- Alfabeto do fortran 90 inclui os seguintes conjuntos:
- Letras Maiúsculas e Minúsculas
- Dígitos
- Caracteres especiais

space

' "

() * + - / : =

_ ! & \$ % ; < >

% ? , .

Os caracteres alfanuméricos válidos são:

- Letras (a - z; A - Z).
- Numerais (0 - 9).
- O caractere underscore “_”.

A única restrição é que o primeiro caractere seja uma letra em declarações de: variáveis, funções, subrotinas, modulo, type, etc

Identificadores: F90

- Um identificador Fortran 90 pode ter não mais do que 31 caracteres.
- O primeiro deve ser uma letra. Os demais caracteres do identificador, pode ser letras, dígitos ou underscores.
- **Identificador em *Fortran 90* é insensível a letras maiúsculas e minúsculas**
- Exemplos: **A**, **Name**, **toTAL123**, **System_**, **myFile_01**, **my_1st_F90_program_X_**.
- Identificadores **Name**, **nAmE**, **naME** e **NaME** são os mesmos.

```
PROGRAM my_1st_F90_program_X_  
IMPLICIT NONE  
READ(*,*) Year, A,Name,toTal123,System_,muFile_01, my_1st_F90  
! INTEGER :: 1A ! erro  
Year = Year + 1  
CALL NaME ()  
END PROGRAM my_1st_F90_program_X_
```

Declaração do Tipo de Variável

A forma geral de uma declaração de tipo de variáveis é:

<tipo>[(**[KIND=]**<parâmetro de espécie>)][,<lista de atributos>] **::** <lista de entidades>

INTEGER
REAL
CHARACTER
LOGICAL
COMPLEX

KIND=single
KIND=doble

,PARAMETER
,TARGET
,ALLOCATABLE
,DIMENSION(<lista de extensões>)
,PUBLIC
,INTENT(<inout>)
,PRIVATE
,OPTIONAL
,POINTER
,EXTERNAL
,INTRINSIC
,SAVE

:: var1
:: var2

```
PROGRAM Nome_do_Programa
IMPLICIT NONE
INTEGER      :: iYear!Variável Year
REAL         :: rYear!Variável Year
CHARACTER    :: cYear!Variável Year
LOGICAL      :: lYear!Variável Year
COMPLEX      :: xYear!Variável Year
REAL(5,*)iYear
PRINT*, 'Year=', iYear
END PROGRAM Nome_do_Programa
```

VARIÁVEIS DO TIPO INTEGER F90

- Novas funções intrínsecas que permiti que o usuário selecione a opção "tipo" de variável inteira ou real.
- **SELECTED_INT_KIND(I)**
 - Devolver o tipo do valor de uma variável inteira que representa todos os valores entre -10^I a 10^I
- **SELECTED_REAL_KIND(P,R)**
 - Devolver o tipo do valor de uma variável real com precisão decimal de **P** dígitos e o expoente variando entre **-R** a **R**.

VARIÁVEIS DO TIPO INTEGER F90

- Uma **constante inteira (integer constant)** é uma string de dígitos com a opção de sinal: **12345**, **-345**, **+789**, **+0**.

```
PROGRAM ConstantInteger
  IMPLICIT NONE
  INTEGER, PARAMETER :: i4=SELECTED_INT_KIND(9)
  INTEGER(KIND=i4) :: CycleLife=10_i4
END PROGRAM ConstantInteger
```

```
PROGRAM Inteiro
  IMPLICIT NONE
```

```
  INTEGER :: X
```

! O valor digitado não pode conter ponto (.) Caso isto
! aconteça, vai gerar um erro de execução no programa,
! abortando o mesmo.

```
  READ *, X
```

```
  PRINT*, "Valor lido:",X
```

```
END PROGRAM Inteiro
```

Precisão numérica para um número inteiro 32bits

Padrão 32bits

VARIÁVEIS DO TIPO INTEGER F90

- Uma **constante inteira (integer constant)** é uma string de dígitos com a opção de sinal: **12345**, **-345**, **+789**, **+0**.

exercicio002

```
PROGRAM intkind
IMPLICIT NONE
INTEGER,PARAMETER :: ik=SELECTED_INT_KIND(2)
INTEGER(kind=ik) :: i
INTEGER :: j
i=1
DO j=1,20
    i=i*2
    WRITE(0,*) j,i
END DO
STOP
END PROGRAM intkind
```

```
progs> ifort intkind.f90
progs> ./a.out
```

1	2
2	4
3	8
4	16
5	32
6	64
7	-128
8	0
9	0
10	0
11	0
12	0
13	0
14	0
15	0

Mude o **selected_int_kind(2)** para **selected_int_kind(4)** no código

```
progs> ifort intkind.f90
progs> a.out
```

1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
10	1024
11	2048
12	4096
13	8192
14	16384
15	-32768

selected_int_kind(p) retorna o kind do tipo para representar todos os inteiros n variando entre $-10^p < n < 10^p$.

VARIÁVEIS DO TIPO REAL

- Uma **variável real** tem duas formas, **decimal e exponencial**:
- Na **Forma decimal**, uma variável real é uma string de dígitos com exatamente um ponto decimal. Uma variável real pode ter uma opção de sinal.
- Exemplo: **2.45, .13, 13., -0.12, -.12.**

```
PROGRAM ConstantReal  
  IMPLICIT NONE  
  INTEGER, PARAMETER :: r4=SELECTED_REAL_KIND(6)  
  REAL(KIND=r4)      :: FractionLife=0.010_r4  
END PROGRAM ConstantReal
```

Precisão numérica para um número real

VARIÁVEIS DO TIPO REAL

- Na Forma **exponencial**, uma variável real inicia com um inteiro/real, seguido pôr um **E/e**, e pôr um inteiro (*i.e.*, o expoente).

Exemplos:

- **12E3** (12×10^3), **-12e3** (-12×10^3), **3.45E-8** (3.45×10^{-8}), **-3.45e-8** (-3.45×10^{-8}).
- **0E0** ($0 \times 100 = 0$). {**12.34-5** está errado!}

```
PROGRAM ConstantReal
  IMPLICIT NONE
  INTEGER, PARAMETER :: r4=SELECTED_REAL_KIND(6)
  REAL(KIND=r4)      :: FractionLife=1.0e-2_r4
END PROGRAM ConstantReal
```

VARIÁVEIS DO TIPO REAL

exercicio003

```
PROGRAM realkind
IMPLICIT NONE
INTEGER,PARAMETER :: rk=SELECTED_REAL_KIND(6,15)
REAL(kind=rk) :: x
INTEGER :: j
x=1.0
DO j=1,64
    x=x*100.0
    WRITE(0,*) j,x
END DO
STOP
END PROGRAM realkind
```

```
progs> gfortran realkind.f90
progs> a.out
1 100.0000
2 10000.00
3 1000000.
4 1.00000000E+08
5 1.00000000E+10
...
18 1.00000001E+36
19 1.00000001E+38
20 +Infinity
21 +Infinity
22 +Infinity
...
```

Use (10,40) no lugar de (6,15)

```
progs> gfortran realkind.f90
progs> ./a.out
1 100.00000000000000
2 10000.000000000000
3 1000000.0000000000
4 1000000000.0000000
...
153 9.999999999999997E+305
154 9.999999999999996E+307
155 +Infinity
156 +Infinity
...
-
```

selected_real_kind(p,r) retorna o *minimo de precisão decimal* (**p**) e a *variação do expoente* (**r**) do tipo real.

Problema 1. (6 pontos)

Estude qual kind de tipos inteiro e real o seu compilador suporta.

exercicio004

```
PROGRAM testkind
  IMPLICIT NONE
  INTEGER :: i,ki,kr
  DO i=1,24
    ki=SELECTED_INT_KIND(i)
    kr=SELECTED_REAL_KIND(i)
    PRINT *,i,ki,kr
  END DO
END PROGRAM testkind
```

Problema 2. (6 pontos)

Encontre todos os erros no seguinte programa:

exercicio005

```
PROGRAM buggy
IMPLICIT NONE
INTEGER :: i
REAL :: pi=3*ARCTAN(1.0)
PRINT *, 'pi = ' _ pi
i=4
j=10.0-i
PRINT *, 'i/(10-i) =', i/j
END PROGRAM buggy
```

```
PROGRAM buggy
IMPLICIT NONE
INTEGER :: i
REAL :: j, pi=4*atan(1.0)
PRINT *, 'pi = ', pi
i=4
j=10.0-i
PRINT *, 'i/(10-i) =', i/j
END PROGRAM buggy
```

Problema 3. (8 pontos)

Investigue o tempo de cpu requerido por diferentes operações matemáticas (+, -, *, **, exp).
. Use a rotina CPU_TIME para medir o tempo de cpu.

exercicio006

PROGRAM OPERATIONS

INTEGER, PARAMETER :: r4=SELECTED_REAL_KIND(5,10)

INTEGER, PARAMETER :: r8=SELECTED_REAL_KIND(10,40)

INTEGER, PARAMETER :: r16=SELECTED_REAL_KIND(30,200)

REAL(KIND=r8) :: time0,time1

REAL(KIND=r8) :: x,y

INTEGER :: i,j

CALL cpu_time(time0)

do i=1,10000

do j=1,10000

(operation you want to measure)

end do

end do

CALL cpu_time(time1)

PRINT *, 'CPU time = ',time1-time0,

END PROGRAM OPERATIONS

exercicio007

```
program cputime
  implicit none
  integer,parameter :: r=selected_real_kind(10,40)
  !integer,parameter :: rk=selected_real_kind(5,10)
  integer,parameter :: rk=selected_real_kind(10,40)
  !integer,parameter :: rk=selected_real_kind(30,200)
  real(r) :: t0,tnop,tplus,tminus,tmult,tpow,texp
  real(r),parameter :: nano=1.0e9
  real(rk) :: op1,op2,x,xnop,xplus,xminus,xmult,xpow,xexp
  integer,parameter :: opmax=10
  integer :: i,j,imax
  integer,external :: iargc
  character(len=80) :: argu
  !call getarg(1,argu)
  read(5,*) imax
  x=0.0
  call cpu_time(t0)
  do i=1,imax
    do j=1,imax
      op1=mod(i,opmax)
      op2=mod(j,opmax)+1.0
      !x=x+op1+op2
    end do
  end do
  call cpu_time(tnop)
  tnop=tnop-t0
  xnop=x
  x=0.0
  call cpu_time(t0)
```

```
do i=1,imax
  do j=1,imax
    op1=mod(i,opmax)
    op2=mod(j,opmax)+1.0
    x=x+op1+op2
  end do
end do
call cpu_time(tplus)
tplus=tplus-t0
xplus=x
x=0.0
call cpu_time(t0)
do i=1,imax
  do j=1,imax
    op1=mod(i,opmax)
    op2=mod(j,opmax)+1.0
    x=x+op1-op2
  end do
end do
call cpu_time(tminus)
tminus=tminus-t0
xminus=x
x=0.0
call cpu_time(t0)
do i=1,imax
  do j=1,imax
    op1=mod(i,opmax)
    op2=mod(j,opmax)+1.0
    x=x+op1*op2
  end do
end do
```



```

call cpu_time(tmult)
tmult=tmult-t0
xmult=x
x=0.0
call cpu_time(t0)
do i=1,imax
    do j=1,imax
        op1=mod(i,opmax)
        op2=mod(j,opmax)+1.0
        x=x+op1**op2
    end do
end do
call cpu_time(tpow)
tpow=tpow-t0
xpow=x
x=0.0
call cpu_time(t0)
do i=1,imax
    do j=1,imax
        op1=mod(i,opmax)
        op2=mod(j,opmax)+1.0
        x=x+exp((op1+op2))
    end do
end do
call cpu_time(texp)
texp=texp-t0
xexp=x

```

```

print '(a,3g24.12)', 'nop ', nano*tnop/(imax**2), tnop, xnop
print '(a,3g24.12)', 'plus ', nano*tplus/(imax**2), tplus, xplus
print '(a,3g24.12)', 'minus ', nano*tminus/(imax**2), tminus, xminus
print '(a,3g24.12)', 'mult ', nano*tmult/(imax**2), tmult, xmult
print '(a,3g24.12)', 'pow ', nano*tpow/(imax**2), tpow, xpow
print '(a,3g24.12)', 'exp ', nano*texp/(imax**2), texp, xexp
stop
end program cputime

```

VARIÁVEIS DO TIPO LOGICAL

Uma variável lógica pode assumir dois valores **.TRUE.** ou **.FALSE.**

- A combinação de duas variáveis lógicas também resulta em **.TRUE.** ou **.FALSE.!**

Exercicio008 - > **construa novas instruções**

```
PROGRAM LOGICO
  IMPLICIT NONE
  LOGICAL :: A= .TRUE.
  LOGICAL :: b=.TRUE.
  INTEGER :: c=10
  IF (A) THEN
    PRINT*, "A VARIÁVEL É VERDADEIRA."
  END IF
  b=c<1
  IF(b) PRINT*, "A VARIÁVEL É VERDADEIRA."
  IF(.not. b) PRINT*, "A VARIÁVEL É FALSE."
END PROGRAM LOGICO
```

VARIÁVEIS DO TIPO CHARACTER

- Uma **string de caracteres** ou variável **character** é uma string de caracteres fechada entre duas aspas dupla ou simples . Exemplos:
“abc”, ‘John Dow’, “#\$%^”, and ‘()’.
- O conteúdo de uma string character consiste de todos os caracteres entre as aspas. Exemplo: O conteúdo de ‘John Dow’ is John Dow.
- O comprimento de uma string é o numero de caracteres entre as aspas. O comprimento de ‘John Dow’ é de 8, incluído o espaço em branco.
- se uma string tem comprimento zero (*i.e.*, sem conteúdo) é uma string vazia (**empty string**).

VARIÁVEIS DO TIPO CHARACTER

Exercicio009

```
PROGRAM LE_CARACTERE
  IMPLICIT NONE
  CHARACTER(LEN=10) :: STR_READ
  PRINT *, "ENTRE COM TEXTO:"
  READ '(A)', STR_READ
  PRINT *, "TEXTO LIDO:", STR_READ
END PROGRAM LE_CARACTERE
```

VARIÁVEIS DO TIPO COMPLEX

- São ou inteiros ou reais, separados por vírgula “,” e contidos entre parênteses “(“ e “)”.
Os **literais complexos** representam números contidos no conjunto dos números complexos, isto é, números do tipo $z = x + iy$, onde $i = (-1)^{0.5}$, x é a parte real e y é a parte imaginária do número complexo z .
(<parte real>,<parte imaginária>), (1.,3.2) (representando $1.0 + 3.2i$)

Exercicio010

```
PROGRAM LE_COMPLEX
```

```
IMPLICIT NONE
```

```
COMPLEX :: a= (5,-5),b,c ! Variável a é inicializada a (5,-5).
```

```
PRINT *, "Valor de b:"
```

```
! O valor de b deve ser entrado como um literal complexo.
```

```
! Exemplo: (-1.5,2.5)
```

```
READ *, b
```

```
c= a*b
```

```
PRINT *, "O valor de c:", c
```

```
! Verifique o resultado no papel.
```

```
END PROGRAM LE_COMPLEX
```

O que é **KIND** ?

- Fortran 90 tem um atributo (função intrínseca) **KIND** para selecionar a precisão de uma variável numérica.
- O **KIND** de uma variável/constante é um inteiro positivo

Exemplo:

- **126_3** : 126 é um inteiro de **KIND 3**
- **3.1415926_8** : 3.1415926 é um real de **KIND 8**

```
PROGRAM TES_FUN_KIND
  IMPLICIT NONE
  INTEGER :: I
  INTEGER :: J
  INTEGER, PARAMETER :: DP= 2
  REAL :: Y
  REAL(KIND= DP) :: X
  !
  I= KIND(X) ! I= 2
  J= KIND(Y) ! DEPENDE DO SISTEMA
               ! (J=1 PARA COMPILADOR F).

  PRINT*, I
  PRINT*, J
END PROGRAM TES_FUN_KIND
```

KIND(0)	! Retorna a espécie padrão do tipo inteiro. ! (Dependente do processador).
KIND(0.0)	! Retorna a espécie padrão do tipo real. ! (Depende do processador).
KIND(.FALSE.)	! Retorna a espécie padrão do tipo lógico. ! (Depende do processador).
KIND('A')	! Fornece a espécie padrão de caractere. ! (Sempre igual a 1).
KIND(0.0D0)	! Usualmente retorna a espécie do tipo real de ! precisão dupla. ! (Pode não ser aceito por todos compiladores)

Exercicio011

O que é **KIND** ?

•As funções intrínsecas **SELECTED_INT_KIND(R)** e **SELECTED_REAL_KIND(P,R)** tem dois argumentos opcionais: **P** e **R**. A variável **P** especifica a precisão (número de dígitos decimais) mínima requerida e **R** especifica o intervalo de variação mínimo da parte exponencial da variável -10^R a $+10^R$.

Exercicio012

```
PROGRAM TES_SELECTED
INTEGER, PARAMETER :: I10= SELECTED_REAL_KIND(10,200)
INTEGER, PARAMETER :: DP= 8
REAL(KIND= I10) :: A,B,C
REAL(KIND= DP) :: D
PRINT*, I10
A= 2.0_I10
B= SQRT(5.0_I10)
C= 3.0E10_I10
D= 1.0E201_DP
PRINT*, A
PRINT*, B
PRINT*, C
PRINT*, D
END PROGRAM TES_SELECTED
```

O que é **KIND** ?

Exercicio013

PROGRAM PROG01

IMPLICIT NONE

! program reads in real number x and
! computes function value f(x).
! **KIND(a)** returns kind parameter of argument
! **TINY(a)** The smallest positive number
! **HUGE(a)** The largest positive number
! **EPSILON(a)** The least positive number that
! added to 1 returns a number
! that is greater than 1
! **PRECISION(a)** The decimal precision
! **DIGITS(a)** The number of significant digits
! **RANGE(a)** The decimal exponent
! **MAXEXPONENT(a)** The largest exponent
! **MINEXPONENT(a)** The smallest exponent
! **Selecting Kinds**

INTEGER, PARAMETER :: &

sp = SELECTED_REAL_KIND(6,30), &

dp = SELECTED_REAL_KIND(10,200)

REAL(KIND=sp) :: a

REAL(KIND=dp) :: b

DOUBLE PRECISION :: c

PRINT *, 'The kind number of argument ->', **KIND(a)**
PRINT *, 'The largest positive number ->', **HUGE(a)**
PRINT *, 'The smallest positive number ->', **TINY(a)**
PRINT *, 'The decimal exponent ->', **RANGE(a)**
PRINT *, 'The decimal precision ->', **PRECISION(a)**
PRINT *, 'The number of significant digits->', **DIGITS(a)**
PRINT *, 'The largest exponent ->', **MAXEXPONENT(a)**
PRINT *, 'The smallest exponent ->', **MINEXPONENT(a)**
PRINT *, ''
PRINT *, 'The kind number of argument ->', **KIND(b)**
PRINT *, 'The largest positive number ->', **HUGE(b)**
PRINT *, 'The smallest positive number ->', **TINY(b)**
PRINT *, 'The decimal exponent ->', **RANGE(b)**
PRINT *, 'The decimal precision ->', **PRECISION(b)**
PRINT *, 'The number of significant digits->', **DIGITS(b)**
PRINT *, 'The largest exponent ->', **MAXEXPONENT(b)**
PRINT *, 'The smallest exponent ->', **MINEXPONENT(b)**
END PROGRAM PROG01

!INTEGER, PARAMETER :: r4 = SELECTED_REAL_KIND(6) ! Kind for 32-bits Real Numbers
!INTEGER, PARAMETER :: i4 = SELECTED_INT_KIND(9) ! Kind for 32-bits Integer Numbers
!INTEGER, PARAMETER :: r8 = SELECTED_REAL_KIND(15) ! Kind for 64-bits Real Numbers
!INTEGER, PARAMETER :: i8 = SELECTED_INT_KIND(14) ! Kind for 64-bits Integer Numbers
!INTEGER, PARAMETER :: r16 = SELECTED_REAL_KIND(31) ! Kind for 128-bits Real Numbers

Tipos Derivados TYPE

- Função que disponibiliza ao programador de definir seus próprios tipos de variáveis.

A forma geral da declaração de um tipo derivado é:

```
TYPE [[,<acesso>] ::] <nome do tipo>  
  [PRIVATE]  
  <declarações de componentes>  
END TYPE [<nome do tipo>]  
TYPE (<nome do tipo>) [::] <lista de nomes>
```

```
PROGRAM def_tipo_der  
  IMPLICIT NONE  
  TYPE :: ponto  
    real :: x, y, z  
  END TYPE  
  TYPE (ponto) :: centro, apice  
  apice%x= 0.0  
  apice%y= 1.0  
  apice%z= 0.0  
  centro = ponto(0.0,0.0,0.0)  
  PRINT*, apice  
  PRINT *, centro  
END PROGRAM def_tipo_de
```

Exercicio014

```
TYPE :: ENTRY  
  REAL :: VALOR= 2.0  
  INTEGER :: INDEX  
  TYPE(ENTRY), POINTER :: NEXT => NULL()  
END TYPE ENTRY  
  
TYPE :: NODO  
  INTEGER :: CONTA  
  TYPE(ENTRY) :: ELEMENTO  
END TYPE NODO  
  
TYPE(NODO) :: N  
TYPE(ENTRY), DIMENSION(100) :: MATRIZ
```

Tipos Derivados TYPE

Exercicio015

```
PROGRAM typedeftest  
IMPLICIT NONE
```

```
TYPE person  
  CHARACTER(len=20) :: first,last  
  INTEGER :: age  
END TYPE person
```

```
TYPE(person) :: student
```

```
student%first='Matti'  
student%last='Meikäläinen'  
student%age=75
```

```
! Not sure whether outputting the whole  
! struct is standard conforming.
```

```
WRITE(6,*) student
```

```
STOP
```

```
END PROGRAM typedeftest
```

```
progs> ifort typedef.f90  
progs> a.out  
Matti Meikäläinen 75
```

EXPRESSÕES E ATRIBUIÇÕES ESCALARES

- Expressão em Fortran 90/95 é formada de operandos e operadores

A/B/C ou A/(B*C)

Type	Operator						Associativity
Arithmetic	**						<i>right to left</i>
	*		/				left to right
	+		-				left to right
Relational	<	<=	>	>=	==	/=	none
Logical	.NOT.						<i>right to left</i>
	.AND.						left to right
	.OR.						left to right
	.EQV.		.NEQV.				left to right

*highest
priority*



24

Tabela 4.6: Tabela-Verdade .NOT.

A	.NOT. A
T	F
F	T

Tabela 4.7: Tabela-Verdade .AND.

A	B	A .AND. B
T	T	T
T	F	F
F	T	F
F	F	F

Tabela 4.8: Tabela-Verdade .OR.

A	B	A .OR. B
T	T	T
T	F	T
F	T	T
F	F	F

- Atribuições numéricas escalares

I= 7

A=2.5

Cpx=(2.1,2.3)

COND= A > B .OR. X < 0.0 .AND. Y > 1.0

EXPRESSÕES E ATRIBUIÇÕES ESCALARES

- Expressões e atribuições de caracteres escalares
operador de concatenação “//”,

```
cvar='ABCD'='AB'//'CD'
```

- Substrings de caracteres

```
Cvar(2:3)='BC'
```

Exercicio016

```
PROGRAM LE_CARACTERE
```

```
IMPLICIT NONE
```

```
CHARACTER(LEN=6) :: STR1='RUA No'
```

```
CHARACTER(LEN=6) :: STR2=' 001'
```

```
PRINT *, "TEXTO 1:", STR1
```

```
PRINT *, "TEXTO 2:", STR2
```

```
PRINT *, "TEXTO CONCATENADO:", STR1//STR2
```

```
END PROGRAM LE_CARACTERE
```

EXPRESSÕES E ATRIBUIÇÕES ESCALARES

Resolução da Expressão

- As expressões são efetuadas da esquerda para a direita.
- Se um operador é encontrado no processo de resolução ,sua **prioridade** é comparada com o próximo operador.
- SE o próximo tem **prioridade mais baixa**, efetua o operador corrente com seus operandos;
- SE o próximo tem **prioridade igual** ao corrente, **as leis de associabilidade** são usadas para determinar o primeiro que deve ser efetuado;
- Se o próximo tem prioridade maior, a varredura continua;

EXPRESSÕES E ATRIBUIÇÕES ESCALARES

Resolução da Expressão

- Uma simples expressão aritmética é uma expressão onde todos os operadores são do mesmo tipo.
- Se os operadores são **INTEGERS** (*resp.*, **REALs**), o resultado é também um **INTEGER** (*resp.*, **REAL**).

```
1.0 + 2.0 * 3.0 / ( 6.0*6.0 + 5.0*44.0) ** 0.25
--> 1.0 + 6.0 / (6.0*6.0 + 5.0*44.0) ** 0.25
--> 1.0 + 6.0 / (36.0 + 5.0*44.0) ** 0.25
--> 1.0 + 6.0 / (36.0 + 220.0) ** 0.25
--> 1.0 + 6.0 / 256.0 ** 0.25
--> 1.0 + 6.0 / 4.0
--> 1.0 + 1.5
--> 2.5
```

EXPRESSÕES E ATRIBUIÇÕES ESCALARES

Resolução da Expressão

- Note que $a**b**c$ é $a**(b**c)$ em vez de $(a**b)**c$, e $a**(b**c) \neq (a**b)**c$.

Isto pode ser uma grande armadilha!

```
5 * (11.0 - 5) ** 2 / 4 + 9
--> 5 * (11.0 - 5.0) ** 2 / 4 + 9
--> 5 * 6.0 ** 2 / 4 + 9
--> 5 * 36.0 / 4 + 9
--> 5.0 * 36.0 / 4 + 9
--> 180.0 / 4 + 9
--> 180.0 / 4.0 + 9
--> 45.0 + 9
--> 45.0 + 9.0
--> 54.0
```

6.02 não é convertido para 6.0**2.0!**

EXPRESSÕES E ATRIBUIÇÕES ESCALARES

Resolução da Expressão

- A esquerda o exemplo usa uma Variável inicializada **Unit**,
- e a direita usa um parâmetro (**PARAMETER**) **PI**.

```
INTEGER :: Total, Amount
INTEGER :: Unit = 5
Amount = 100.99
Total = Unit * Amount
```

Isto é equivalente a
Radius ** 2 * PI

```
REAL, PARAMETER :: PI = 3.1415926
REAL :: Area
INTEGER :: Radius
Radius = 5
Area = (Radius ** 2) * PI
```


Comandos e Construtos de Controle de Fluxo

- 1 O comando **IF** fornece um mecanismo para controle de desvio de fluxo, dependendo de uma condição. Há duas formas: o comando **IF** e o construtor **IF**, sendo o último uma forma geral do primeiro.

IF (<expressão relacional e/ou lógica>) <comando executável>

O <comando executável> é qualquer um, exceto aqueles que marcam o início ou o final de um bloco, como por exemplo **IF**, **ELSE IF**, **ELSE**, **END IF**.

IF (FLAG) CYCLE

IF (X-Y > 0.0) X= 0.0

IF (COND .OR. P < Q . &A

ND. R <= 1.0) S(I,J)= T(J,I)

Exercicio017

```
PROGRAM TES_CMD_IF
  IMPLICIT NONE
  INTEGER :: I
  INTEGER, PARAMETER :: DP= 2
  REAL(KIND= DP) :: X
  I= KIND(X) ! I= 2
  IF(I /=2)STOP "I /=2 "
END PROGRAM TES_CMD_IF
```

Comandos e Construtos de Controle de Fluxo

- 1 Um **construto IF** permite que a execução de uma sequência de comandos (bloco) seja realizada, dependendo de uma condição ou de um outro bloco, dependendo de outra condição.

```
[<nome>:] IF (<expressão relacional e/ou lógica>) THEN  
    <bloco>  
END IF [<nome>]
```

```
[<nome>:] IF (<expressão relacional e/ou lógica>) THEN  
    <bloco 1>  
ELSE [<nome>]  
    <bloco 2>  
END IF [<nome>]
```

```
[<nome>:] IF (<expressão relacional e/ou lógica>) THEN  
    <bloco>  
[ELSE IF (<expressão relacional e/ou lógica>) THEN [<nome>]  
    <bloco>]  
...  
[ELSE [<nome>]  
    <bloco>]  
END IF [<nome>]
```

Exercicio018

```
PROGRAM IF_FAT  
!CALCULA O FATORIAL.  
IMPLICIT NONE  
INTEGER :: I, FAT, J  
PRINT *, "ENTRE COM VALOR:"  
READ *, I  
IF (I < 0) THEN  
    PRINT *, "ERRO."  
ELSE IF (I == 0) THEN  
    PRINT *, "FAT(",I,")=",1  
ELSE  
    FAT= 1  
    DO J= 1, I  
        IF(J<2)CYCLE  
        FAT= FAT*J  
    END DO  
    PRINT *, "FAT(",I,")=",FAT  
END IF  
END PROGRAM IF_FAT
```

Comandos e Construtos de Controle de Fluxo

- **1** Um laço **DO** é usado quando for necessário calcular uma série de operações semelhantes, dependendo ou não de algum parâmetro que é atualizado em cada início da série.

[<nome>:] **DO** [<variável> = <expressão1>, <expressão2> [, <expressão3>]]

<bloco>

END DO [<nome>]

<variável> : inteiro (índice da iteração);

<expressão 1>: o valor inicial da <variável>;

<expressão 2>: o valor máximo, ou limite, da <variável>;

<expressão 3>: incremento.

Exercicio019

```
PROGRAM MOD_PASSO
IMPLICIT NONE
INTEGER :: I,J
! BLOCO SEM MODIFICAÇÃO DE PASSO.
DO I= 1, 10
    PRINT *, I
END DO
! BLOCO COM MODIFICAÇÃO DO PASSO.
J= 1
DO I= 1, 10, J
    IF (I > 5)J= 3
    PRINT *, I,J
END DO
END PROGRAM MOD_PASSO
```

Comandos e Construtos de Controle de Fluxo

Exercicio020

```
PROGRAM main
IMPLICIT NONE
INTEGER, PARAMETER :: ncol=10
INTEGER, PARAMETER :: nlin=10
INTEGER, PARAMETER :: ntim=1
INTEGER, PARAMETER :: nvar=1
REAL, PARAMETER :: nlevs(15)=(/1000, 925, 850, 775, 700, 500, 400, 300, 250, 200, 150, 100, 70, 50, 30/)
CHARACTER(LEN=256) :: NAMEBIN='teste.bin'
INTEGER :: i
INTEGER :: unit=6
REAL :: undef=9.999E+20
REAL :: loni=-180.0
REAL :: dlon=0.0005
REAL :: lati=-60.0
REAL :: dlat=0.0005
CHARACTER(LEN=12) :: itime='06Z06JAN2003'
CHARACTER(LEN= 4 ) :: dtime='24hr'
CHARACTER(LEN= 5 ) :: vname
WRITE(unit, '(A,A)') 'dset ^',TRIM(NAMEBIN)
WRITE(unit, '(A,e10.4)') 'undef ',undef
WRITE(unit, '(A)') 'title curso de fortran'
WRITE(unit, '(A)') 'options '
WRITE(unit, '(A,I10,A,2F12.5)') 'xdef ', ncol, ' linear ',loni,dlon
WRITE(unit, '(A,I10,A,2F12.5)') 'ydef ', nlin, ' linear ',lati,dlat
WRITE(unit, '(A,I10,A,2A,A)') 'tdef ', ntim, ' linear ',itime,' ', dtime
WRITE(unit, '(A,I10,A)') 'zdef ', SIZE(nlevs), ' levels '
WRITE(unit, '(5F10.2)') (nlevs(i),i=1,SIZE(nlevs))
WRITE(unit, '(A,I5)') 'vars ',nvar

DO i=1,nvar
  WRITE(vname, '(a3,I2.2)') 'var',i

  WRITE(unit, '(A,I5,A,A,A)') vname,SIZE(nlevs),' 99 ',' variavel ',i
END DO

WRITE(unit,'(A)') 'endvars'
END PROGRAM main
```

Convertendo inteiro para caracter

Comandos e Construtos de Controle de Fluxo

- 1 Construtor **DO ilimitado** :Como caso particular de um construtor **DO**, a seguinte instrução é possível:.
- 2 Instrução **EXIT**: Esta instrução permite a saída, por exemplo, de um laço sem limite, mas o seu uso não está restrito a este caso particular, podendo ser usada em um construtor **DO** geral.
- 3 Instrução **CYCLE** : Transfere controle à declaração **END DO** do construtor correspondente sem ocorrer a execução dos comandos posteriores à instrução.

Exercicio021

```
[<nome>:] DO  
    EXIT [<nome>]  
    <bloco>  
    CYCLE [<nome>]  
END DO [<nome>]  
  
EXIT [<nome>]  
  
CYCLE [<nome>]
```

```
PROGRAM DO_CYCLE  
IMPLICIT NONE  
INTEGER :: INDEX= 1  
DO  
    INDEX= INDEX + 1  
    IF(INDEX == 20) CYCLE  
    IF(INDEX == 30) EXIT  
    PRINT *, "VALOR DO ÍNDICE:",INDEX  
END DO  
END PROGRAM DO_CYCLE
```

Comandos e Construtos de Controle de Fluxo

- **1** Construtor **CASE** :alternativa para selecionar uma de diversas opções:. A principal diferença entre este construtor e um bloco **IF** está no fato de somente uma expressão ser calculada para decidir o fluxo e esta pode ter uma série de resultados pré-definidos:

```
[<nome>:] SELECT CASE (<expressão>)  
[CASE (<seletor>) [<nome>]  
  <bloco>]  
[CASE DEFAULT  
  <bloco>]  
END SELECT [<nome>]
```

```
SELECT CASE (NUMERO)    ! NUMERO é do tipo inteiro.  
CASE (:-1)              ! Todos os valores de NUMERO menores que 0.  
  N_SINAL= -1  
CASE (0)                 ! Somente NUMERO= 0.  
  N_SINAL= 0  
CASE (1:)                ! Todos os valores de NUMERO > 0.  
  N_SINAL= 1  
END SELECT
```

```
SELECT CASE (CH)        ! CH é do tipo de caractere.  
CASE ('C', 'D', 'R:')   ! Valore reais.  
  CH_TYPE= .TRUE.  
CASE ('I': 'N')         ! Valore reais.  
  INT_TYPE= .TRUE.  
CASE DEFAULT           ! Todos os valores não incluso acima  
  REAL_TYPE= .TRUE.  
END SELECT
```

seletores de caso somente testam o primeiro caractere da variável character CH, mesmo ela tendo comprimento maior que 1.

Comandos e Construtos de Controle de Fluxo

Exercicio022

```
PROGRAM CASE_STRING
IMPLICIT NONE
CHARACTER(LEN= 5) :: NOME
PRINT *, "ENTRE COM O NOME (5 CARACTERES):"
READ "(A5)", NOME
SELECT CASE (NOME)
CASE ("a":"z")      ! SELECIONA NOME QUE COMEÇA COM LETRAS MINÚSCULAS.
    PRINT *, "PALAVRA INICIA COM LETRA MINÚSCULA."
CASE ("A":"Z")      ! SELECIONA NOME QUE COMEÇA COM LETRAS MAIÚSCULAS.
    PRINT *, "PALAVRA INICIA COM LETRAS MAIÚSCULA."
CASE ("0":"9")      ! SELECIONA NÚMEROS.
    PRINT *, "PALAVRA INICIA COM NÚMEROS!!!"
CASE DEFAULT        ! OUTROS TIPOS DE CARACTERES.
    PRINT *, "PALAVRA INICIA COM CARACTERES ESPECIAIS!!!"
END SELECT
END PROGRAM CASE_STRING
```

```
PROGRAM TESTA_CASE
IMPLICIT NONE
INTEGER :: A
PRINT *, "ENTRE COM A (INTEIRO):"
READ *, A
SELECT CASE (A)
CASE (:-1)
    PRINT *, "MENOR QUE ZERO."
CASE (0)
    PRINT *, "IGUAL A ZERO."
CASE (1:)
    PRINT *, "MAIOR QUE ZERO."
END SELECT
END PROGRAM TESTA_CASE
```

Exercicio023

seletores de caso somente testam o primeiro caractere da variável character CH, mesmo ela tendo comprimento maior que 1.

Processamento de Matrizes

Terminologia e especificações de matrizes :Uma matriz consiste de um conjunto retangular de elementos, todos do mesmo tipo e espécie do tipo Uma outra definição equivalente seria: uma matriz é um grupo de **posições continua na memória** do computador as quais são acessadas por intermédio de um único nome, fazendo-se uso dos subscritos da matriz. O Fortran 77/90/95 permite que uma matriz tenha até **sete** subscrito

“(/ ” e “ /) ” são os construtores de matrizes, definem ou inicializam os valores de um vetor ou matriz.

<tipo>, DIMENSION(<lista de extensões>) [, <outros atributos>] :: <lista de nomes>

INTEGER
REAL
LOGICAL
CHARACTER

PARAMETER
ALLOCATABLE
INTENT(INOUT)
OPTIONAL
SAVE
EXTERNAL
INTRINSIC
PUBLIC
PRIVATE
POINTER
TARGET

Processamento de Matrizes

1. Inicialização de vetores contendo 3 elementos:

INTEGER :: I

INTEGER, DIMENSION(3) :: IA= (/1,2,3/), IB= (/I, I=1,3)/)

2. Declaração da matriz automática LOGB. Aqui, LOGA é uma matriz qualquer (“muda” ou “dummy”) e **SIZE** é uma função intrínseca que retorna um escalar inteiro correspondente ao tamanho do seu argumento:

LOGICAL, DIMENSION(SIZE(LOGA)) :: LOGB

3. Declaração das matrizes dinâmicas, ou alocáveis, de duas dimensões A e B. A forma das matrizes será definida a posteriori por um comando **ALLOCATE**:

REAL, DIMENSION(:,:), ALLOCATABLE :: A,B

4. Declaração das matrizes de forma assumida de três dimensões A e B. A forma das matrizes será assumida a partir das informações transferidas pela rotina que aciona o sub-programa onde esta declaração é feita.

REAL, DIMENSION(:, :, :) :: A,B

Processamento de Matrizes

1. **Matrizes de tipos derivados.** A capacidade de se misturar matrizes com definições de tipos derivados possibilita a construção de objetos de complexidade crescente. Alguns exemplos ilustram isto.

```
TYPE :: TRIPLETO
  REAL :: U
  REAL, DIMENSION(3) :: DU
  REAL, DIMENSION(3,3) :: D2U
END TYPE TRIPLETO

TYPE(TRIPLETO) :: T

TYPE(TRIPLETO), DIMENSION(10) :: V
```

Exercicio024

```
PROGRAM ALUNOS_VET
IMPLICIT NONE
INTEGER :: I, NDISC= 5 !MUDE ESTE VALOR, CASO SEJA
MAIOR.
TYPE:: ALUNO
  CHARACTER(LEN= 20):: NOME
  INTEGER:: CODIGO
  REAL:: N1,N2,N3,MF
END TYPE ALUNO
TYPE(ALUNO), DIMENSION(5):: DISC
DO I= 1,NDISC
  PRINT*, "NOME:"
  READ "(A)", DISC(I)%NOME
  PRINT*, "CÓDIGO:"
  READ*, DISC(I)%CODIGO
  PRINT*, "NOTAS: N1,N2,N3:"
  READ*, DISC(I)%N1, DISC(I)%N2, DISC(I)%N3
  DISC(I)%MF= (DISC(I)%N1 + DISC(I)%N2 +
DISC(I)%N3)/3.0
END DO
DO I= 1,NDISC
  PRINT*, " "
  PRINT*, "> ", DISC(I)%NOME, " (", DISC(I)%CODIGO, ") <-"
  PRINT*, "      MÉDIA FINAL: ", DISC(I)%MF
END DO
END PROGRAM ALUNOS_VE
```

Expressões e atribuições com Matrizes

1. Expressões

Exercicio025

```

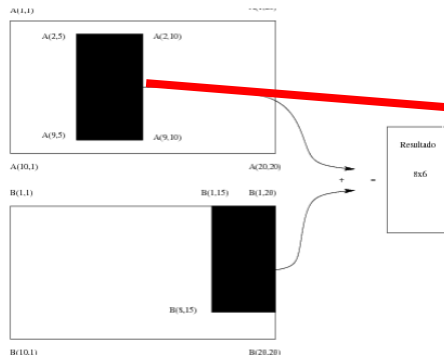
PROGRAM ALUNOS_VET
IMPLICIT NONE
REAL, DIMENSION(20) :: A=0.0,B,C
REAL, DIMENSION(5,5,5) :: AA
REAL :: VAL_MAX
REAL :: MEDIA

B=2.0;C=4.0
A= A/3.1 + B*SQRT(C)
VAL_MAX= MAXVAL(A, MASK=(A<1000.0))
MEDIA= SUM(AA,MASK=(AA>3000.0)) / &
        COUNT(MASK=(AA>3000.0))
END PROGRAM ALUNOS_VET
    
```

3 Atribuições de matrizes e sub-matrizes

```

REAL, DIMENSION(1:9,1:20) :: A, B, C
C= A(2:9,5:10) + B(1:8,15:20) !Forma (/8,6/).
    
```



2 Seções de matrizes

$$\begin{aligned}
 RA = \begin{bmatrix} 00000 \\ 0X000 \\ 00000 \\ 00000 \\ 00000 \end{bmatrix} &\Rightarrow RA(2,2) = X & RA = \begin{bmatrix} 00000 \\ 0X000 \\ 00000 \\ 00000 \\ 00000 \end{bmatrix} &\Rightarrow RA(2:2,2:2) = X; \\
 RA = \begin{bmatrix} 00000 \\ 00000 \\ 00X00 \\ 00000 \\ 00000 \end{bmatrix} &\Rightarrow RA(3,3:5); & RA = \begin{bmatrix} 0XXX0 \\ 00000 \\ 0XXX0 \\ 00000 \\ 0XXX0 \end{bmatrix} &\Rightarrow RA(1::2,2:4);
 \end{aligned}$$

Exercicio026

```

PROGRAM TESTA_ATR_MATR
IMPLICIT NONE
REAL, DIMENSION(3,3) :: A
REAL, DIMENSION(2,2) :: B
INTEGER :: I, J
DO I= 1,3
  DO J= 1,3
    A(I,J)= SIN(REAL(I)) + COS(REAL(J))
  END DO
END DO
B= A(1:2,1:3:2)
PRINT*, "MATRIZ A:"
PRINT"(3(F12.5))", ((A(I,J), J= 1,3), I= 1,3)
PRINT*, "MATRIZ B:"
PRINT"(2(F12.5))", ((B(I,J), J= 1,2), I= 1,2)
END PROGRAM TESTA_ATR_MATR
    
```

Exercicio027 Expressões e atribuições com Matrizes

```
PROGRAM MATC
IMPLICIT NONE
INTEGER, PARAMETER :: N=5
CHARACTER(LEN=1) :: RA(N,N)
INTEGER :: I,J
RA=RESHAPE(SOURCE=(&
'0','0','0','0','0',&
'0','0','0','0','0',&
'0','0','0','0','0',&
'0','0','0','0','0',&
'0','0','0','0','0'),SHAPE= (/5,5/))
PRINT*, 'RA(2:2,2:2)=X'; RA(2:2,2:2)='X'
DO I=1,N
DO J=1,N
WRITE (*,'(A,2X)',ADVANCE='NO ') RA(I,J)
END DO
PRINT*
END DO
RA='0'; PRINT*, 'RA(3,3:5)=X'; RA(3,3:5)='X'
DO I=1,N
DO J=1,N
WRITE (*,'(A,2X)',ADVANCE='NO ') RA(I,J)
END DO
PRINT*
END DO
RA='0'; PRINT*, 'RA(1::2,2:4)=X'; RA(1::2,2:4)='X'
DO I=1,N
DO J=1,N
WRITE (*,'(A,2X)',ADVANCE='NO ') RA(I,J)
END DO
PRINT*
END DO
END PROGRAM MATC
```

2 Seções de matrizes

$$RA = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & X & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \Rightarrow RA(2:2, 2:2) = X;$$

$$RA = \begin{bmatrix} 0 & X & X & X & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & X & X & X & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & X & X & X & 0 \end{bmatrix} \Rightarrow RA(1::2, 2:4);$$

$$RA = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & X & X & X \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \Rightarrow RA(3, 3:5);$$

$$RA = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & X & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \Rightarrow RA(2, 2) = X.$$

Expressões e atribuições com Matrizes

1. Matrizes de tamanho zero

```
PROGRAM ALUNOS_VET
IMPLICIT NONE
REAL, DIMENSION(5) :: A,B
REAL, DIMENSION(5,5) :: A
!Quando I assume o valor N, as sub-matrizes B(N+1:N)
!e A(N+1:N,N) se tornam nulas
DO I= 1, N
  X(I)= B(I)/A(I,I)
  B(I+1:N)= B(I+1:N) - A(I+1:N,I)*X(I)
END DO
END PROGRAM ALUNOS_VE
```

2 Construtores de matrizes

(/ <lista de valores do construtor> /)

V= (/ 1, 3, 5 /)

A= (/ I+J, 2*I, 2*J, I**2, J**2, &
SIN(REAL(I)), COS(REAL(J)) /)

(/ (i, i= 1,6) /) ! Resulta: (/ 1, 2, 3, 4, 5, 6 /)

(/ 7, (i, i= 1,4), 9 /) ! Resulta: (/ 7, 1, 2, 3, 4, 9 /)

(/ A(I,2:4), A(1:5:2,I+3) /)

! Resulta: (/ A(I,2), A(I,3), A(I,4), A(1,I+3), A(3,I+3),
A(5,I+3) /)

3 A função intrínseca RESHAPE.

RESHAPE(SOURCE= (/ 1, 2, 3, 4, 5, 6 /), SHAPE= (/ 2, 3 /))

REAL :: RA (3,2)

RA= RESHAPE(SOURCE= (/ ((I+J, I= 1,3), J= 1,2) /),&
SHAPE= (/ 3,2 /))

```
PROGRAM TESTA_ATR_MATR
IMPLICIT NONE
REAL, DIMENSION(3,3) :: A
REAL, DIMENSION(2,2) :: B
INTEGER :: I, J
!
A= RESHAPE(SOURCE= &
  (/((SIN(REAL(I))+COS(REAL(J))),&
    I= 1,3), J= 1,3)/), &
  SHAPE= (/3,3/))
B= A(1:2,1:3:2)
PRINT*, "MATRIZ A:"
PRINT"(3(F12.5))", ((A(I,J), J= 1,3), I= 1,3)
PRINT*, "MATRIZ B:"
PRINT"(2(F12.5))", ((B(I,J), J= 1,2), I= 1,2)
END PROGRAM TESTA_ATR_MATR
```

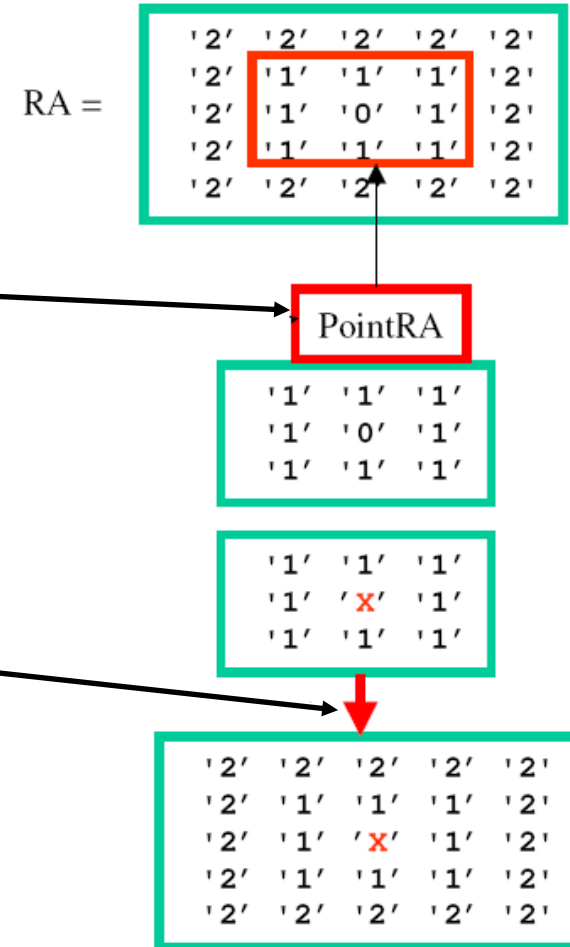
Exercicio028

Exercicio029

Expressões e atribuições com Matrizes

```
PROGRAM MatC2
IMPLICIT NONE
INTEGER, PARAMETER :: n=5
CHARACTER(LEN=1), TARGET :: RA(n,n)
CHARACTER(LEN=1), POINTER :: PointRA(:, :)
INTEGER :: i,j
RA=RESHAPE(SOURCE=(&
'2','2','2','2','2',&
'2','1','1','1','2',&
'2','1','0','1','2',&
'2','1','1','1','2',&
'2','2','2','2','2'),&
SHAPE= (/5,5/))
PointRA=>RA(2:4,2:4)
PRINT*, 'PointRA'
DO i=1, SIZE(PointRA,1)
DO j=1, SIZE(PointRA,2)
WRITE (*, '(A,2x)', advance='no ') PointRA(i,j)
END DO
PRINT*
END DO
PointRA(2,2)='X'
PRINT*, 'RA'
DO i=1,n
DO j=1,n
WRITE (*, '(A,2x)', advance='no ') RA(i,j)
END DO
PRINT*
END DO
END PROGRAM MatC2
```

2 Ponteiro



Expressões e atribuições com Matrizes

1. Matrizes alocáveis

Fortran 90 fornece tanto matrizes **alocáveis** quanto matrizes **automáticas**, ambos os tipos sendo **matrizes dinâmica**

Exercicio030

```
PROGRAM TESTA_ALOC
IMPLICIT NONE
INTEGER, PARAMETER :: nLin=10
INTEGER, PARAMETER :: nCols=10
REAL, DIMENSION(:,,:), ALLOCATABLE :: A
INTEGER :: STATUS
!ALLOCATE (<lista de objetos alocados> [, STAT= <status>])
IF (.NOT. ALLOCATED(A)) ALLOCATE (A(nLin, nCols) , &
STAT= STATUS)
IF (STATUS > 0) THEN
PRINT*, " Comandos de processamento de erro.";STOP
ELSE
A=5.0
PRINT*, SIZE(A),MAXVAL(A),MINVAL(A)
END IF
! Uso da matriz A
!DEALLOCATE (<lista de objetos alocados> [, STAT= <status>])
IF (ALLOCATED(A)) DEALLOCATE (A, STAT= STATUS)
END PROGRAM TESTA_ALOC
```

Exercicio031

```
PROGRAM TESTA_ALOC
IMPLICIT NONE
INTEGER, DIMENSION(:,,:), ALLOCATABLE :: B
INTEGER :: I,J,N= 2
!
PRINT*, "VALOR INICIAL DE N:",N
ALLOCATE (B(N,N))
B= N
PRINT*, "VALOR INICIAL DE B:"
PRINT"(2(I2))", ((B(I,J), J= 1,N), I= 1,N)
DEALLOCATE (B)
N= N + 1
PRINT*, "SEGUNDO VALOR DE N:",N
ALLOCATE (B(N,N))
B= N
PRINT*, "SEGUNDO VALOR DE B:"
PRINT"(3(I2))", ((B(I,J), J= 1,N), I= 1,N)
DEALLOCATE (B)
N= N + 1
PRINT*, "TERCEIRO VALOR DE N:",N
ALLOCATE (B(N+1,N+1))
B= N + 1
PRINT*, "TERCEIRO VALOR DE B:"
PRINT"(5(I2))", ((B(I,J), J= 1,N+1), I= 1,N+1)
END PROGRAM TESTA_ALOC
```

Expressões e atribuições com Matrizes

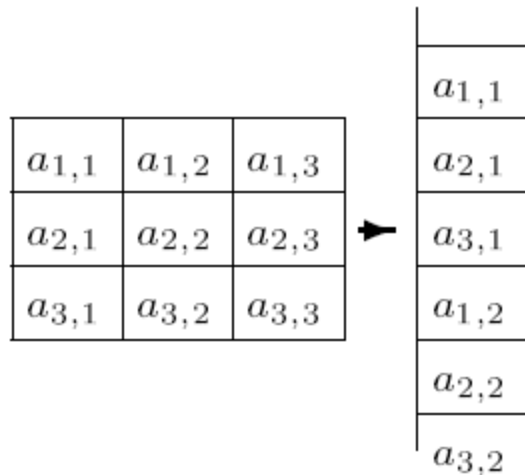
1. Matrizes alocáveis

Exercicio032

```
PROGRAM TESTA_ALOC
IMPLICIT NONE
INTEGER :: nLin=10
INTEGER :: nCols=10
REAL, DIMENSION(:,:), ALLOCATABLE :: A
INTEGER :: STATUS
PRINT*, " nLin e nCols"
READ(*,*)nLin,nCols
IALLOCATE (<lista de objetos alocados> [, STAT= <status>])
IF (.NOT. ALLOCATED(A)) ALLOCATE (A(nLin, nCols) , STAT= STATUS)
IF (STATUS > 0)THEN
PRINT*, " Comandos de processamento de erro.";STOP
ELSE
  A =5.0
  A(1,1)=0.0
PRINT*, SIZE(A),MAXVAL(A),MINVAL(A)
END IF
! Uso da matriz A
IDEALLOCATE (<lista de objetos alocados> [, STAT= <status>])
IF (ALLOCATED(A)) DEALLOCATE (A, STAT= STATUS)
END PROGRAM TESTA_ALOC
```


Expressões e atribuições com Matrizes

1. A ordem dos elementos de matrizes



3 Construto WHERE.

WHERE (<expressão lógica matriz>
<operações atribuições matrizes>
END WHERE

WHERE (<expressão lógica matriz>
<operações atribuições matrizes 1>
ELSEWHERE
<operações atribuições matrizes 2>
END WHERE

2 Comando e construtor WHERE

WHERE (<exp lógica matriz>) &
<var matriz>= <exp matriz>

REAL, DIMENSION(10,10) :: A
WHERE (A > 0.0) A= 1.0/A

Exercicio033

```
PROGRAM TESTA_WHERE
IMPLICIT NONE
REAL, DIMENSION(3,3) :: A
INTEGER :: I, J
A= RESHAPE(SOURCE=&
(/ ((SIN(REAL(I+J)), I= 1,3), J= 1,3) /), SHAPE= (/3,3/))
PRINT*, "MATRIZ A ORIGINAL:"
PRINT*, A(1,:)
PRINT*, A(2,:)
PRINT*, A(3,:)
WHERE (A >= 0.0)
  A= SQRT(A)
ELSEWHERE
  A= A**2
END WHERE
PRINT*, "MATRIZ A MODIFICADA:"
PRINT*, A(1,:)
PRINT*, A(2,:)
PRINT*, A(3,:)
END PROGRAM TESTA_WHERE
```

Exercicio034 **Expressões e atribuições com Matrizes**

```
PROGRAM TESTA_WHERE
```

```
  IMPLICIT NONE
```

```
  INTEGER :: nLin , nCol, STATUS
```

```
  REAL, ALLOCATABLE :: A (:,:)
```

```
  INTEGER :: I, J
```

```
  PRINT*, " nLin e nCol"
```

```
  READ(*,*)nLin,nCol
```

```
  !ALLOCATE (<lista de objetos alocados> [, STAT= <status>])
```

```
  IF (.NOT. ALLOCATED(A)) ALLOCATE (A(nLin, nCol) , STAT= STATUS)
```

```
  IF (STATUS > 0)THEN
```

```
    PRINT*, " Comandos de processamento de erro."; STOP
```

```
  ELSE
```

```
    A= RESHAPE(SOURCE= (/ ((SIN(REAL(I+J)), I= 1, nLin), J= 1, nCol ) /), SHAPE= (/ nLin, nCol /))
```

```
    PRINT*, SIZE(A),MAXVAL(A),MINVAL(A)
```

```
    PRINT*, "MATRIZ A ORIGINAL:"
```

```
    PRINT*, A(1,:)
```

```
    PRINT*, A(2,:)
```

```
    PRINT*, A(3,:)
```

```
    WHERE (A >= 0.0)
```

```
      A= SQRT(A)
```

```
    ELSEWHERE
```

```
      A= A**2
```

```
  END WHERE
```

```
  PRINT*, "MATRIZ A MODIFICADA:"
```

```
  PRINT*, A(1,:)
```

```
  PRINT*, A(2,:)
```

```
  PRINT*, A(3,:)
```

```
  ! Uso da matriz a
```

```
  !DEALLOCATE (<lista de objetos alocados> [, STAT= <status>])
```

```
  IF (ALLOCATED(A)) DEALLOCATE (A, STAT= STATUS)
```

```
  END IF
```

```
END PROGRAM TESTA_WHERE
```

Expressões e atribuições com Matrizes

Exercicio035

```
PROGRAM whereexample1
  IMPLICIT NONE
  INTEGER,PARAMETER :: rk=SELECTED_REAL_KIND(10,40),i1=1,i2=10
  REAL(kind=rk),DIMENSION(i1:i2) :: x
  LOGICAL :: mask(i1:i2)
  INTEGER :: n
  x=1.0
  mask=.false.
  mask(i1:i2:2)=.true.
  WHERE (mask)
    x=-1.0
  END WHERE
  PRINT '(50l6)',mask
  PRINT '(50f6.1)',x
  STOP
END PROGRAM whereexample1
```

```
progs> ifort whereexample1.f90
```

```
progs> ./a.out
```

T	F	T	F	T	F	T	F	T	F
-1.0	1.0	-1.0	1.0	-1.0	1.0	-1.0	1.0	-1.0	1.0

Expressões e atribuições com Matrizes

1 Sintaxe mais geral do construto FORALL:

```
[<nome>:] FORALL (<índice>= <menor>:<maior>[:<passo>], &  
    [, <índice>= <menor>:<maior>[:<passo>]] [...] &  
    [, <expressão lógica escalar>])  
    <corpo>  
END FORALL [<nome>]
```

2. Comando FORALL

Especifica que as atribuições individuais sejam realizadas em **qualquer** ordem, inclusive **simultaneamente**

```
FORALL (I= 1:N) A(I,I)= X(I)  
FORALL (I= 1:N, J= 1:M) A(I,J)= I + J  
FORALL (I= 1:N, J= 1:N, Y(I,J) /= 0.0) &  
X(J,I)= 1.0/Y(I,J)
```

2.Construtor FORALL

```
A(2:N-1, 2:N-1)= A(2:N-1, 1:N-2) + A(2:N-1, 3:N) &  
    + A(1:N-2, 2:N-1) + A(3:N, 2:N-1)  
B(2:N-1, 2:N-1)= A(2:N-1, 2:N-1)
```

construtor **FORALL**:

```
FORALL (I= 2:N-1, J= 2:N-1)  
    A(I,J)= A(I,J-1) + A(I, J+1) + A(I-1, J) + A(I+1, J)  
    B(I,J)= A(I,J)  
END FORALL  
construtor FORALL encadeados  
FORALL (I= 1:N-1)  
    FORALL (J= I+1:N)  
        A(I,J)= A(J,I)  
    END FORALL  
END FORALL  
construtor FORALL  
FORALL (I= 1:N)  
    WHERE (A(I,:) == 0) A(I,:)= I  
    B(I,:)= I/A(I,:)  
END FORALL
```

Expressões e atribuições com Matrizes

1 Exemplo1 FORALL:

Exercicio036

```
PROGRAM TESFORALL
IMPLICIT NONE
INTEGER, DIMENSION(30) :: A,B
INTEGER :: N
!
FORALL(N=1:30)
    A(N)= N
    B(N)= A(30-N+1)
END FORALL
PRINT*, 'VETOR A:'
PRINT*, A
PRINT*, "
PRINT*, 'VETOR B:'
PRINT*, B
END PROGRAM TESFORALL
```

2. Exemplo1 FORALL

Exercicio037

```
PROGRAM TESFORALL2
IMPLICIT NONE
INTEGER, DIMENSION(3,3) :: A
INTEGER :: I,J
A= RESHAPE(SOURCE= (/ (I, I= 1,9)/), SHAPE= (/3,3/))
PRINT*, "MATRIZ A:"
PRINT*, A(1,:)
PRINT*, A(2,:)
PRINT*, A(3,:)
FORALL (I= 1:3, J= 1:3) A(I,J)= A(J,I)
PRINT*, "TRANSPOSTA DE A:"
PRINT*, A(1,:)
PRINT*, A(2,:)
PRINT*, A(3,:)
END PROGRAM TESFORALL2
```

Expressões e atribuições com Matrizes

1 Exemplo FORALL:

Exercicio038

```
PROGRAM forallexample
IMPLICIT NONE
INTEGER,PARAMETER :: &
& rk=SELECTED_REAL_KIND(10,40)
INTEGER :: a(10,10)
INTEGER :: i,j
a=0
FORALL (i=1:10:2,j=1:10:2,i+j>10)
a(i,j)=(i+j)/2
END FORALL
PRINT '(10i4)',a
STOP
END PROGRAM forallexample
```

```
progs> ifort forallexample.f90
progs> a.out
0000000000
0000000000
0000000060
0000000000
0000006070
0000000000
0000607080
0000000000
0060708090
0000000000
```

Expressões e atribuições com Matrizes

Exercicio039

```
PROGRAM main
IMPLICIT NONE
INTEGER, PARAMETER :: ncol=5
INTEGER, PARAMETER :: nlin=5
INTEGER, PARAMETER :: ntim=1
INTEGER, PARAMETER :: nvar=1
CHARACTER(LEN=256) :: NAMEBIN='teste.bin'
CHARACTER(LEN=256) :: NAMECTL='teste.ctl'
INTEGER :: unit=10
REAL :: undef=9.999E+20
REAL :: loni=-180.0
REAL :: dlon=1.000
REAL :: lati=-60.0
REAL :: dlat=1.000
CHARACTER(LEN=12) :: itime='06Z06JAN2003'
CHARACTER(LEN=4 ) :: dtime='24hr'
CHARACTER(LEN=5) :: vname
INTEGER :: zlev(nvar)
REAL , PARAMETER :: nlevs(18)=(/1000, 925, 850, 775, 700, 500, 400,&
300, 250, 200, 150, 100, 70, 50,&
30, 20, 10, 3/)
REAL(KIND=4) :: RA(nlin,ncol)
INTEGER :: lrec
INTEGER :: j
INTEGER :: ios
INTEGER :: i,it,iv,iz,lrec
```

Continuação

```
RA=RESHAPE(SOURCE=(&  
2,2,2,2,2,&  
2,1,1,1,2,&  
2,1,0,1,2,&  
2,1,1,1,2,&  
2,2,2,2,2/),&  
SHAPE=(/nlin,ncol/))  
zlev=RESHAPE(SOURCE=(&1/),SHAPE=(/nvar/))
```

```
! WHERE(RA==1)  
! RA=0  
! ELSEWHERE(RA==0)  
! RA=1  
! END WHERE  
FORALL(I=2:nlin-1, J=2:ncol-1)  
RA(i,j)=5  
END FORALL
```

```
INQUIRE(IOLength=lrec)RA
```

```
OPEN(unit,FILE=TRIM(NAMEBIN),STATUS='UNKNOWN',FORM='UNFORMATTED',&  
ACCESS='DIRECT',ACTION='WRITE',RECL=lrec,iostat=ios)
```


Continuação

```
irec=0
DO it=1, ntim
DO iv=1,nvar
DO iz=1,zlev(iv)
irec=irec+1
WRITE(unit,rec=irec) RA
END DO
END DO
END DO
CLOSE(unit,STATUS='KEEP')
OPEN(unit,FILE=TRIM(NAMECTL),STATUS='UNKNOWN',FORM='FORMATTED' ,&
ACCESS='SEQUENTIAL',ACTION='WRITE',iostat=ios)
WRITE(unit,'(A,A )')dset '^',TRIM(NAMEBIN)
WRITE(unit,'(A,e10.4 )')undef ',undef
WRITE(unit,'(A )')title curso de fortran'
WRITE(unit,'(A )')options '
WRITE(unit,'(A,I10,A,2F12.5)')xdef ', ncol , ' linear ',loni,dlon
WRITE(unit,'(A,I10,A,2F12.5)')ydef ', nlin , ' linear ',lati,dlat
WRITE(unit,'(A,I10,A,2A,A )')tdef ', ntim , ' linear ',itime,' ', dtime
WRITE(unit,'(A,I10,A )')zdef ', SIZE(nlevs),' levels '
WRITE(unit,'(5F10.2 )')(nlevs(i),i=1,SIZE(nlevs))
WRITE(unit,'(A,I5 )')vars ',nvar
DO i=1,nvar
WRITE(vname,'(a3,I2.2 )')'var',i
WRITE(unit,'(A,I5,A,A,A )')vname,SIZE(nlevs),' 99 ', ' variavel ', '( )'
END DO
WRITE(unit,'(A )')endvars'
CLOSE(unit,STATUS='KEEP')
END PROGRAM main
```

Expressões e atribuições com Matrizes

Exercicio040

```
SUBROUTINE matriz_imprime ( a , n )  
IMPLICIT NONE  
INTEGER :: n , i , j  
REAL :: a ( n , n )  
DO i = 1 , n  
  DO j = 1 , n  
    WRITE ( *, '(f,2x)', advance='no ' ) a(i,j)  
  END DO  
  PRINT *  
END DO  
PRINT *  
END SUBROUTINE matriz_imprime
```

```
SUBROUTINE matriz_transposta (a,n)  
IMPLICIT NONE  
INTEGER :: n,i,j  
REAL :: a (n,n) , temp  
DO i = 1 , n  
  DO j = 1 , n  
    IF ( i < j ) THEN  
      temp = a (i,j)  
      a (i,j) = a (j,i)  
      a (j,i) = temp  
    END IF  
  END DO  
END DO  
CALL matriz_imprime (a,n)  
SUBROUTINE matriz_transposta
```

```
SUBROUTINE matriz_soma (a,b,c,n)  
IMPLICIT NONE  
INTEGER :: n,i,j  
REAL :: a(n,n),b(n,n),c(n,n)  
DO i = 1,n  
  DO j = 1,n  
    c ( i , j ) = a ( i , j ) + b ( i , j )  
  END DO  
END DO  
CALL matriz_imprime (c,n)  
END SUBROUTINE matriz_soma
```

```
SUBROUTINE matriz_mul (a,b,c,n)  
IMPLICIT NONE  
INTEGER :: n,i,j,k  
REAL :: a(n,n) , b(n,n) , c(n,n)  
c = 0.0  
DO i = 1 , n  
  DO j = 1 , n  
    DO k = 1 , n  
      c (i,j) = c (i,j) + a (i,k) * b (k,j)  
    END DO  
  END DO  
END DO  
CALL matriz_imprime (c,n)  
END SUBROUTINE matriz_mul
```

Expressões e atribuições com Matrizes

```
SUBROUTINE matriz_inversa (matriz,inversa,n)
IMPLICIT NONE
INTEGER :: n
REAL,DIMENSION(n,n) :: matriz
REAL,DIMENSION(n,n) :: inversa
LOGICAL :: invertivel=.TRUE.
INTEGER :: i,j,k,l
REAL :: m
REAL ,DIMENSION (n,2*n) :: matriz_umentada
! Matriz aumentada com uma matriz identidade
DO i =1 , n
DO j = 1 ,2*n
IF (j <= n ) THEN
matriz_umentada (i,j)=matriz(i,j)
ELSEIF ((i+n) == j) THEN
matriz_umentada(i,j) = 1
ELSE
matriz_umentada(i,j) = 0
ENDIF
END DO
END DO
! Reduzir a matriz aumentada a uma matriz triangular superior pela eliminacao gaussiana
DO k = 1 , n-1
! Verica se algum elemento da diagonal e zero
IF (ABS(matriz_umentada(k,k)) <= 1.0E-6) THEN
invertivel = .FALSE.
DO i = k+1,n
! Verica se os elementos sao maiores que zero
IF ( ABS ( matriz_umentada (i,k)) > 1.0E-6) THEN
DO j = 1,2*n
matriz_umentada(k,j) = matriz_umentada(k,j) + matriz_umentada(i,j)
END DO
invertivel=.TRUE.
EXIT
ENDIF
! Se algum elemento da diagonal for zero, nao podemos calcular a inversa
```

Expressões e atribuições com Matrizes

```
! Se algum elemento da diagonal for zero, nao podemos calcular a inversa
IF (invertivel==.FALSE.) THEN
WRITE (*, '(/,x,A,/)' ) ** A matriz nao e inverstivel! ** '
inversa = 0
STOP
END IF
END DO
END IF
! Eliminacao gaussiana
DO j =k+1,n
m=matriz_aumentada(j,k)/matriz_aumentada(k,k)
DO i = k,2*n
matriz_aumentada(j,i) = matriz_aumentada(j,i) - m*matriz_aumentada(k,i)
END DO
END DO
END DO
! Teste para invertibilidade
DO i = 1,n
! Elementos da diagonal nao podem ser zero
IF(ABS(matriz_aumentada(i,i))<=1.0E-6) THEN
WRITE(*, '(/,x,A,/)' ) ** A matriz nao e inverstivel ! ** '
inversa = 0
STOP
END IF
END DO
! Elementos da diagonal iguais a 1
DO i = 1 , n
m =matriz_aumentada(i,i)
DO j = i,2*n
matriz_aumentada(i,j)=matriz_aumentada(i,j)/m
END DO
END DO
! Reduzir o lado esquerdo da matriz aumentada a matriz identidade
```

Expressões e atribuições com Matrizes

! Reduzir o lado esquerdo da matriz aumentada a matriz identidade

DO k = n-1,1,-1

DO i = 1,k

m=matriz_aumentada(i,k+1)

DO j = k,2*n

matriz_aumentada(i,j)=matriz_aumentada(i,j)-matriz_aumentada(k+1,j)*m

END DO

END DO

END DO

! Armazene o resultado

DO i =1,n

DO j =1,n

inversa(i,j) = matriz_aumentada(i,j+n)

END DO

END DO

CALL matriz_imprime (inversa,n)

END SUBROUTINE matriz_inversa

Expressões e atribuições com Matrizes

Exemplo

Exercicio040

Funções Internas : F90

<i>Function</i>	<i>Meaning</i>	<i>Arg. Type</i>	<i>Return Type</i>
ABS (x)	absolute value of x	INTEGER	INTEGER
		REAL	REAL
SQRT (x)	square root of x	REAL	REAL
SIN (x)	sine of x radian	REAL	REAL
COS (x)	cosine of x radian	REAL	REAL
TAN (x)	tangent of x radian	REAL	REAL
ASIN(x)	arc sine of x	REAL	REAL
ACOS (x)	arc cosine of x	REAL	REAL
ATAN(x)	arc tangent of x	REAL	REAL
EXP (x)	$\exp(x)$	REAL	REAL
LOG (x)	natural logarithm of x	REAL	REAL

LOG10 (x) is the common logarithm of x!

Funções Internas : F90

<i>Function</i>	<i>Meaning</i>	<i>Arg. Type</i>	<i>Return Type</i>
INT (x)	truncate to integer part x	REAL	INTEGER
NINT (x)	round nearest integer to x	REAL	INTEGER
FLOOR (x)	greatest integer less than or equal to x	REAL	INTEGER
FRACTION (x)	the fractional part of x	REAL	REAL
REAL (x)	convert x to REAL	INTEGER	REAL

Examples:

INT(-3.5) = -3

NINT(3.5) = 4

NINT(-3.4) = -3

FLOOR(3.6)= 3

FLOOR(-3.5)= -4

FRACTION(12.3)= 0.3

REAL(-10) = -10.0

Funções Internas : F90

<i>Function</i>	<i>Meaning</i>	<i>Arg. Type</i>	<i>Return Type</i>
MAX(<i>x</i> 1, <i>x</i> 2, ..., <i>x</i> <i>n</i>)	maximum of <i>x</i> 1, <i>x</i> 2, ... <i>x</i> <i>n</i>	INTEGER	INTEGER
		REAL	REAL
MIN(<i>x</i> 1, <i>x</i> 2, ..., <i>x</i> <i>n</i>)	minimum of <i>x</i> 1, <i>x</i> 2, ... <i>x</i> <i>n</i>	INTEGER	INTEGER
		REAL	REAL
MOD(<i>x</i> , <i>y</i>)	remainder <i>x</i> - INT(<i>x</i> / <i>y</i>)* <i>y</i>	INTEGER	INTEGER
		REAL	REAL

Examples:

MAX(1,2,3,4,5,6,7,8)=8

MIN(12,3,4,5,6,7,8,9,-3.5)=-3.5

MOD(2004/4)=0

Funções Internas : F90

- As Funções tem a mais alta prioridade.
- Os argumentos da função são resolvidos primeiro.
- O valor retornado pela função é tratado como um valor na expressão.

```
REAL :: A = 1.0, B = -5.0, C = 6.0, R
R = (-B + SQRT(B*B - 4.0*A*C))/(2.0*A)
(-B + SQRT(B*B - 4.0*A*C))/(2.0*A)
--> (5.0 + SQRT(B*B - 4.0*A*C))/(2.0*A)
--> (5.0 + SQRT(25.0 - 4.0*A*C))/(2.0*A)
--> (5.0 + SQRT(25.0 - 4.0*C))/(2.0*A)
--> (5.0 + SQRT(25.0 - 24.0))/(2.0*A)
--> (5.0 + SQRT(1.0))/(2.0*A)
--> (5.0 + 1.0)/(2.0*A)
--> 6.0/(2.0*A)
--> 6.0/2.0
--> 3.0
```

R recebe **3.0**

SubProgramas e Modulos

- Fortran 90/95, tem três unidades distintas de programa:

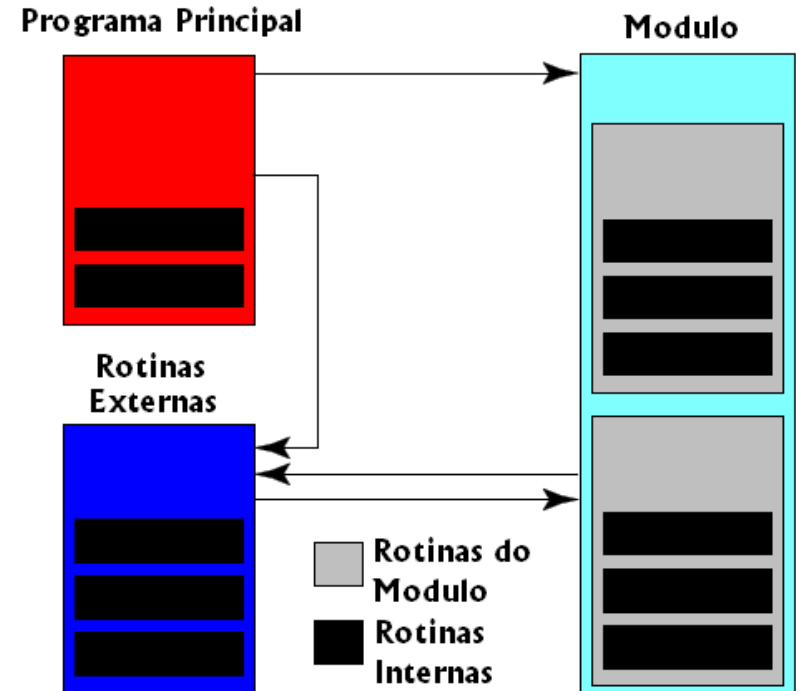
- Programa principal.

```
[PROGRAM <nome do programa>
  [<declarações de variáveis>]
  [<comandos executáveis>]
[CONTAINS
  <sub-programas internos>
END [PROGRAM [<nome do programa>]]
```

- Rotinas externas.

```
[SUBROUTINE <nome do subprograma>
  [<declarações de variáveis>]
  [<comandos executáveis>]
CONTAINS
  <sub-programas internos>
END [SUBROUTINE [<nome do subprograma>]]
```

```
[FUNCTION <nome função>
  [<declarações de variáveis>]
  [<comandos executáveis>]
[CONTAINS
  <sub-programas internos>
END [FUNCTION [<nome função>]]
```



- Módulos.

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

```

PROGRAM Main
  IMPLICIT NONE
  INTEGER :: teste
  REAL    :: A
  a=1.0
  a=soma(a)
  print*, "a=", a
CONTAINS
  REAL FUNCTION Soma(var) result(var2)
    IMPLICIT NONE
    REAL , INTENT(IN  ) :: var
    var2=var
    var2=var2+1
  END FUNCTION Soma
END PROGRAM Main

```

```

PROGRAM Main
  IMPLICIT NONE
  INTEGER :: teste
  REAL    :: A(3)
  a=0.0
  a=soma(a)
  print*, "a=", a
CONTAINS
  FUNCTION Soma(var) result(var2)
    IMPLICIT NONE
    REAL , INTENT(IN  ) :: var(:)
    REAL :: var2(size(var))
    INTEGER :: i
    var2=var
    DO i=1,size(var)
      var2(i)=var2(i)+i
    END DO
  END FUNCTION Soma
END PROGRAM Main

```

```
PROGRAM Main
  IMPLICIT NONE
  INTEGER :: teste
  REAL    :: A(3)
  a=0.0
  a=soma(a)
  print*, "a=", a
CONTAINS
  FUNCTION Soma(var) result(var2)
    IMPLICIT NONE
    REAL , INTENT(IN ) :: var(:)
    REAL :: var2(size(var))
    INTEGER :: i
    var2=var
    DO i=1,size(var)
      var2(i)=var2(i)+i
    END DO
  END FUNCTION Soma
END PROGRAM Main
```

SubProgramas e Modulos

• Rotinas externas.

```
[PURE] [ELEMENTAL] [RECURSIVE] SUBROUTINE <nome subrotina> [(<lista argumentos mudos>)]  
  [<declarações de variáveis>]  
  [<comandos executáveis>]  
[CONTAINS  
  <sub-programas internos>  
END [SUBROUTINE [<nome subrotina>]]
```

ou pode ser uma função:

```
[PURE] [ELEMENTAL] [<tipo>] [RECURSIVE] FUNCTION <nome função> &  
  (<lista argumentos mudos>) [RESULT (<nome resultado>)]  
  [<declarações de variáveis>]  
  [<comandos executáveis>]  
[CONTAINS  
  <sub-programas internos>  
END [FUNCTION [<nome função>]]
```

2-Rotinas internas

INTENT ([**IN**][**OUT**][**INOUT**])
OPTIONAL
SAVE
EXTERNAL
INTRINSIC
POINTER
TARGET

```
[RECURSIVE] SUBROUTINE <nome subrotina> [(<lista argumentos mudos>)]  
  [<declarações de variáveis>]  
  [<comandos executáveis>]  
END SUBROUTINE [<nome subrotina>]]  
ou  
[<tipo>] [RECURSIVE] FUNCTION <nome função> (<lista argumentos mudos>) &  
  [RESULT (<nome resultado>)]  
  [<declarações de variáveis>]  
  [<comandos executáveis>]  
END FUNCTION [<nome função>]]
```

SubProgramas e Modulos

•Rotinas internas. Exercicio041

```
PROGRAM ROT_INT
```

```
  IMPLICIT NONE
```

```
  REAL :: X,A
```

```
INTERFACE
```

```
  FUNCTION CALC_A_POW(Y)
```

```
    REAL, INTENT(IN) :: Y
```

```
    REAL              :: CALC_A_POW
```

```
  END FUNCTION CALC_A_POW
```

```
END INTERFACE
```

```
PRINT*, "ENTRE COM O VALOR DE X:"
```

```
READ*, X
```

```
PRINT*, "ENTRE COM O VALOR DE A:"
```

```
READ*, A
```

```
PRINT*, "O RESULTADO DE A + SQRT(X) É:"
```

```
PRINT*, CALC_A_RAIZ(X)
```

```
PRINT*, "O RESULTADO DE A + SQRT(1+ X**2) É:"
```

```
PRINT*, CALC_A_RAIZ(1.0+ X**2)
```

```
STOP
```

```
CONTAINS
```

```
  FUNCTION CALC_A_RAIZ(Y)
```

```
    IMPLICIT NONE
```

```
    REAL, INTENT(IN) :: Y
```

```
    REAL              :: CALC_A_RAIZ
```

```
    CALC_A_RAIZ= A + SQRT(Y)
```

```
    Y= CALC_A_POW(Y)
```

```
    RETURN
```

```
  END FUNCTION CALC_A_RAIZ
```

```
END PROGRAM ROT_INT
```

Comando **RETURN**

Atributo **INTENT**
([IN][OUT][INOUT])

Comando **STOP**

```
FUNCTION CALC_A_POW(Y)
  IMPLICIT NONE
  REAL, INTENT(IN) :: Y
  REAL              :: CALC_A_POW
  CALC_A_RAIZ= (Y**2)
  RETURN
END FUNCTION CALC_A_POW
```

SubProgramas e Modulos

•Rotinas internas.

Exercicio042

Um array de um procedure pode ser de comprimento assumido ou forma assumida.

- neste caso a interface do procedure deve ser escrito explicitamente.

```
PROGRAM assumedtest
  IMPLICIT NONE
  INTERFACE
    SUBROUTINE sub(x,y)
      IMPLICIT NONE
      REAL :: x(:),y(:, :)
    END SUBROUTINE sub
  END INTERFACE
  REAL :: x(3),y(3,3)
  x=1.0
  y=2.0
  CALL sub(x,y)
  PRINT *,x
  PRINT *,y
  STOP
END PROGRAM assumedtest

SUBROUTINE sub(x,y)
  IMPLICIT NONE
  REAL :: x(:),y(:, :)
  PRINT *,SIZE(x),SIZE(y)
  x=2.0*x
  y=3.0*y
  RETURN
END SUBROUTINE sub
```

```
progs> ifort assumedtest.f90
progs> a.out
3 9
2.000000 2.000000 2.000000
6.000000 6.000000 6.000000 6.000000 6.000000
6.000000 6.000000 6.000000 6.000000
```

Opinião pessoal: A forma mais limpa e eficiente para passar o array para a subrotina é passar também o seu tamanho; for example:

```
call sub(x,y,3)
...
subroutine sub(x,y,n)
  implicit none
  integer :: n
  real :: x(n),y(n,n)
  print *,size(x),size(y)
  x=2.0*x
  y=3.0*y
  return
end subroutine sub
```


Exercicio043

```
PROGRAM ROT_INT
IMPLICIT NONE
REAL :: X,A
INTERFACE
FUNCTION CALC_A_POW(Y)
REAL, INTENT(IN) :: Y
REAL :: CALC_A_POW
END FUNCTION CALC_A_POW
END INTERFACE
PRINT*, "ENTRE COM O VALOR DE X:"
READ*, X
PRINT*, "ENTRE COM O VALOR DE A:"
READ*, A
PRINT*, "O RESULTADO DE A + SQRT(X) É:"
PRINT*, CALC_A_RAIZ(X)
PRINT*, "O RESULTADO DE A + SQRT(1+ X**2) É:"
PRINT*, CALC_A_RAIZ(1.0 + X**2)
STOP
CONTAINS
FUNCTION CALC_A_RAIZ(Y)
IMPLICIT NONE
REAL, INTENT(IN) :: Y
REAL :: CALC_A_RAIZ
CALC_A_RAIZ= A + SQRT(Y)
Y=CALC_A_POW(Y)
RETURN
END FUNCTION CALC_A_RAIZ
END PROGRAM ROT_INT
```

```
FUNCTION CALC_A_POW(Y)
IMPLICIT NONE
REAL, INTENT(IN) :: Y
REAL :: CALC_A_POW
CALC_A_RAIZ= (Y**2)
RETURN
END FUNCTION CALC_A_POW
```

SubProgramas e Modulos

- Rotinas internas externas.

- Para gerar chamadas a sub-programas de forma correta, o compilador necessita conhecer certos detalhes destes sub-programas, como os seus nomes e o número, tipo e espécie dos argumentos. Em Fortran 90/95, esta tarefa é desempenhada pelos blocos de interface.

- No caso de rotinas intrínsecas, sub-programas internos e módulos, esta informação é sempre conhecida pelo compilador; por isso, diz-se que as interfaces são explícitas.

- Contudo, quando o compilador chama uma rotina externa, esta informação não é conhecida de antemão e daí a interface é dita implícita.

Exercicio044 SubProgramas e Modulos

```
FUNCTION TESTA_DISC(C2,C1,C0)
IMPLICIT NONE
LOGICAL :: TESTA_DISC
! VARIÁVEIS MUDAS:
REAL, INTENT(IN) :: C0, C1, C2
IF(C1*C1 - 4*C0*C2 >= 0.0)THEN
TESTA_DISC= .TRUE.
ELSE
TESTA_DISC= .FALSE.
END IF
RETURN
END FUNCTION TESTA_DISC
```

```
! CALCULA AS RAÍZES REAIS, CASO EXISTAM,
! DE UM POLINÔNIO DE GRAU 2.
! USA FUNÇÃO TESTA_DISC.
!
SUBROUTINE BASCARA(A2,A1,A0,RAIZES_REAIS,R1,R2)
IMPLICIT NONE
! VARIÁVEIS MUDAS:
REAL, INTENT(IN) :: A0, A1, A2
LOGICAL, INTENT(OUT) :: RAIZES_REAIS
REAL, INTENT(OUT) :: R1, R2
! VARIÁVEIS LOCAIS:
REAL :: DISC
INTERFACE
FUNCTION TESTA_DISC(C2,C1,C0)
LOGICAL :: TESTA_DISC
REAL, INTENT(IN) :: C0, C1, C2
END FUNCTION TESTA_DISC
END INTERFACE
!
RAIZES_REAIS= TESTA_DISC(A2,A1,A0)
IF(.NOT. RAIZES_REAIS)RETURN
DISC= A1*A1 - 4*A0*A2
R1= 0.5*(-A1 - SQRT(DISC))/A2
R2= 0.5*(-A1 + SQRT(DISC))/A2
RETURN
END SUBROUTINE BASCARA
```

```
! CALCULA AS RAÍZES REAIS DE UM POLINÔMIO DE GRAU 2.
PROGRAM BASCARA3
IMPLICIT NONE
LOGICAL :: CONTROLE
REAL :: A, B, C
REAL :: X1, X2 ! RAÍZES REAIS.
INTERFACE
SUBROUTINE BASCARA(A2,A1,A0,RAIZES_REAIS,R1,R2)
REAL, INTENT(IN) :: A0, A1, A2
LOGICAL, INTENT(OUT) :: RAIZES_REAIS
REAL, INTENT(OUT) :: R1, R2
END SUBROUTINE BASCARA
END INTERFACE
!
DO
PRINT*, "ENTRE COM OS VALORES DOS COEFICIENTES (A,B,C),"
PRINT*, "ONDE A*X**2 + B*X + C."
READ*, A,B,C
CALL BASCARA(A,B,C,CONTROLE,X1,X2)
IF(CONTROLE)THEN
PRINT*, "AS RAÍZES (REAIS) SÃO:"
PRINT*, "X1=",X1
PRINT*, "X2=",X2
EXIT
ELSE
PRINT*, "AS RAÍZES SÃO COMPLEXAS. TENTE NOVAMENTE."
END IF
END DO
END PROGRAM BASCARA3
```

Exercicio045 SubProgramas e Modulos

```
SUBROUTINE SUB(RA, RB, RC, MAX_A)
IMPLICIT NONE
REAL, DIMENSION(:, :), INTENT(IN) :: RA, RB
REAL, DIMENSION(:, :), INTENT(OUT) :: RC
REAL, INTENT(OUT) :: MAX_A
MAX_A = MAXVAL(RA)
RC = 0.5*RB
RETURN
END SUBROUTINE SUB
```

```
PROGRAM PROG_SUB
IMPLICIT NONE
REAL, DIMENSION(0:9,10) :: A
REAL, DIMENSION(5,5) :: C
REAL :: VALOR_MAX
INTEGER :: I
INTERFACE
SUBROUTINE SUB(RA, RB, RC, MAX_A)
REAL, DIMENSION(:, :), INTENT(IN) :: RA, RB
REAL, DIMENSION(:, :), INTENT(OUT) :: RC
REAL, INTENT(OUT) :: MAX_A
END SUBROUTINE SUB
END INTERFACE
I
A = RESHAPE(SOURCE=((COS(REAL(I))), I= 1,100)/), &
SHAPE= (/10,10/)
CALL SUB(A, A(0:4,:5), C, VALOR_MAX)
PRINT*, "A MATRIZ A:"
DO I= 0, 9
PRINT*, A(I,:)
END DO
PRINT*, ""
PRINT*, "O MAIOR VALOR EM A:", VALOR_MAX
PRINT*, ""
PRINT*, "A MATRIZ C:"
DO I= 1, 5
PRINT*, C(I,:)
END DO
END PROGRAM PROG_SUB
```

O espaço na memória da CPU alocado em tempo real de processamento

Matrizes de forma assumida

REAL, DIMENSION(0:,:), INTENT(INOUT) :: DA

Matrizes de tamanho assumido

REAL TABELA(*)

Matrizes ajustáveis

REAL X(NPTS), Y(NPTS), Z(NPTS)

Argumentos opcionais

REAL, INTENT(IN), OPTIONAL :: INICIO, FINAL, TOL

SubProgramas e Modulos

O espaço na memória da CPU alocado em tempo real de processamento

Objetos automático

REAL, DIMENSION(:), INTENT(INOUT) :: A, B

REAL, DIMENSION(SIZE(A)) :: TEMP

!A função SIZE fornece o tamanho de TEMP.

CHARACTER(LEN= *) :: PALAVRA1

CHARACTER(LEN= LEN(PALAVRA1)) :: PALAVRA2

Exercicio046

```
PROGRAM LOREN
IMPLICIT NONE
CHARACTER(LEN= *), PARAMETER :: A= "SÓ UM PEQUENO EXEMPLO."
PRINT*, DOBRO(A)

CONTAINS


FUNCTION DOBRO(A)
CHARACTER(LEN= *), INTENT(IN) :: A
CHARACTER(LEN= 2*LEN(A)+2) :: DOBRO
DOBRO= A // " " // A
RETURN
END FUNCTION DOBRO
END PROGRAM LOREN
```

SubProgramas e Modulos

Exemplo usa matrizes alocáveis, de forma assumida e automáticas:

Exercicio047

```
SUBROUTINE SUB_MAT(A,RES)  
IMPLICIT NONE  
!MATRIZ DE FORMA ASSUMIDA  
REAL, DIMENSION(:,,:), INTENT(IN) :: A  
REAL, INTENT(OUT) :: RES  
!MATRIZ AUTOMÁTICA  
REAL, DIMENSION(SIZE(A,1),SIZE(A,2)) :: TEMP  
!  
TEMP= SIN(A)  
RES= MINVAL(A+TEMP)  
RETURN  
END SUBROUTINE SUB_MAT
```



SubProgramas e Modulos

Exemplo usa matrizes alocáveis, de forma assumida e automáticas:

```
PROGRAM MATRIZ_AUT
IMPLICIT NONE
REAL, DIMENSION(:,:), ALLOCATABLE :: A
REAL :: RES
INTEGER :: N, M, I
INTERFACE
SUBROUTINE SUB_MAT(A,RES)
REAL, DIMENSION(:,:), INTENT(IN) :: A
REAL, INTENT(OUT) :: RES
REAL, DIMENSION(SIZE(A,1),SIZE(A,2)) :: TEMP
END SUBROUTINE SUB_MAT
END INTERFACE
PRINT*, "ENTRE COM DIMENSÕES DA MATRIZ:"
READ*, N,M
ALLOCATE(A(N,M))
A= RESHAPE(SOURCE=/(TAN(REAL(I)), I= 1,N*M)/, &
SHAPE= (/N,M/))
PRINT*, "MATRIZ A:"
DO I= 1, N
PRINT*, A(I,:)
END DO
CALL SUB_MAT(A,RES)
PRINT*, ""
PRINT*, "O MENOR VALOR DE A + SIN(A) É:",RES
END PROGRAM MATRIZ_AUT
```

SubProgramas e Modulos

Sub-programas como argumentos de rotinas:

Exercicio048

```
! FUNÇÃO SEN(EXP(X)).  
FUNCTION MEU_F(X)  
REAL :: MEU_F  
REAL, INTENT(IN) :: X  
!  
MEU_F= SIN(EXP(X))  
RETURN  
END FUNCTION MEU_F
```

```
! Parâmetros:  
! f: Função externa a ser plotada.  
! xi: Ponto inicial (entrada).  
! xf: Ponto final (entrada).  
! npt: Número de pontos a ser gerados (entrada)  
! plot: matriz de forma (/ npt, 2 /) contendo  
! as abscissas e ordenadas dos pontos (saida).  
!  
SUBROUTINE PLOTA(F, XI, XF, NPT, PLOT)  
IMPLICIT NONE  
INTEGER, INTENT(IN) :: NPT  
REAL, INTENT(IN) :: XI, XF  
REAL, DIMENSION(2,NPT), INTENT(OUT) :: PLOT  
INTERFACE  
FUNCTION F(X)  
REAL :: F  
REAL, INTENT(IN) :: X  
END FUNCTION F  
END INTERFACE  
! VARIÁVEIS LOCAIS:  
INTEGER :: I  
REAL :: PASSO, X  
!  
PASSO= (XF - XI)/REAL(NPT - 1)  
X= XI  
DO I= 1, NPT  
PLOT(1,I)= X  
PLOT(2,I)= F(X)  
X= X + PASSO  
END DO  
RETURN  
END SUBROUTINE PLOTA
```


SubProgramas e Modulos

Sub-programas como argumentos de rotinas:

**! USA SUBROTINA PLOTA E CHAMA A FUNÇÃO
IEXTERNA MEU_F.**

PROGRAM TES_PLOTA

IMPLICIT NONE

INTEGER :: PTS, J

REAL :: YI, YF

REAL, DIMENSION(:, :), ALLOCATABLE :: XY

INTERFACE

SUBROUTINE PLOTA(F, XI, XF, NPT, PLOT)

INTEGER, INTENT(IN) :: NPT

REAL, INTENT(IN) :: XI, XF

REAL, DIMENSION(2,NPT), INTENT(OUT) :: PLOT

INTERFACE

FUNCTION F(X)

REAL :: F

REAL, INTENT(IN) :: X

END FUNCTION F

END INTERFACE

END SUBROUTINE PLOTA

FUNCTION MEU_F(Y)

REAL :: MEU_F

REAL, INTENT(IN) :: Y

END FUNCTION MEU_F

END INTERFACE

PRINT*, "ENTRE COM O NÚMERO DE PONTOS:"

READ*, PTS

PRINT*, "ENTRE COM OS LIMITES INFERIOR E SUPERIOR:"

READ*, YI, YF

ALLOCATE(XY(2,PTS))

CALL PLOTA(MEU_F, YI, YF, PTS, XY)

DO J= 1, PTS

PRINT*, XY(:,J)

END DO

END PROGRAM TES_PLOTA

SubProgramas e Modulos

Sub-programas como argumentos de rotinas:

Recursividade e rotinas recursivas

Recursividade ocorre quando rotinas chamam a si mesma, seja de forma direta ou indireta :

Recursividade direta:

A invoca A diretamente.

Recursividade indireta:

A invoca B, a qual invoca A.

Exercicio049

```
PROGRAM Mayne  
  IMPLICIT NONE  
  PRINT*, fact(3) ! etc  
CONTAINS  
  RECURSIVE FUNCTION fact(N) RESULT(N_Fact)  
    INTEGER, INTENT(IN) :: N  
    INTEGER :: N_Fact ! also defines type of fact  
    IF (N > 0) THEN  
      N_Fact = N * fact(N-1)  
    ELSE  
      N_Fact = 1  
    END IF  
  END FUNCTION fact  
END PROGRAM Mayne
```

SubProgramas e Modulos

Sub-programas como argumentos de rotinas:

Exercicio049

```
PROGRAM Mayne
  IMPLICIT NONE
  PRINT*, fact(3) ! etc
CONTAINS
  RECURSIVE FUNCTION fact(N) RESULT(N_Fact)
    INTEGER, INTENT(IN) :: N
    INTEGER :: N_Fact ! also defines type of fact
    IF (N > 0) THEN
      N_Fact = N * fact(N-1)
    ELSE
      N_Fact = 1
    END IF
  END FUNCTION FACT
END PROGRAM Mayne
```

SubProgramas e Modulos

Funções de efeito lateral e rotinas puras

funções que alteram o valor de seus argumentos são denominadas funções de efeito lateral (side-effect functions).

$RES = FUN1(A,B,C) + FUN2(A,B,C)$

```
FUNCTION FUN1(A,B,C)
INTEGER :: FUN1
REAL, INTENT(INOUT) :: A
REAL, INTENT(IN)    :: B,C
A = A * A
FUN1 = A/B
RETURN
END FUNCTION FUN1
```

```
FUNCTION FUN2(A,B,C)
INTEGER :: FUN2
REAL, INTENT(INOUT) :: A
REAL, INTENT(IN)    :: B,C
A = 2 * A
FUN2 = A/C
RETURN
END FUNCTION FUN2
```

SubProgramas e Modulos

Rotinas elementais

rotinas intrínsecas **elementais**, as quais são rotinas com argumentos mudos escalares que podem ser **invocadas** com argumentos reais **matriciais**, desde que os argumentos matriciais tenham todos a **mesma forma** (isto é, que sejam conformáveis). Para uma função, a forma do resultado é a forma dos argumentos matriciais.

```
TYPE :: INTERVALO  
  REAL :: INF, SUP  
END TYPE INTERVALO
```

!pode-se definir a seguinte função elemental:

```
ELEMENTAL FUNCTION SOMA_INTERVALOS(A,B)  
  INTRINSIC NONE  
  TYPE(INTEGER) :: SOMA_INTERVALOS  
  TYPE(INTEGER), INTENT(IN) :: A, B  
  SOMA_INTERVALOS%INF= A%INF + B%INF  
  SOMA_INTERVALOS%SUP= A%SUP + B%SUP  
  RETURN  
END FUNCTION SOMA_INTERVALOS
```

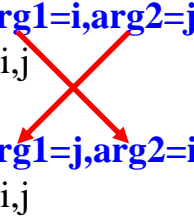
SubProgramas e Modulos

Associação de argumentos pode ser **positional** (normal) ou por **keywords**:

Exercicio050

```
program argassoc
  implicit none
  integer :: i,j
  i=1;j=2
  call sub(i,j) ! Positional
  write(0,*) i,j
  i=1;j=2
  call sub(arg1=i,arg2=j) ! por keyword
  write(0,*) i,j
  i=1;j=2
  call sub(arg1=j,arg2=i) !por keyword
  write(0,*) i,j
  stop
end program argassoc

subroutine sub(arg1,arg2)
  implicit none
  integer,intent(inout) :: arg1,arg2
  arg1=arg1*arg2
  arg2=-arg1
  return
end subroutine sub
```



```
progs> gfortran argassoc.f90
progs> a.out
 2  -2
 2  -2
-2   2
```

SubProgramas e Modulos

Exercicio051

```
!=====
PURE SUBROUTINE bar(func, a, b, s)
IMPLICIT NONE
REAL, INTENT(IN) :: a, b
REAL, INTENT(INOUT) :: s
INTERFACE
ELEMENTAL FUNCTION func(x)
REAL :: func
REAL, INTENT(IN) :: x
END FUNCTION
END INTERFACE
s = sum(func( (/ a, b /) ))
END SUBROUTINE bar
!=====
ELEMENTAL FUNCTION baz(x)
IMPLICIT NONE
REAL :: baz
REAL, INTENT(IN) :: x
baz = x
END FUNCTION baz
```

```
!=====
PURE FUNCTION foo(func, a, b)
IMPLICIT NONE
REAL :: foo
REAL, INTENT(IN) :: a, b
INTERFACE
ELEMENTAL FUNCTION func(x)
REAL :: func
REAL, INTENT(IN) :: x
END FUNCTION
PURE SUBROUTINE bar(func, a, b, s)
EXTERNAL func
REAL, INTENT(IN) :: a, b
REAL, INTENT(INOUT) :: s
END SUBROUTINE bar
END INTERFACE
CALL bar(func, a, b, foo)
END FUNCTION foo
```

```
PROGRAM main
IMPLICIT NONE
REAL :: a, b
REAL :: foo
REAL, EXTERNAL :: baz
a = 1.
b = 1.
WRITE(*,*) foo(baz, a, b)
END PROGRAM main
```

Exemplo subroutines

```
!=====
PURE SUBROUTINE bar(func, a, b, s)
  IMPLICIT NONE
  REAL, INTENT(IN) :: a, b
  REAL, INTENT(INOUT) :: s
  INTERFACE
    ELEMENTAL FUNCTION func(x)
      REAL :: func
      REAL, INTENT(IN) :: x
    END FUNCTION
  END INTERFACE

  s = sum(func( (/ a, b /) ))

END SUBROUTINE bar
!=====
ELEMENTAL FUNCTION baz(x)
  IMPLICIT NONE
  REAL :: baz
  REAL, INTENT(IN) :: x
  baz = x
END FUNCTION baz
```


Continua

!=====

PURE FUNCTION foo(func, a, b)

IMPLICIT NONE

REAL :: foo

REAL, INTENT(IN) :: a, b

INTERFACE

ELEMENTAL FUNCTION func(x)

REAL :: func

REAL, INTENT(IN) :: x

END FUNCTION

PURE SUBROUTINE bar(func, a, b, s)

EXTERNAL func

REAL, INTENT(IN) :: a, b

REAL, INTENT(INOUT) :: s

END SUBROUTINE bar

END INTERFACE

CALL bar(func, a, b, foo)

END FUNCTION foo

Continua

```
PROGRAM main  
  IMPLICIT NONE  
  REAL :: a, b  
  REAL :: foo  
  REAL, EXTERNAL :: baz  
  a = 1.  
  b = 1.  
  WRITE(*,*) foo(baz, a, b)  
END PROGRAM main
```

Exemplo 2

Exercicio052

Exemplo

ftp://ftp1.cptec.inpe.br/pkubota/curso_fortran/SubroutineGrads.f90

Continua

```
PROGRAM main
IMPLICIT NONE
TYPE :: Grads
  INTEGER :: ncol=-1
  INTEGER :: nlin=-1
  INTEGER :: ntim=-1
  INTEGER :: nvar=-1
  REAL :: undef=-9.999E+20
  REAL :: loni=-1.0
  REAL :: dlon=-1.0
  REAL :: lati=-1.0
  REAL :: dlat=-1.0
  CHARACTER(LEN=12) :: itime='06Z06JAN2003'
  CHARACTER(LEN=4 ) :: dtime='24hr'
  REAL :: nlevs(18)=(/1000, 925, 850, 775, 700, 500, 400,&
                        300, 250, 200, 150, 100, 70, 50,&
                        30, 20, 10, 3/)

  REAL(KIND=4), POINTER :: RA(:, :, :)
END TYPE Grads

INTEGER, PARAMETER :: ncol=5
INTEGER, PARAMETER :: nlin=5
INTEGER, PARAMETER :: ntim=2
INTEGER, PARAMETER :: nvar=1
CHARACTER(LEN=256) :: NAMEBIN='teste.bin'
CHARACTER(LEN=256) :: NAMECTL='teste.ctl'
INTEGER :: unit=10
```

Continua

```
REAL :: undef=9.999E+20
REAL :: loni=0.0
REAL :: dlon=1.5
REAL :: lati=-60.0
REAL :: dlat=1.5
CHARACTER(LEN=12) :: itime='06Z06JAN2003'
CHARACTER(LEN=4 ) :: dtime='24hr'
CHARACTER(LEN=5) :: vname
INTEGER :: nzlev(nvar)
REAL , PARAMETER :: nlevs(18)=(/1000, 925, 850, 775, 700, 500, 400,&
300, 250, 200, 150, 100, 70, 50,&
30, 20, 10, 3/)
REAL(KIND=4) :: RA(nlin,ncol)
INTEGER :: lrec
INTEGER :: j
INTEGER :: ios
INTEGER :: i,it,iv,iz,irec
TYPE(Grads), ALLOCATABLE :: var(:,:)
RA=RESHAPE(SOURCE=(/ &
2.,2.,2.,2.,2.,&
2.,1.,1.,1.,2.,&
2.,1.,0.,1.,2.,&
2.,1.,1.,1.,2.,&
2.,2.,2.,2.,2./),&
SHAPE=(/nlin,ncol/))
nzlev=RESHAPE(SOURCE=(/1/),SHAPE=(/nvar/))
```

Continua

```
ALLOCATE(var(nvar,ntim))  
DO it=1, ntim  
  DO iv=1,nvar  
    ALLOCATE(var(iv,it)%RA(nlin,ncol,nzlev(iv)))  
  END DO  
END DO  
  ! WHERE(RA==1)  
  ! RA=0  
  ! ELSEWHERE(RA==0)  
  ! RA=1  
  ! END WHERE  
FORALL(I=2:nlin-1, J=2:ncol-1)  
  RA(i,j)=5.0  
END FORALL
```

Continua

```
DO it=1, SIZE(var,2)
  DO iv=1,SIZE(var,1)
    var(iv,it)%ncol = ncol
    var(iv,it)%nlin = nlin
    var(iv,it)%ntim = ntim
    var(iv,it)%nvar = nvar
    var(iv,it)%undef = undef
    var(iv,it)%loni = loni
    var(iv,it)%dlon = dlon
    var(iv,it)%lati = lati
    var(iv,it)%dlat = dlat
    var(iv,it)%itime = itime
    var(iv,it)%dtime = dtime
    PRINT*,it
    DO iz=1,SIZE(var(iv,it)%RA,3)
      FORALL(I=1:nlin, J=1:ncol)
        var(iv,it)%RA(i,j,iz)=RA(i,j) + it*RA(i,j)
      END FORALL
    END DO
  END DO
END DO

CALL GeraBin(NAMEBIN,var)
CALL GeraCtl(NAMEBIN,var)

CONTAINS
```

Continua

```
SUBROUTINE GeraBin(NAMEBIN,var)
IMPLICIT NONE
CHARACTER(LEN=*) , INTENT(IN ) :: NAMEBIN
TYPE(Grads) , INTENT(IN ) :: var(:, :)
INTEGER :: lrec
INTEGER :: ios
INTEGER :: unit=1
INTEGER :: irec
INTEGER :: it
INTEGER :: iv
INTEGER :: iz , i, j
REAL(KIND=4) :: RA(nlin,ncol)
INQUIRE(IOLENGTH=lrec)var(1,1)%RA(:, :, 1)

OPEN(unit,FILE=TRIM(NAMEBIN),STATUS='UNKNOWN',FORM='UNFORMATTED',&
ACCESS='DIRECT' ,ACTION='WRITE',RECL=lrec,iostat=ios)

lrec=0
DO it=1, SIZE(var,2)
    DO iv=1,SIZE(var,1)
        DO iz=1,SIZE(var(iv,it)%RA,3)
            lrec=lrec+1
            WRITE(unit,rec=lrec) var(iv,it)%RA(:, :, iz)
        END DO
    END DO
END DO

CLOSE(unit,STATUS='KEEP')
END SUBROUTINE GeraBin
```


Continua

```
SUBROUTINE GeraCtl(NAMEBIN,var)
CHARACTER(LEN=*) , INTENT(IN ) :: NAMEBIN
TYPE(GRADS) , INTENT(IN ) :: var(:, :)
INTEGER :: ios
CHARACTER(LEN=256) :: NAMECTL
INTEGER :: unit=1
CHARACTER(LEN=15) :: vname
INTEGER :: iv
NAMECTL=NAMEBIN(1:LEN(TRIM(NAMEBIN))-4)//'.ctl'
OPEN(unit,FILE=TRIM(NAMECTL),STATUS='UNKNOWN',FORM='FORMATTED' ,&
ACCESS='SEQUENTIAL',ACTION='WRITE',iostat=ios)
WRITE(unit,'(A,A )')'dset ^',TRIM(NAMEBIN)
WRITE(unit,'(A )')'options '
WRITE(unit,'(A,e10.4 )')'undef ',var(1,1)%undef
WRITE(unit,'(A )')'title curso de fortran'
WRITE(unit,'(A,I10,A,2F12.5)')'xdef ', var(1,1)%ncol , ' linear ',var(1,1)%loni,var(1,1)%dlon
WRITE(unit,'(A,I10,A,2F12.5)')'ydef ', var(1,1)%nlin , ' linear ',var(1,1)%lati,var(1,1)%dlat
WRITE(unit,'(A,I10,A,2A,A )')'tdef ', var(1,1)%ntim , ' linear ',var(1,1)%itime,' ', var(1,1)%dtime
WRITE(unit,'(A,I10,A,F10.2 )')'zdef ', SIZE(var(1,1)%nlevs),' levels ', var(1,1)%nlevs(1)
WRITE(unit,'(5F10.2 )')(var(1,1)%nlevs(i),i=2,SIZE(var(1,1)%nlevs))
WRITE(unit,'(A,I5 )')'vars ',var(1,1)%nvar
DO iv=1,var(1,1)%nvar
WRITE(vname,'(a3,I2.2 )')'var',iv
WRITE(unit,'(A,I5,A,A,A )')TRIM(vname),SIZE(var(iv,1)%RA,3),' 99 ', ' variavel ', '(' )
vname=' '
END DO
WRITE(unit,'(A )')'endvars'
CLOSE(unit,STATUS='KEEP')
END SUBROUTINE GeraCtl
END PROGRAM main
```

Modulos

Módulos

Agrupar dados globais, tipos derivados e suas operações associadas, blocos de interface, grupos NAMELIST e rotinas internas

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

Declaração de objetos globais

Objetos tenham o seu valor compartilhado entre diferentes unidades de programa.

```
MODULE <nome módulo>  
IMPLICIT NONE  
INTEGER :: I, J, K  
END MODULE <nome módulo>  
!-----  
PROGRAM <nome programa>  
  USE <nome módulo>  
  IMPLICIT NONE  
END PROGRAM <nome programa>
```

```
MODULE modu1  
  implicit none  
  real, parameter :: pi= 3.1415926536  
  real, parameter :: euler_e= 2.718281828  
END MODULE modu1
```

Exercicio053

```
program mod1  
use modu1  
implicit none  
real :: x  
do  
  print*, "Entre com x:"  
  read*, x  
  print*, "sen(pi*x)=", sen()  
  print*, "ln(e*x)=", ln()  
end do  
CONTAINS  
  function sen()  
    real :: sen  
    sen= sin(pi*x)  
    return  
  end function sen  
!  
  function ln()  
    real :: ln  
    ln= log(euler_e*x)  
    return  
  end function ln  
end program mod1
```

Modulos

```
MODULE MODU1  
IMPLICIT NONE  
REAL, PARAMETER :: PI= 3.1415926536  
REAL, PARAMETER :: EULER_E= 2.718281828  
END MODULE MODU1
```

```
PROGRAM MOD1  
USE MODU1  
IMPLICIT NONE  
REAL :: X  
DO  
    PRINT*, "ENTRE COM X:"  
    READ*, X  
    PRINT*, "SEN(PI*X)=", SEN()  
    PRINT*, "LN(E*X)=", LN()  
END DO  
CONTAINS  
FUNCTION SEN()  
    REAL :: SEN  
    SEN= SIN(PI*X)  
    RETURN  
END FUNCTION SEN  
!  
FUNCTION LN()  
    REAL :: LN  
    LN= LOG(EULER_E*X)  
    RETURN  
END FUNCTION LN  
END PROGRAM MOD1
```

Modulos

Rotinas de módulos

Rotinas(subrotinas e funções) definidas no módulos são denominadas rotinas de módulos

outra grande diferença está no fato de que as interfaces das rotinas de módulo são explícitas também para a unidade de programa que o invoca.

```
MODULE MODU2
IMPLICIT NONE
REAL, PARAMETER :: PI= 3.1415926536
REAL, PARAMETER :: EULER_E= 2.718281828
REAL :: X

CONTAINS

FUNCTION SEN()
  REAL :: SEN
  SEN= SIN(PI*X)
  RETURN
END FUNCTION SEN
!
FUNCTION LN()
  REAL :: LN
  LN= LOG(EULER_E*X)
  RETURN
END FUNCTION LN
END MODULE MODU2
```

```
PROGRAM MOD2
USE MODU2
IMPLICIT NONE
DO
  PRINT*, "ENTRE COM O VALOR DE X:"
  READ*, X
  PRINT*, "SEN(PI*X)=", SEN()
  PRINT*, "LN(E*X)=", LN()
END DO
END PROGRAM MOD2
```

Exercicio054

Modulos

```
MODULE MODU2
IMPLICIT NONE
REAL, PARAMETER :: PI= 3.1415926536
REAL, PARAMETER :: EULER_E= 2.718281828
REAL :: X
CONTAINS
FUNCTION SEN()
REAL :: SEN
SEN= SIN(PI*X)
RETURN
END FUNCTION SEN
!
FUNCTION LN()
REAL :: LN
LN= LOG(EULER_E*X)
RETURN
END FUNCTION LN
END MODULE MODU2
```

```
PROGRAM MOD2
USE MODU2
IMPLICIT NONE
DO
PRINT*, "ENTRE COM O VALOR DE X:"
READ*, X
PRINT*, "SEN(PI*X)=", SEN()
PRINT*, "LN(E*X)=", LN()
END DO
END PROGRAM MOD2
```

Modulos

Atributos e declarações PUBLIC e PRIVATE

Força o usuário a **invocar** as rotinas de módulo que realmente deveriam ser **acessíveis** a este modulo. O **controle** é exercido através dos atributos ou declarações **PUBLIC** ou **PRIVATE**

```
PUBLIC [[:<lista acesso>]  
PRIVATE [[:<lista acesso>]
```

```
MODULE <nome módulo>  
IMPLICIT NONE  
PRIVATE  
INTEGER :: A, B, C  
INTEGER, PUBLIC :: I, J, K  
END MODULE <nome módulo>  
PROGRAM <nome programa>  
  USE <nome módulo>  
  IMPLICIT NONE  
END PROGRAM <nome programa>
```

Exercicio055

```
MODULE MODU2  
IMPLICIT NONE  
PRIVATE  
REAL, PARAMETER :: PI= 3.1415926536  
REAL, PARAMETER :: EULER_E= 2.71  
REAL, PUBLIC :: X  
CONTAINS  
  FUNCTION SEN()  
    REAL :: SEN  
    SEN= SIN(PI*X)  
    RETURN  
  END FUNCTION SEN  
!  
  FUNCTION LN()  
    REAL :: LN  
    LN= LOG(EULER_E*X)  
    RETURN  
  END FUNCTION LN  
END MODULE MODU2
```

```
PROGRAM MOD2  
USE MODU2  
IMPLICIT NONE  
DO  
  PRINT*, "ENTRE X:"  
  READ*, X  
  PRINT*, "SEN(PI*X)=", SEN()  
  PRINT*, "LN(E*X)=", LN()  
END DO  
END PROGRAM MOD2
```

Exercicio055

Modulos

```
MODULE MODU2
IMPLICIT NONE
PRIVATE
REAL, PARAMETER :: PI= 3.1415926536
REAL, PARAMETER :: EULER_E= 2.71
REAL, PUBLIC :: X
CONTAINS
FUNCTION SEN()
REAL :: SEN
SEN= SIN(PI*X)
RETURN
END FUNCTION SEN
!
FUNCTION LN()
REAL :: LN
LN= LOG(EULER_E*X)
RETURN
END FUNCTION LN
END MODULE MODU2
```

```
PROGRAM MOD2
USE MODU2
IMPLICIT NONE
DO
PRINT*, "ENTRE X:"
READ*, X
PRINT*, "SEN(PI*X)=", SEN()
PRINT*, "LN(E*X)=", LN()
END DO
END PROGRAM MOD2
```

Modulos

Interfaces e rotinas genéricas

É a habilidade do programador **definir** suas **próprias rotinas genéricas**, de tal forma que um único nome é suficiente para **invocar** uma determinada rotina, enquanto que a ação que realmente é executada quando este nome é usado **depende** do tipo de seus **argumentos**

```
INTERFACE <nome genérico>  
  <bloco interface rotina específica 1>  
  <bloco interface rotina específica 2>  
  ...  
END INTERFACE <nome genérico>
```

```
INTERFACE <nome genérico>  
  [<blocos interfaces>]  
  [MODULE PROCEDURE <lista nomes rotinas>]  
! Em Fortran 95 blocos de interfaces e declarações  
! MODULE PROCEDURE  
! podem aparecer em qualquer ordem.  
END INTERFACE [<nome genérico>]
```

Uma declaração **MODULE PROCEDURE** somente é permitida se um **<nome genérico>** é fornecido.

Interfaces e rotinas genéricas

```
MODULE GENTROCA  
IMPLICIT NONE
```


```
TYPE :: PONTO  
  REAL :: X,Y  
END TYPE PONTO
```

```
INTERFACE TROCA  
  MODULE PROCEDURE TROCA_PONTO, TROCA_REAL,&  
                    TROCA_INT, TROCA_LOG  
END INTERFACE TROCA
```

```
CONTAINS
```

```
ELEMENTAL SUBROUTINE TROCA_PONTO(A,B)  
  TYPE(PONTO), INTENT(INOUT) :: A, B  
  TYPE(PONTO) :: TEMP  
  TEMP= A  
  A= B  
  B= TEMP  
END SUBROUTINE TROCA_PONTO
```

```
ELEMENTAL SUBROUTINE TROCA_REAL(A,B)  
  REAL, INTENT(INOUT) :: A, B  
  REAL :: TEMP  
  TEMP= A  
  A= B  
  B= TEMP  
END SUBROUTINE TROCA_REAL
```



```
ELEMENTAL SUBROUTINE TROCA_INT(A,B)  
  INTEGER, INTENT(INOUT) :: A, B  
  INTEGER :: TEMP  
  TEMP= A  
  A= B  
  B= TEMP  
END SUBROUTINE TROCA_INT
```

```
ELEMENTAL SUBROUTINE TROCA_LOG(A,B)  
  LOGICAL, INTENT(INOUT) :: A, B  
  LOGICAL :: TEMP  
  TEMP= A  
  A= B  
  B= TEMP  
END SUBROUTINE TROCA_LOG  
END MODULE GENTROCA
```


```
PROGRAM USA_GENTROCA  
USE GENTROCA  
TYPE(PONTO) :: B= PONTO(1.0,0.0),  
TYPE(PONTO) :: C= PONTO(5.0,6.0)  
PRINT*, 'VALORES ORIGINAIS:'  
PRINT*, B,C  
CALL TROCA(B,C)  
PRINT*, 'NOVOS VALORES:'  
PRINT*, B,C  
END PROGRAM USA_GENTROCA
```

Modulos

```
MODULE GENTROCA
IMPLICIT NONE
TYPE :: PONTO
REAL :: X,Y
END TYPE PONTO
INTERFACE TROCA
MODULE PROCEDURE TROCA_PONTO, TROCA_REAL,&
TROCA_INT, TROCA_LOG
END INTERFACE TROCA
CONTAINS
ELEMENTAL SUBROUTINE TROCA_PONTO(A,B)
TYPE(PONTO), INTENT(INOUT) :: A, B
TYPE(PONTO) :: TEMP
TEMP= A
A= B
B= TEMP
END SUBROUTINE TROCA_PONTO
ELEMENTAL SUBROUTINE TROCA_REAL(A,B)
REAL, INTENT(INOUT) :: A, B
REAL :: TEMP
TEMP= A
A= B
B= TEMP
END SUBROUTINE TROCA_REAL
```



Continua



```
ELEMENTAL SUBROUTINE TROCA_INT(A,B)
  INTEGER, INTENT(INOUT) :: A, B
  INTEGER :: TEMP
  TEMP= A
  A= B
  B= TEMP
END SUBROUTINE TROCA_INT
```

```
ELEMENTAL SUBROUTINE TROCA_LOG(A,B)
  LOGICAL, INTENT(INOUT) :: A, B
  LOGICAL :: TEMP
  TEMP= A
  A= B
  B= TEMP
END SUBROUTINE TROCA_LOG
END MODULE GENTROCA
```

```
PROGRAM USA_GENTROCA
USE GENTROCA
TYPE(PONTO) :: B= PONTO(1.0,0.0)
TYPE(PONTO) :: C= PONTO(5.0,6.0)
PRINT*, 'VALORES ORIGINAIS:'
PRINT*, B,C
CALL TROCA(B,C)
PRINT*, 'NOVOS VALORES:'
PRINT*, B,C
END PROGRAM USA_GENTROCA
```

Modulos

Sobrecarga de operadores

ftp://ftp1.cptec.inpe.br/pkubota/curso_fortran/Sobrecarga_Operadores.f90

Exercicio057

```
INTERFACE OPERATOR(oper)
```

```
[definitions]
```

```
END INTERFACE
```

```
MODULE plus
```

```
  IMPLICIT NONE
```

```
  INTERFACE operator(+)
```

```
    MODULE PROCEDURE addst
```

```
  END INTERFACE
```

```
CONTAINS
```

```
  FUNCTION addst(string1,string2) RESULT(result)
```

```
    IMPLICIT NONE
```

```
    CHARACTER (LEN=*), INTENT(in):: string1, string2
```

```
    CHARACTER (LEN=LEN(string1) + LEN(string2)):: result
```

```
    result=string1//string2
```

```
  END FUNCTION addst
```

```
END MODULE plus
```

```
PROGRAM test
```

```
  USE plus
```

```
  IMPLICIT NONE
```

```
  PRINT *, 'Hip ' + 'hei!'
```

```
END PROGRAM test
```

Modulos

ftp://ftp1.cptec.inpe.br/pkubota/curso_fortran/Sobrecarga_Operadores_polar.f90

Exercicio058

```
MODULE polar_mod
  IMPLICIT NONE
  COMPLEX,PARAMETER::i = (0.0,1.0)
  REAL,PARAMETER::pi = 3.14159265359
  REAL,PARAMETER::e = 2.718282!exp(1.0)
  TYPE::polar_t
    REAL::l,th
  END TYPE polar_t
  INTERFACE OPERATOR(+)
    MODULE PROCEDURE add_pp
  END INTERFACE
  INTERFACE OPERATOR(-)
    MODULE PROCEDURE sub_pp
  END INTERFACE
  INTERFACE OPERATOR(*)
    MODULE PROCEDURE mul_pp
  END INTERFACE
  INTERFACE OPERATOR(/)
    MODULE PROCEDURE div_pp
  END INTERFACE
CONTAINS
elemental FUNCTION add_pp(u,v) RESULT(o)
  TYPE(polar_t),INTENT(in)::u,v
  TYPE(polar_t)::o
  COMPLEX::a,b,c
  a = u%l*EXP(i*u%th*pi)
  b = v%l*EXP(i*v%th*pi)
  c = a+b
  o%l = ABS(c)
  o%th = ATAN2(imag(c),REAL(c))/pi
END FUNCTION add_pp
```

```
elemental FUNCTION sub_pp(u,v) RESULT(o)
  TYPE(polar_t),INTENT(in)::u,v
  TYPE(polar_t)::o
  COMPLEX::a,b,c
  a = u%l*EXP(i*u%th*pi)
  b = v%l*EXP(i*v%th*pi)
  c = a-b
  o%l = ABS(c)
  o%th = ATAN2(imag(c),REAL(c))/pi
END FUNCTION sub_pp
```

```
elemental FUNCTION mul_pp(u,v) RESULT(o)
  TYPE(polar_t),INTENT(in)::u,v
  TYPE(polar_t)::o
  COMPLEX::a,b,c
  a = u%l*EXP(i*u%th*pi)
  b = v%l*EXP(i*v%th*pi)
  c = a*b
  o%l = ABS(c)
  o%th = ATAN2(imag(c),REAL(c))/pi
END FUNCTION mul_pp
```

```
elemental FUNCTION div_pp(u,v) RESULT(o)
  TYPE(polar_t),INTENT(in)::u,v
  TYPE(polar_t)::o
  COMPLEX::a,b,c
  a = u%l*EXP(i*u%th*pi)
  b = v%l*EXP(i*v%th*pi)
  c = a/b
  o%l = ABS(c)
  o%th = ATAN2(imag(c),REAL(c))/pi
END FUNCTION div_pp
END MODULE polar_mod
```

Modulos

Exercicio058

```
MODULE polar_mod
  IMPLICIT NONE
  COMPLEX,PARAMETER::i = (0.0,1.0)
  REAL,PARAMETER::pi = 3.14159265359
  REAL,PARAMETER::e = 2.718282!exp(1.0)
  TYPE::polar_t
    REAL::l,th
  END TYPE polar_t
  INTERFACE OPERATOR(+)
    MODULE PROCEDURE add_pp
  END INTERFACE
  INTERFACE OPERATOR(-)
    MODULE PROCEDURE sub_pp
  END INTERFACE
  INTERFACE OPERATOR(*)
    MODULE PROCEDURE mul_pp
  END INTERFACE
  INTERFACE OPERATOR(/)
    MODULE PROCEDURE div_pp
  END INTERFACE
  CONTAINS
  elemental FUNCTION add_pp(u,v) RESULT(o)
    TYPE(polar_t),INTENT(in)::u,v
    TYPE(polar_t)::o
    COMPLEX::a,b,c
    a = u%l*EXP(i*u%th*pi)
    b = v%l*EXP(i*v%th*pi)
    c = a+b
    o%l = ABS(c)
    o%th = ATAN2(imag(c),REAL(c))/pi
  END FUNCTION add_pp
```

Modulos


```
elemental FUNCTION sub_pp(u,v) RESULT(o)
  TYPE(polar_t),INTENT(in)::u,v
  TYPE(polar_t)::o
  COMPLEX::a,b,c
  a = u%l*EXP(i*u%th*pi)
  b = v%l*EXP(i*v%th*pi)
  c = a-b
  o%l = ABS(c)
  o%th = ATAN2(imag(c),REAL(c))/pi
END FUNCTION sub_pp
```

```
elemental FUNCTION mul_pp(u,v) RESULT(o)
  TYPE(polar_t),INTENT(in)::u,v
  TYPE(polar_t)::o
  COMPLEX::a,b,c
  a = u%l*EXP(i*u%th*pi)
  b = v%l*EXP(i*v%th*pi)
  c = a*b
  o%l = ABS(c)
  o%th = ATAN2(imag(c),REAL(c))/pi
END FUNCTION mul_pp
```

```
elemental FUNCTION div_pp(u,v) RESULT(o)
  TYPE(polar_t),INTENT(in)::u,v
  TYPE(polar_t)::o
  COMPLEX::a,b,c
  a = u%l*EXP(i*u%th*pi)
  b = v%l*EXP(i*v%th*pi)
  c = a/b
  o%l = ABS(c)
  o%th = ATAN2(imag(c),REAL(c))/pi
END FUNCTION div_pp
END MODULE polar_mod
```

Modulos

```
PROGRAM main
  USE polar_mod
  IMPLICIT NONE
  CALL test_member
  CALL test_other
  CALL test_scalar
  CALL test_real
CONTAINS
SUBROUTINE test_member
  TYPE(polar_t),DIMENSION(3)::b
  b = polar_t(1.0,0.5)
  WRITE(*,*) 'test_member'
  WRITE(*,*) b
  b(:) = b(:)/b(1)
  WRITE(*,*) b
  WRITE(*,*) ''
END SUBROUTINE test_member
SUBROUTINE test_other
  TYPE(polar_t),DIMENSION(3)::b
  TYPE(polar_t),DIMENSION(3)::c
  b = polar_t(1.0,0.5)
  c = polar_t(1.0,0.5)
  WRITE(*,*) 'test_other'
  WRITE(*,*) b
  b(:) = b(:)/c(1)
  WRITE(*,*) b
  WRITE(*,*) ''
END SUBROUTINE test_other
```



```
SUBROUTINE test_scalar
  TYPE(polar_t),DIMENSION(3)::b
  TYPE(polar_t)::c
  b = polar_t(1.0,0.5)
  c = b(1)
  WRITE(*,*) 'test_scalar'
  WRITE(*,*) b
  b(:) = b(:)/c
  WRITE(*,*) b
  WRITE(*,*) ''
END SUBROUTINE test_scalar

SUBROUTINE test_real
  REAL,DIMENSION(3)::b
  b = 2.0
  WRITE(*,*) 'test_real'
  WRITE(*,*) b
  b(:) = b(:)/b(1)
  WRITE(*,*) b
  WRITE(*,*) ''
END SUBROUTINE test_real
END PROGRAM main
```


Modulos

```
PROGRAM main
  USE polar_mod
  IMPLICIT NONE
  CALL test_member
  CALL test_other
  CALL test_scalar
  CALL test_real
CONTAINS
SUBROUTINE test_member
  TYPE(polar_t),DIMENSION(3)::b
  b = polar_t(1.0,0.5)
  WRITE(*,*) 'test_member'
  WRITE(*,*) b
  b(:) = b(:)/b(1)
  WRITE(*,*) b
  WRITE(*,*) ''
END SUBROUTINE test_member
SUBROUTINE test_other
  TYPE(polar_t),DIMENSION(3)::b
  TYPE(polar_t),DIMENSION(3)::c
  b = polar_t(1.0,0.5)
  c = polar_t(1.0,0.5)
  WRITE(*,*) 'test_other'
  WRITE(*,*) b
  b(:) = b(:)/c(1)
  WRITE(*,*) b
  WRITE(*,*) ''
END SUBROUTINE test_other
```

Modulos

```
SUBROUTINE test_scalar
  TYPE(polar_t),DIMENSION(3)::b
  TYPE(polar_t)::c
  b = polar_t(1.0,0.5)
  c = b(1)
  WRITE(*,*) 'test_scalar'
  WRITE(*,*) b
  b(:) = b(:)/c
  WRITE(*,*) b
  WRITE(*,*) ''
END SUBROUTINE test_scalar

SUBROUTINE test_real
  REAL,DIMENSION(3)::b
  b = 2.0
  WRITE(*,*) 'test_real'
  WRITE(*,*) b
  b(:) = b(:)/b(1)
  WRITE(*,*) b
  WRITE(*,*) ''
END SUBROUTINE test_real
END PROGRAM main
```

Modulos

Exercicio059

Exemplo

ftp://ftp1.cptec.inpe.br/pkubota/curso_fortran/ModuleSubroutineGrads.f90

Modulos

```
MODULE ModTypes  
IMPLICIT NONE
```

```
TYPE :: Grads
```

```
  INTEGER :: ncol=-1
```

```
  INTEGER :: nlin=-1
```

```
  INTEGER :: ntim=-1
```

```
  INTEGER :: nvar=-1
```

```
  REAL :: undef=-9.999E+20
```

```
  REAL :: loni=-1.0
```

```
  REAL :: dlon=-1.0
```

```
  REAL :: lati=-1.0
```

```
  REAL :: dlat=-1.0
```

```
  CHARACTER(LEN=12) :: itime='06Z06JAN2003'
```

```
  CHARACTER(LEN=4 ) :: dtime='24hr'
```

```
  REAL :: nlevs(1)=(/1000/)
```

```
  REAL(KIND=4), POINTER :: Lat(:)
```

```
  REAL(KIND=4), POINTER :: RA(:, :, :)
```

```
END TYPE Grads
```

```
CONTAINS
```

CONTINUA

```
SUBROUTINE GeraBin(NAMEBIN,var)
  IMPLICIT NONE
  CHARACTER(LEN=*) , INTENT(IN ) :: NAMEBIN
  TYPE(Grads) , INTENT(IN ) :: var(:, :)
  INTEGER :: lrec
  INTEGER :: ios
  INTEGER :: unit=1
  INTEGER :: irec
  INTEGER :: it
  INTEGER :: iv
  INTEGER :: iz , i, j
  INQUIRE(IOLENGTH=lrec) var(1,1)%RA(:, :, 1)
  OPEN(unit, FILE=TRIM(NAMEBIN), STATUS='UNKNOWN', FORM='UNFORMATTED', &
    ACCESS='DIRECT' , ACTION='WRITE', RECL=lrec, iostat=ios)
  irec=0
  DO it=1, SIZE(var, 2)
    DO iv=1, SIZE(var, 1)
      DO iz=1, SIZE(var(iv, it)%RA, 3)
        irec=irec+1
        WRITE(unit, rec=irec) var(iv, it)%RA(:, :, iz)
      END DO
    END DO
  END DO
  CLOSE(unit, STATUS='KEEP')
END SUBROUTINE GeraBin
```

CONTINUA

```
SUBROUTINE GeraCtl(NAMEBIN,var)
  CHARACTER(LEN=*), INTENT(IN ) :: NAMEBIN
  TYPE(GRADS) , INTENT(IN ) :: var(:, :)
  INTEGER :: ios
  CHARACTER(LEN=256) :: NAMECTL
  INTEGER :: unit=1
  CHARACTER(LEN=15) :: vname
  INTEGER :: iv,i

  NAMECTL=NAMEBIN(1:LEN(TRIM(NAMEBIN))-4)//'.ctl'

  OPEN(unit,FILE=TRIM(NAMECTL),STATUS='UNKNOWN',FORM='FORMATTED' ,&
ACCESS='SEQUENTIAL',ACTION='WRITE',iostat=ios)

  WRITE(unit,'(A,A )')'dset ^',TRIM(NAMEBIN)
  WRITE(unit,'(A )')'options '
  WRITE(unit,'(A,e10.4 )')'undef ',var(1,1)%undef
  WRITE(unit,'(A )')'title curso de fortran'
  WRITE(unit,'(A,I10,A,2F12.5)')'xdef ', var(1,1)%ncol , ' linear ',var(1,1)%loni,var(1,1)%dlon

  IF(ASSOCIATED(var(1,1)%Lat)) THEN
    WRITE(unit,'(A,I10,A,F12.5)')'ydef ', var(1,1)%nlin , ' levels ',var(1,1)%Lat(1)
    IF(SIZE(var(1,1)%Lat) > 2 ) WRITE(unit,'(10F12.5)')(var(1,1)%Lat(i),i=2,SIZE(var(1,1)%Lat))
  ELSE
    WRITE(unit,'(A,I10,A,2F12.5)')'ydef ', var(1,1)%nlin , ' linear ',var(1,1)%lati,var(1,1)%dlat
  END IF
```

CONTINUA

```
WRITE(unit,'(A,I10,A,2A,A )')'tdef ', var(1,1)%ntim , ' linear ',var(1,1)%itime,' ', var(1,1)%dtime
WRITE(unit,'(A,I10,A )')'zdef ', SIZE(var(1,1)%nlevs),' levels '
WRITE(unit,'(5F10.2 )')(var(1,1)%nlevs(i),i=1,SIZE(var(1,1)%nlevs))
WRITE(unit,'(A,I5 )')'vars ',var(1,1)%nvar
DO iv=1,var(1,1)%nvar
    WRITE(vname,'(a3,I2.2 )')'var',iv
    WRITE(unit,'(A,I5,A,A,A )')TRIM(vname),SIZE(var(iv,1)%RA,3),' 99 ',' variavel ', '( )'
    vname= ' '
END DO
WRITE(unit,'(A )')'endvars'
CLOSE(unit,STATUS='KEEP')
END SUBROUTINE GeraCtl
END MODULE ModTypes
```

CONTINUA

```
PROGRAM main
USE ModTypes, Only : Grads,GeraBin,GeraCtl
IMPLICIT NONE
INTEGER, PARAMETER :: ncol=5
INTEGER, PARAMETER :: nlin=5
INTEGER, PARAMETER :: ntim=2
INTEGER, PARAMETER :: nvar=1
CHARACTER(LEN=256) :: NAMEBIN='teste.bin'
CHARACTER(LEN=256) :: NAMECTL='teste.ctl'
INTEGER :: unit=10
REAL :: undef=9.999E+20
REAL :: loni=0.0
REAL :: dlon=1.5
REAL :: lati=-60.0
REAL :: dlat=1.5
CHARACTER(LEN=12) :: itime='06Z06JAN2003'
CHARACTER(LEN=4 ) :: dtime='24hr'
CHARACTER(LEN=5) :: vname
INTEGER :: nzlev(nvar)
REAL(KIND=4) :: RA(nlin,ncol)
REAL(KIND=4) :: RT(ntim)
REAL(KIND=4), TARGET :: RLAT(nlin)
INTEGER :: lrec
INTEGER :: j
INTEGER :: ios
INTEGER :: i,it,iv,iz,irec
TYPE(Grads), ALLOCATABLE :: var(:,,)
```


CONTINUA

```
!RA=RESHAPE(SOURCE=(&  
! 2.,2.,2.,2.,2.,&  
! 2.,1.,1.,1.,2.,&  
! 2.,1.,0.,1.,2.,&  
! 2.,1.,1.,1.,2.,&  
! 2.,2.,2.,2.,2./),&  
! SHAPE=(/nlin,ncol/))  
RT=RESHAPE(SOURCE=(&  
2.,2.,2.,2.,2.,&  
2.,1.,1.,1.,2.,&  
2.,1.,0.,1.,2.,&  
2.,1.,1.,1.,2.,&  
2.,2.,2.,2.,2./),&  
SHAPE=(/ntim/))  
nzlev=RESHAPE(SOURCE=(/1/),SHAPE=(/nvar/))  
FORALL(I=1:nlin) RLAT(i) = -60.0 + (i-1)*dlat  
ALLOCATE(var(nvar,ntim))  
DO it=1, ntim  
    DO iv=1,nvar  
        ALLOCATE(var(iv,it)%RA(nlin,ncol,nzlev(iv)))  
    END DO  
END DO  
! WHERE(RA==1)  
! RA=0  
! ELSEWHERE(RA==0)  
! RA=1  
! END WHERE  
FORALL(I=2:nlin-1, J=2:ncol-1)  
    RA(i,j)=5.0  
END FORALL
```

CONTINUA

```
DO it=1, SIZE(var,2)
  DO iv=1,SIZE(var,1)
    var(iv,it)%Lat => RLAT
    var(iv,it)%ncol = ncol
    var(iv,it)%nlin = nlin
    var(iv,it)%ntim = ntim
    var(iv,it)%nvar = nvar
    var(iv,it)%undef = undef
    var(iv,it)%loni = loni
    var(iv,it)%dlon = dlon
    var(iv,it)%lati = lati
    var(iv,it)%dlat = dlat
    var(iv,it)%itime = itime
    var(iv,it)%dtime = dtime
    DO iz=1,SIZE(var(iv,it)%RA,3)
      FORALL(I=1:nlin, J=1:ncol)
        !var(iv,it)%RA(i,j,iz)=RT(it) + it*RT(it)
        var(iv,it)%RA(i,j,iz)=RA(i,j) + it*RA(i,j)
      END FORALL
    END DO
  END DO
END DO

CALL GeraBin(NAMEBIN,var)
CALL GeraCtl(NAMEBIN,var)

END PROGRAM main
```

EXEMPLOS

`./CursodeFortranInpe2013/exercicio_extras/ModuleMatrixTamanhoAssumido.f90`

`./CursodeFortranInpe2013/exercicio_extras/Difusion1D.f90`

`./CursodeFortranInpe2013/exercicio_extras/Difusion2D.f90`

`./CursodeFortranInpe2013/exercicio_extras/Advection2D.f90`

`./CursodeFortranInpe2013/exercicio_extras/ModuleMatrixAjustaveis.f90`

`./CursodeFortranInpe2013/exercicio_extras/ModuleMatrixFormaAssumida.f90`

Comandos de Entrada/Saída de Dados

PRINT OPEN READ WRITE CLOSE REWIND BACKSPACE

Exercicio060

```
PROGRAM F_OUT
IMPLICIT NONE
INTEGER, PARAMETER :: DP=&
SELECTED_REAL_KIND(15,300)
REAL(DP) :: A, B, C
A= 1.0_DP/7.0_DP
B= SQRT(2.0_DP)
C= 4*ATAN(1.0_DP)
PRINT*, A, B, C
END PROGRAM F_OUT
```

**Entrada formatada a
partir do teclado**

```
READ(*,FMT='(1X,F7.5,5X,F9.6,3X,F11.8)')A, B, C
```

OU

```
READ(*,FMT='(1X,F7.5,5X,F9.6,3X,F11.8)')A, B, C
```

Exercicio061

```
PROGRAM F_OUT_FORMATADO
IMPLICIT NONE
INTEGER, PARAMETER :: DP=&
SELECTED_REAL_KIND(15,300)
REAL(DP) :: A, B, C
A= 1.0_DP/7.0_DP
B= SQRT(2.0_DP)
C= 4*ATAN(1.0_DP)
PRINT"(F7.5,1X,F9.7,1X,F11.9)", A, B, C
PRINT"(F7.7,1X,F9.9,1X,E11.5)", A, B, C
PRINT"('A= ',F7.5,2X,'B= ',F9.7,3X,'C= ',F11.9)", A, B, C
END PROGRAM F_OUT_FORMATADO
```

**Saida formatada a
partir do teclado**

```
WRITE(*,FMT='(1X,F7.5,5X,F9.6,3X,F11.8)')A, B, C
```

OU

```
WRITE(*,FMT='(1X,F7.5,5X,F9.6,3X,F11.8)')A, B, C
```

Comandos de Entrada/Saída de Dados

Especificador	Significado
Iw.w	Escreve um inteiro com até w caracteres
Fw.d	Escreve um real com w caracteres das quais d são decimais
Ew.d	Escreve um real na forma exponencial com w caracteres das quais d são decimais
Gw.d	Escreve um real na forma normal ou exponencial com w caracteres das quais d são decimais
Aw	Escreve uma string com w caracteres
A	Escreve uma string do inicial até a ultimo caracter inicializado
nX	Pula n caracteres
/	Quebra de linha

Comandos de Entrada/Saída de Dados

Saída de dados em um arquivo

```
OPEN(10, FILE="EX1.DAT").
```

```
WRITE(10, FMT="(F7.5,1X,F9.7,1X,F11.9)") A, B, C.
```

```
CLOSE(10)
```

Exercicio062

```
PROGRAM ARQ_SAI
IMPLICIT NONE
INTEGER, PARAMETER :: DP= SELECTED_REAL_KIND(15,300)
REAL(DP) :: A, B, C
A= 1.0_DP/7.0_DP
B= SQRT(2.0_DP)
C= 4*ATAN(1.0_DP)
OPEN(10, FILE="EX1.DAT")
WRITE(10, FMT="(F7.5,1X,F9.7,1X,F11.9)") A, B, C
CLOSE(10)
END PROGRAM ARQ_SAI
```

ftp://ftp1.cptec.inpe.br/pkubota/curso_fortran/IOOPEN.f90

Comandos de Entrada/Saída de Dados

Leitura/escrita de dados em arquivos.

STATUS="OLD/NEW"

OLD=deve existir

NEW=não deve existir

read(10, fmt="(f7.5,1x,f9.7,1x,f11.9)") a, b, c

write(11, fmt="(3(e11.5,1x))")d, e, f

Exercicio063

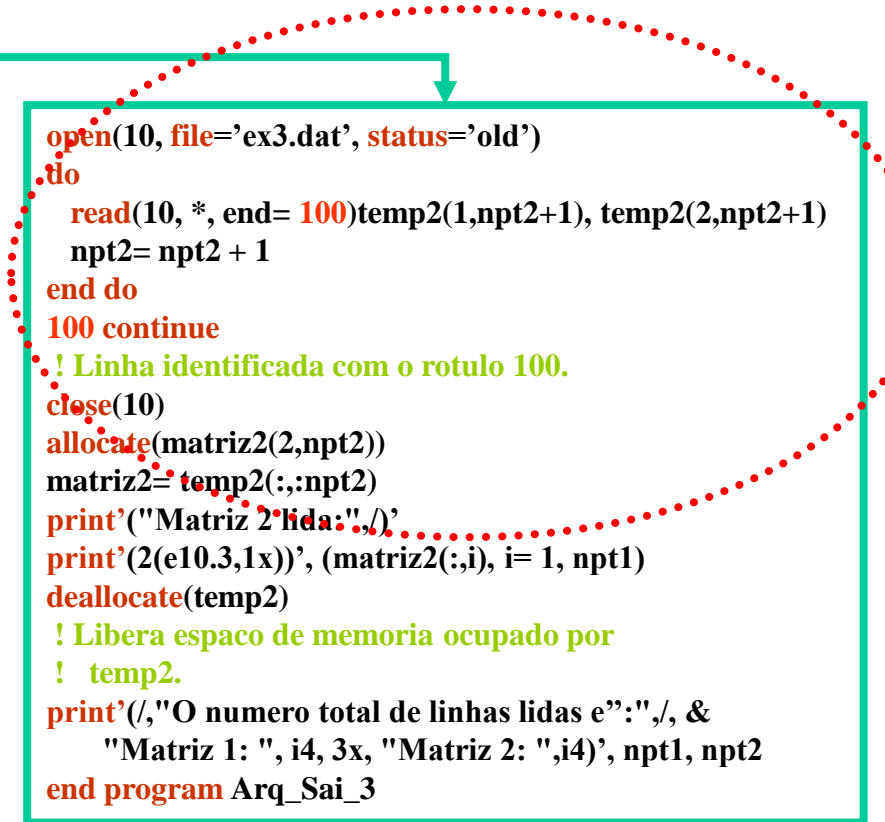
```
PROGRAM ARQ_SAI_2
IMPLICIT NONE
INTEGER, PARAMETER :: DP= SELECTED_REAL_KIND(15,300)
REAL(DP) :: A, B, C ! DADOS DE ENTRADA.
REAL(DP) :: D, E, F ! DADOS DE SAIDA.
OPEN(10, FILE="EX1.DAT", STATUS="OLD")
OPEN(11, FILE="EX2.DAT", STATUS="NEW")
READ(10, FMT="(F7.5,1X,F9.7,1X,F11.9)") A, B, C
PRINT*, "OS VALORES LIDOS FORAM:"
PRINT*, "A= ",A
PRINT*, "B= ",B
PRINT*, "C= ",C
D= A + B + C
E= B**2 + COS(C)
F= SQRT(A**2 + B**3 + C**5)
WRITE(11, FMT="(3(E11.5,1X))")D, E, F
END PROGRAM ARQ_SAI_2
```

Comandos de Entrada/Saída de Dados

Leitura/escrita de dados em arquivos. número desconhecido de linha

Exercicio064

```
program Arq_Sai_3
implicit none
integer, parameter :: dp= SELECTED_REAL_KIND(15,300)
integer :: controle, i, npt1= 0, npt2= 0
character(len= 1) :: char
real(kind=dp), dimension(:,,:), allocatable :: temp1
real(kind=dp), dimension(:,,:), allocatable :: temp2
real(kind=dp), dimension(:,,:), allocatable :: matriz1
real(kind=dp), dimension(:,,:), allocatable :: matriz2
allocate(temp1(2,1000), temp2(2,1000))
! Aloca matrizes temporarias.
! Estima um limite superior no numero de linhas.
! Metodo 1: usando opcao IOSTAT.
open(10, file='ex3.dat', status='old')
do
  read(10, *, iostat= controle) temp1(1,npt1+1), temp1(2,npt1+1)
  if(controle < 0) exit      ! Final de arquivo detectado.
  npt1= npt1 + 1
end do
close(10)
allocate(matriz1(2,npt1))
matriz1= temp1(:,npt1)
write(*, fmt="( 'Matriz 1 lida:',/)" )
print'(2(e10.3,1x))', (matriz1(:,i), i= 1, npt1)
deallocate(temp1)
! Libera espaco de memoria ocupado por emp1.
print'(/,a)', 'Pressione ENTER/RETURN para continuar.'
read(*,'(a)')char
! Metodo 2: usando opcao END.
```



```
open(10, file='ex3.dat', status='old')
do
  read(10, *, end= 100) temp2(1,npt2+1), temp2(2,npt2+1)
  npt2= npt2 + 1
end do
100 continue
! Linha identificada com o rotulo 100.
close(10)
allocate(matriz2(2,npt2))
matriz2= temp2(:,npt2)
print'("Matriz 2 lida:",/)'
print'(2(e10.3,1x))', (matriz2(:,i), i= 1, npt1)
deallocate(temp2)
! Libera espaco de memoria ocupado por
! temp2.
print'(/,"O numero total de linhas lidas e":",/, &
      "Matriz 1: ", i4, 3x, "Matriz 2: ",i4)', npt1, npt2
end program Arq_Sai_3
```


Comandos de Entrada/Saída de Dados

Leitura/escrita de dados em arquivos. número desconhecido de linha

ftp://ftp1.cptec.inpe.br/pkubota/curso_fortran/ARQ_SAI_3.f90

```
PROGRAM ARQ_SAI_3
IMPLICIT NONE
INTEGER, PARAMETER :: DP= SELECTED_REAL_KIND(15,300)
INTEGER :: CONTROLE, I, NPT1= 0, NPT2= 0
CHARACTER(LEN= 1) :: CHAR
REAL(KIND=DP), DIMENSION(:,,:), ALLOCATABLE :: TEMP1
REAL(KIND=DP), DIMENSION(:,,:), ALLOCATABLE :: TEMP2
REAL(KIND=DP), DIMENSION(:,,:), ALLOCATABLE :: MATRIZ1
REAL(KIND=DP), DIMENSION(:,,:), ALLOCATABLE :: MATRIZ2
ALLOCATE(TEMP1(2,1000), TEMP2(2,1000))
! ALOCA MATRIZES TEMPORARIAS. ESTIMA UMLIMITE SUPERIOR NO NUMERO DE LINHAS.
! METODO 1: USANDO OPCAO IOSTAT.
OPEN(10, FILE='EX3.DAT', STATUS='OLD')
DO
    READ(10, *, IOSTAT= CONTROLE)TEMP1(1,NPT1+1), TEMP1(2,NPT1+1)
    IF(CONTROLE < 0) EXIT ! FINAL DE ARQUIVO DETECTADO.
    NPT1= NPT1 + 1
END DO
CLOSE(10)
ALLOCATE(MATRIZ1(2,NPT1))
MATRIZ1= TEMP1(:, :NPT1)
WRITE(*, FMT="( 'MATRIZ 1 LIDA: ',/ )")
PRINT'(2(E10.3,1X))', (MATRIZ1(:,I), I= 1, NPT1)
DEALLOCATE(TEMP1)
! LIBERA ESPACO DE MEMORIA OCUPADO POR EMP1.
PRINT'(/,A)', 'PRESSIONE ENTER/RETURN PARA CONTINUAR.'
READ(*,'(A)')CHAR
! METODO 2: USANDO OPCAO END.
```

CONTINUA

```
OPEN(10, FILE='EX3.DAT', STATUS='OLD')
DO
READ(10, *, END= 100)TEMP2(1,NPT2+1), TEMP2(2,NPT2+1)
NPT2= NPT2 + 1
END DO
100 CONTINUE
! LINHA IDENTIFICADA COM O ROTULO 100.
CLOSE(10)
ALLOCATE(MATRIZ2(2,NPT2))
MATRIZ2= TEMP2(:,NPT2)
PRINT'("MATRIZ 2 LIDA:",/)'
PRINT'(2(E10.3,1X))', (MATRIZ2(:,I), I= 1, NPT1)
DEALLOCATE(TEMP2)
! LIBERA ESPACO DE MEMORIA OCUPADO POR
! TEMP2.
PRINT'(/,"O NUMERO TOTAL DE LINHAS LIDAS E":",/, "MATRIZ 1: ", I4, 3X, "MATRIZ 2: ",I4)', NPT1, NPT2
END PROGRAM ARQ_SAI_3
```

ftp://ftp1.cptec.inpe.br/pkubota/curso_fortran/ModuleEntradaSaidaToGrads.f90

Comandos de Entrada/Saída de Dados

Declaração NAMELIST

A estrutura do **Namelist** pode ser usada para lidar com diversas variáveis ou com um simples grupo. Por exemplo

```
PROGRAM NAMEHUNT
  IMPLICIT NONE
  INTEGER :: DEER=2, RABBIT=4, MOOSE=1, DUCK=0
  NAMELIST /GAME/ DEER,RABBIT,MOOSE,DUCK
  WRITE(*,*) 'CAPTURED GAME:'
  READ(*,NML=GAME)
  WRITE(*,NML=GAME)
END PROGRAM NAMEHUNT
```

!A ESTRADA INICIA COM & E O NOME DO GRUPO , E FINALIZA COM /:
&GAME RABBIT=1, DEER=3, HORSE=1 /

!OUTPUT:

&GAME

DEER= 3, RABBIT= 1, MOOSE= 1, DUCK= 0, /

Comandos de Entrada/Saída de Dados

Declaração NAMELIST

ftp://ftp1.cptec.inpe.br/pkubota/curso_fortran/NameList.f90

```
PROGRAM TESTA_NAMELIST
IMPLICIT NONE
INTEGER, PARAMETER :: DP= SELECTED_REAL_KIND(15,300)
INTEGER :: NO_DE_OVOS
INTEGER :: LITROS_DE_LEITE,
INTEGER :: QUILOS_DE_ARROZ= 12
INTEGER :: NUMERO
! NOTE QUE QUILOS_DE_ARROZ FOI INICIALIZADO.
INTEGER, DIMENSION(10) :: LISTA_INT1
INTEGER, DIMENSION(10) :: LISTA_INT2
REAL , DIMENSION(10) :: ALEAT, VET_TES= 1.0
REAL(DP) :: PI, E_EULER, M_ELECTRON, C_VACUUM
LOGICAL :: TES
CHARACTER(LEN= 10) :: NOME

NAMELIST /FOME/ NOME, NO_DE_OVOS, LITROS_DE_LEITE, QUILOS_DE_ARROZ

NAMELIST /SAI/ NUMERO, LISTA_INT1, LISTA_INT2, VET_TES

NAMELIST /CTES/ TES, PI, E_EULER, M_ELECTRON, C_VACUUM

OPEN(10, FILE= "DADOS.ENT", STATUS= "OLD")
OPEN(11, FILE= "DADOS.DAT")
READ(10, NML= FOME)
READ(10, NML= SAI)
READ(10, NML= CTES)
```

Comandos de Entrada/Saída de Dados

Declaração NAMELIST

ftp://ftp1.cptec.inpe.br/pkubota/curso_fortran/NameList.f90

! MOSTRA NA TELA OS VALORES LIDOS.

PRINT'(A)', "LISTA /FOME/ LIDA:"

WRITE(*, NML= FOME)

PRINT'(/A)', "LISTA /SAI/ LIDA:"

WRITE(*, NML= SAI)

PRINT'(/A)', "LISTA /CTES/ LIDA:"

WRITE(*, NML= CTES)

PRINT'(/A)', "PARA CONFIRMAR:"

PRINT'("VETOR VET_TES: ",10(F4.2,X))', VET_TES

PRINT'(/,"LISTA 1:",10I3)', LISTA_INT1

PRINT'(/,"LISTA 2:",10I3)', LISTA_INT2

! ALTERA ALGUNS VALORES E GRAVA AS LISTAS NO ARQUIVO.

CALL RANDOM_NUMBER(ALEAT)

PRINT'(/,"LISTA DE NO. ALEATORIOS:",/,10(F7.5,X))', ALEAT

LISTA_INT1= LISTA_INT22**

LISTA_INT2= ALEAT*LISTA_INT2

WRITE(11, NML= SAI)

WRITE(11, NML= CTES)

WRITE(11, NML= FOME) ! VERIFIQUE DADOS.DAT.

END PROGRAM TESTA_NAMELIST

Comandos de Entrada/Saída de Dados

Pegando as Propriedades do arquivos **INQUIRE**

A propriedade de uma leitura ou uma unidade pode ser requisitada através:

```
INQUIRE(FILE='XXX', LIST...)  
INQUIRE(UNIT=X, LIST...)
```

A lista são algumas das opções do **inquire**

```
exists  (le exists; logical)  
opened (le or unit is opened; logical)  
form   ('formatted'/'unformatted')  
access ('sequential'/'direct';character)  
action ('read', 'write', 'readwrite';character)  
... etc.
```

!EXAMPLE:

```
LOGICAL :: ISFILE
```

```
INQUIRE (FILE='FOO.DAT', EXISTS=ISFILE)
```

```
IF (.NOT. ISFILE) THEN
```

```
    WRITE(*,*) 'FILE DOES NOT EXIST'
```

```
ELSE
```

```
    WRITE(*,*) 'FILE EXIST'
```

```
ENDIF
```

```

PROGRAM Main
IMPLICIT NONE
INTEGER :: UnitFile=10
LOGICAL :: lopen
LOGICAL :: exist
INTEGER :: ios
CHARACTER(LEN=255) :: access,name
INQUIRE (unit=UnitFile,OPENED=lopen,IOSTAT=ios,ACCESS=access,EXIST=exist,NAME=name)

```

```

!INQUIRE ({[UNIT=]unitspec (integer) | FILE=file (character)}
![, ACCESS=access (character) ('SEQUENTIAL','DIRECT' ) ]
![, BINARY=binary (character) ('YES','NO' ) ]
![, BLANK=blank (character) ('NULL','ZERO' ) ]
![, BLOCKSIZE=blocksize (integer ) (# ) ]
![, DIRECT=direct (character) ('YES','NO' ) ]
![, ERR=errlabel (integer ) (# ) ]
![, EXIST=exist (LOGICAL ) (.TRUE.,.FALSE. ) ]
![, FORM=form (character) ('FORMATTED','UNFORMATTED','UNDEFINED' ) ]
![, FORMATTED=formatted (character) ('YES' ,'NO','UNKNOWN' ) ]
![, IOSTAT=iocheck (integer ) (0=ok ,#=error ) ]
![, IOFOCUS=focus (LOGICAL ) (.TRUE.,.FALSE. ) ]
![, ACTION=mode (character) ('READ','WRITE','READWRITE','UNDEFINED' ) ]
![, NAME=name (character) (name ) ]
![, NAMED=named (LOGICAL ) (.TRUE.,.FALSE. ) ]
![, NEXTREC=nextrec (integer ) (0---infinito ) ]
![, NUMBER=num (integer ) (0---infinito ) ]
![, OPENED=opened (LOGICAL ) (.TRUE.,.FALSE. ) ]
![, RECL=recl (integer ) (0---infinito ) ]
![, SEQUENTIAL=seq (character) (YE ; NO,'UNKNOWN' ) ]
![, SHARE=share (character) ('COMPAT','DENYRW', 'DENYWR', 'DENYRD', and 'DENYNONE' ) ]
![, UNFORMATTED=unformatted (character) (YE ; NO ) ]
![, IOLENGTH=iolength (integer ) (# ) ]
![, POSITION=iposition (character) ('REWIND','APPEND','ASIS','UNDEFINED') ) ]

```

```

PRINT*,lopen,ios,TRIM(access),exist,TRIM(name)
OPEN(unit = UnitFile,FILE='INQUIRE.txt')
INQUIRE (unit=UnitFile,OPENED=lopen,IOSTAT=ios,ACCESS=access,EXIST=exist,NAME=name)
PRINT*,lopen,ios,TRIM(

```

Comandos de Entrada/Saída de Dados

Declaração INQUIRE

IOSTAT= <ios>, A variável <ios> é a única definida se uma condição de erro ocorre durante a execução do INQUIRE.

ERR= <err-label>, tem o mesmo significado descrito no comando OPEN (seção 9.4).

EXIST= <ex>, onde <ex> é uma variável lógica. O valor .TRUE. é atribuído se o arquivo (ou unidade) existir e .FALSE. em caso contrário.

OPENED= <open>, onde <open> é uma variável lógica. O valor .TRUE. é atribuído se o arquivo (ou unidade) estiver conectado a uma unidade (ou arquivo) e .FALSE. em caso contrário.

NUMBER= <num>, onde <num> é uma variável inteira à qual é atribuído o número da unidade conectada ao arquivo, ou -1 se nenhuma unidade estiver conectada ao arquivo.

ACCESS= <acc>, onde <acc> é uma variável de caracteres à qual são atribuídos um dos valores 'SEQUENTIAL' ou 'DIRECT' dependendo do método de acesso para um arquivo que está conectado e 'UNDEFINED' se não houver conexão.

SEQUENTIAL= <seq>,

DIRECT= <dir>, onde <seq> e <dir> são variáveis de caracteres às quais são atribuídos os valores 'YES', 'NO' ou 'UNKNOWN', dependendo se o arquivo puder ser aberto para acesso direto ou seqüencial, respectivamente, ou se isto não pode ser determinado.

FORM= <frm>, onde <frm> é uma variável de caracteres à qual são atribuídos um dos valores 'FORMATTED'

ou 'UNFORMATTED', dependendo na forma para a qual o arquivo é realmente conectado, ou 'UNDEFINED' se não houver conexão.

Comandos de Entrada/Saída de Dados

Declaração INQUIRE

FORMATTED= <fmt>,

UNFORMATTED= <unf>, onde <fmt> e <unf> são variáveis de caracteres às quais são atribuídos os valores 'YES', 'NO' ou 'UNKNOWN', dependendo se o arquivo puder ser aberto para acesso formatado ou não formatado, respectivamente, ou se isto não pode ser determinado.

RECL= <rec>, onde <rec> é uma variável inteira à qual é atribuído o valor do tamanho do registro de um arquivo conectado para acesso direto, ou o tamanho máximo do registro permitido para um arquivo conectado para acesso seqüencial. O comprimento é o número de caracteres para registros formatados contendo somente caracteres do tipo padrão e dependente do sistema em caso contrário. Se não houver conexão, <rec> resulta indefinido.

ACTION= <act>, onde <act> é uma variável de caracteres à qual são atribuídos os valores 'READ', 'WRITE' ou 'READWRITE', conforme a conexão. Se não houver conexão, o valor é 'UNDEFINED'. READ= <rd>, onde <rd> é uma variável de caracteres à qual são atribuídos os valores 'YES', 'NO' ou 'UNKNOWN', dependendo se leitura é permitida, não permitida ou indeterminada para o arquivo.

WRITE= <wr>, onde <wr> é uma variável de caracteres à qual são atribuídos os valores 'YES', 'NO' ou 'UNKNOWN', dependendo se escrita é permitida, não permitida ou indeterminada para o arquivo.

READWRITE= <rw>, onde <rw> é uma variável de caracteres à qual são atribuídos os valores 'YES', 'NO' ou 'UNKNOWN', dependendo se leitura e escrita são permitidas, não permitidas ou indeterminadas para o arquivo.

Comandos de Entrada/Saída de Dados

ftp://ftp1.cptec.inpe.br/pkubota/curso_fortran/IOFILE.f90

```
PROGRAM Main
IMPLICIT NONE
INTEGER :: UnitFile=10
INTEGER :: iocheck
INTEGER :: ios
LOGICAL :: lopen
LOGICAL :: exist
CHARACTER(LEN=255) :: access,form,cname
CHARACTER(LEN=256) :: name
!
!OPEN ([UNIT=]unitspec
![, ACCESS=access (character )('APPEND', 'DIRECT','SEQUENTIAL' ) ]
![, BLANK=blanks (character )('NULL' , 'ZERO' ) ]
![, BLOCKSIZE=blocksize(integer )(# ) ]
![, ERR=errlabel (integer )(# ) ]
![, FILE=file (character )(namefile ) ]
![, FORM=form (character )('FORMATTED','UNFORMATTED','BINARY' ) ]
![, IOSTAT=iocheck (integer )(0=ok ,#=error ) ]
![, ACTION=mode (character )('READ','WRITE','READWRITE' ) ]
![, RECL=recl (integer )(# ) ]
![, SHARE=share (character )('COMPAT','DENYRW','DENYWR','DENYRD','DENYNONE' ) ]
![, STATUS=status (character )('OLD','NEW','SCRATCH','UNKNOWN' ) ]
![, IOFOCUS=focus (LOGICAL )(.TRUE.,.FALSE. ) ]
![, TITLE=title )])
!
OPEN(unit=UnitFile,FILE='INQUIRE.txt',FORM='UNFORMATTED',ACCESS='SEQUENTIAL',&
STATUS='UNKNOWN',ACTION='WRITE')
```

Comandos de Entrada/Saída de Dados

```
!  
!WRITE ([UNIT=] unitspec  
![, [{[ FMT=] formatspec] [{[ NML=] nmlspec} ]]  
![, ERR=errlabel (integer )(# )]  
![, IOSTAT=iocheck (integer )(# )]  
![, REC=rec (integer )(# )] )  
!iolist  
!  
cname='LUIZ INACIO DA SILVA'  
WRITE(unit=UnitFile)cname  
!  
! WRITE(unit=UnitFile,FMT='(A)',ERR=10,IOSTAT=iocheck)'LUIZ INACIO DA SILVA'  
! IF(iocheck == 0)CLOSE(unit=UnitFile,STATUS='KEEP')  
!  
CLOSE(unit=UnitFile,STATUS='KEEP')
```

Comandos de Entrada/Saída de Dados

```
!  
!READ { { formatspec, | nmlspec } |  
!([UNIT=] unitspec  
! [ ,[{[FMT=] formatspec] | [NML=]nmlspec}]  
! [ , END=endlabel (integer) (#) ]  
! [ , ERR=errlabel (integer) (#) ]  
! [ , IOSTAT=iocheck (integer) (0=ok ,#=error) ]  
! [ , REC=rec (integer) (#) ]}  
!iolist  
! OPEN(unit=UnitFile,FILE='INQUIRE.txt')  
!  
INQUIRE (FILE='INQUIRE.txt',OPENED=lopen,IOSTAT=ios,ACCESS=access,EXIST=exist,FORM=form)  
PRINT*, ACCESS=',TRIM(access)  
PRINT*, FORM=',TRIM(form)  
PRINT*, opened=',lopen  
!  
! OPEN(unit=UnitFile,FILE='INQUIRE.txt',FORM='FORMATTED',ACCESS='SEQUENTIAL',&  
! STATUS='OLD',ACTION='READ')  
!  
! READ(unit=UnitFile,FMT='(A)',ERR=10,IOSTAT=iocheck,END=10)name  
! READ(unit=UnitFile,FMT='(A)',ERR=10,IOSTAT=iocheck,END=20)name  
!  
10 CONTINUE  
!PRINT*, 10 CONTINUE ' , ' ERROR '  
20 CONTINUE  
!PRINT*, 20 CONTINUE '  
END PROGRAM Main
```

Comandos de Entrada/Saída de Dados

Comando OPEN

OPEN([UNIT=] <int-exp> [,<op-list>])

FILE= <fln>, onde <fln> é uma expressão de caractere que fornece o nome do arquivo..

IOSTAT= <ios>, onde <ios> é uma variável inteira padrão que é fixada a zero se o comando executou corretamente e a um valor positivo em caso contrário.

ERR= <rótulo-erro>, onde <rótulo-erro> é o rótulo de um comando na mesma unidade de âmbito, para o qual o controle do fluxo será transferido caso ocorra algum erro na execução do comando OPEN.

STATUS= <status>, onde <status> é uma expressão de caracteres que fornece o status do arquivo.

O status pode ser um dos seguintes:

'OLD'– Arquivo deve existir.

'NEW'– Arquivo não deve existir. O arquivo será criado pelo comando OPEN. A partir deste momento, o seu status se torna 'OLD'.

'REPLACE'– Se o arquivo não existe, ele será criado. Se o arquivo já existe, este será eliminado e um novo arquivo é criado com o mesmo nome. Em ambos os casos, o status é então alterado para 'OLD'.

'SCRATCH'– O arquivo é temporário e será deletado quando este for fechado com o comando CLOSE ou na saída da unidade de programa.

'UNKNOWN'– Status do arquivo desconhecido; depende do sistema. Este é o valor padrão do especificador, caso este seja omitido. O especificador "FILE=" deve estar presente se 'NEW' ou 'REPLACE' são especificados ou se 'OLD' é especificado e a unidade não está conectada.

Comandos de Entrada/Saída de Dados

Comando OPEN

OPEN([UNIT=] <int-exp> [,<op-list>])

ACCESS= <acc>, onde <acc> é uma expressão de caracteres que fornece o valor 'SEQUENTIAL' ou 'DIRECT'.

Para um arquivo que já exista, este valor deve ser uma opção válida. Se o arquivo ainda não existe, ele será criado com o método de acesso apropriado. Se o especificador é omitido, o valor 'SEQUENTIAL' é assumido. O significado das especificações é:

'DIRECT'– O arquivo consiste em registros acessados por um número de identificação. Neste caso, o tamanho do registro deve ser especificado por "RECL=". Registros individuais podem ser especificados e atualizados sem alterar o restante do arquivo.

'SEQUENTIAL'– O arquivo é escrito/lido seqüencialmente, linha a linha.

FORM= <fm>, onde <fm> é uma expressão de caracteres que fornece os valores 'FORMATTED' ou 'UNFORMATTED', determinando se o arquivo deve ser conectado para E/S formatada ou não formatada. Para um arquivo que já exista, o valor deve ser uma opção válida. Se o arquivo ainda não existe, ele será criado com um conjunto de forma que incluem o forma especificada. Se o especificador é omitido, os valores-padrão são:

'FORMATTED'– Para acesso seqüencial.

'UNFORMATTED'– Para conexão de acesso direto.

RECL= <rl>, onde <rl> é uma expressão inteira cujo valor deve ser positivo.

Para arquivo de acesso direto, a opção especifica o tamanho dos registros e é obrigatório.

Para arquivo seqüencial, a opção especifica o tamanho máximo de um registro, e é opcional com um valor padrão que depende do processador.

Para arquivos formatados, o tamanho é o número de caracteres para registros que contenham somente caracteres-padrão.

Para arquivos não formatados, o tamanho depende do sistema, mas o comando INQUIRE (seção 9.10) pode ser usado para encontrar o tamanho de uma lista de E/S.

Em qualquer situação, para um arquivo que já exista, o valor especificado deve ser permitido para o arquivo.

Comandos de Entrada/Saída de Dados

Comando OPEN

OPEN([UNIT=] <int-exp> [,<op-list>])

BLANK= <bl>, onde <bl> é uma expressão de caracteres que fornece os valores 'NULL' ou 'ZERO'. A conexão deve ser com E/S formatada. Este especificador determina o padrão para a interpretação de brancos em campos de entrada numéricos.

'NULL'– Os brancos são ignorados, exceto que se o campo for completamente em branco, ele é interpretado como zero.

'ZERO'– Os brancos são interpretados como zeros. Se o especificador é omitido, o valor padrão é 'NULL'.

POSITION= <pos>, onde <pos> é uma expressão de caracteres que fornece os valores 'ASIS', 'REWIND' ou 'APPEND'.

O método de acesso deve ser seqüencial e se o especificador é omitido, o valor padrão é 'ASIS'. Um arquivo novo é sempre posicionado no seu início. Para um arquivo que já existe e que já está conectado:

'ASIS'– O arquivo é aberto sem alterar a posição corrente de leitura/escrita no seu interior.

'REWIND'– O arquivo é posicionado no seu ponto inicial.

'APPEND'– O arquivo é posicionado logo após o registro de final de arquivo, possibilitando a inclusão de dados novos.

ACTION= <act>, onde <act> é uma expressão de caracteres que fornece os valores 'READ', 'WRITE' ou 'READWRITE'.

'READ'– Se esta especificação é escolhida, os comandos WRITE, PRINT e ENDFILE não devem ser usados para esta conexão.

'WRITE'– Se esta especificação é escolhida, o comando READ não pode ser usado.

'READWRITE'– Se esta especificação é escolhida, não há restrição.

Se o especificador é omitido, o valor padrão depende do processador.

**DELIM= **, onde é uma expressão de caracteres que fornece os valores 'APOSTROPHE', 'QUOTE' ou 'NONE'.

Se 'APOSTROPHE' ou 'QUOTE' são especificados, o caractere correspondente será usado para delimitar constantes de caractere escritos com formatação de lista ou com o uso da declaração NAMELIST, e será duplicado onde ele aparecer dentro de tal constante de caractere. Além disso, caracteres fora do padrão serão precedidos por valores da espécie.

Comandos de Entrada/Saída de Dados

Comando READ

**READ([UNIT=]<u>, [FMT=] <format>[, IOSTAT= <ios>] &
[, ERR= <err-label>][, END= <end-label>] &
[, EOR= <eor-label>][, ADVANCE= <mode>] &
[, REC= <int-exp>][, SIZE= <size>][, NML= <grp>]) <lista>**

UNIT= <u>, onde <u> é um número de unidade lógica válido ou “*” para a entrada padrão. Caso esta especificação seja o primeiro argumento do comando, a palavra-chave é opcional.

FMT= <format>, onde <format> é uma string de caracteres formatadores, um rótulo válido em Fortran ou “*” para formato livre. Caso esta especificação seja o segundo argumento do comando, a palavra-chave é opcional.

IOSTAT= <ios>, onde <ios> é uma variável inteira padrão que armazena o status do processo de leitura. Os valores possíveis são:

<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 registro é detectada em entrada sem avanço ou quando uma condição de final de arquivo é detectada.

ERR= <err-label>, é um rótulo válido para onde o controle de fluxo é transferido quando ocorre um erro de leitura.

END= <end-label>, é um rótulo válido para onde o controle de fluxo é transferido se uma condição de final de arquivo é encontrada. Esta opção somente existe no comando READ com acesso sequencial.

Comandos de Entrada/Saída de Dados

Comando READ

**READ([UNIT=]<u>, [FMT=] <format>[, IOSTAT= <ios>] &
[, ERR= <err-label>][, END= <end-label>] &
[, EOR= <eor-label>][, ADVANCE= <mode>] &
[, REC= <int-exp>][, SIZE= <size>][, NML= <grp>]) <lista>**

EOR= <eor-label>, é um rótulo válido para onde o controle de fluxo é transferido se uma condição de final de registro é encontrada. Esta opção somente existe no comando **READ** com acesso sequencial e formatado e somente se a especificação **ADVANCE= 'NO'** também estiver presente.

ADVANCE= <mode>, onde <mode> possui dois valores: 'YES' ou 'NO'. A opção 'NO' especifica que cada comando **READ** inicia um novo registro na mesma posição; isto é, implementa entrada de dados sem avanço (non-advancing I/O). A opção padrão é 'YES', isto é a cada leitura o arquivo é avançado em uma posição. Se a entrada sem avanço é usada, então o arquivo deve estar conectado para acesso sequencial e o formato deve ser explícito.

REC= <int-exp>, onde <int-exp> é uma expressão inteira escalar cujo valor é o número de índice do registro lido durante acesso direto. Se **REC** estiver presente, os especificadores **END** e **NML** e o formato livre “*” não podem ser também usados.

SIZE= <size>, onde <size> é uma variável escalar inteira padrão, a qual guarda o número de caracteres lidos. Este especificador somente existe no comando **READ** e somente se a especificação **ADVANCE= 'NO'** também estiver presente.

NML= <grp>, onde <grp> é o nome de um grupo **NAMelist** previamente definido em uma declaração **NAMelist**. O nome <grp> não é uma constante de caracteres e sim um nome válido em Fortran2

Comandos de Entrada/Saída de Dados

Comando WRITE

**WRITE([UNIT=]<u>, [FMT=] <format>[, IOSTAT= <ios>] &
[, ERR= <err-label>][, ADVANCE= <mode>] &
[, REC= <int-exp>][, NML= <grp>]) <lista>**

UNIT= <u>, onde <u> é um número de unidade lógica válido ou “*” para a saída padrão. Caso esta especificação seja o primeiro argumento do comando, a palavra-chave é opcional.

IOSTAT= <ios>, onde <ios> é uma variável inteira padrão que armazena o status do processo de escrita. Os valores possíveis são:
<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 registro é detectada em escrita sem avanço ou quando uma condição de final de arquivo é detectada.

ERR= <err-label>, é um rótulo válido para onde o controle de fluxo é transferido quando ocorre um erro de escrita.

ADVANCE= <mode>, onde <mode> possui dois valores: 'YES' ou 'NO'. A opção 'NO' especifica que cada comando **WRITE** inicia um novo registro na mesma posição; isto é, implementa saída de dados sem avanço (non-advancing I/O). A opção padrão é 'YES', isto é a cada escrita, o arquivo é avançado em uma posição. Se a saída sem avanço é usada, então o arquivo deve estar conectado para acesso sequencial e o formato deve ser explícito.

REC= <int-exp>, onde <int-exp> é uma expressão inteira escalar cujo valor é o número de índice do registro a ser escrito durante acesso direto. Se REC estiver presente, os especificadores END e NML e o formato livre “*” não podem ser também usados.

NML= <grp>, onde <grp> é o nome de um grupo NAMELIST previamente definido em uma declaração NAMELIST. O nome <grp> não é uma constante de caracteres e sim um nome válido em Fortran3 .

Comandos de Entrada/Saída de Dados

Comando REWIND

REWIND([UNIT=]<u>[, IOSTAT=<ios>][ERR=<err-label>])

A a releitura, re-escritura ou verificação por leitura de um registro, uma operação similar pode ser realizada sobre um arquivo completo. Com este propósito, o comando REWIND: pode ser usado para reposicionar um arquivo, cujo número de unidade é especificado pela expressão escalar inteira <u>. Novamente, os demais especificadores opcionais têm o mesmo significado que no comando READ. Se o arquivo já estiver no seu início, nada ocorre. O mesmo ocorre caso o arquivo não exista.

Exemplo: F90

```
PROGRAM Projectile
IMPLICIT NONE
REAL, PARAMETER :: g = 9.8 ! acceleration due to gravity
REAL, PARAMETER :: PI = 3.1415926 ! you know this. don't you?
REAL :: Angle ! launch angle in degree
REAL :: Time ! time to flight
REAL :: Theta ! direction at time in degree
REAL :: U ! launch velocity
REAL :: V ! resultant velocity
REAL :: Vx ! horizontal velocity
REAL :: Vy ! vertical velocity
REAL :: X ! horizontal displacement
REAL :: Y ! vertical displacement
```

```
READ(*,*) Angle, Time, U
Angle = Angle * PI / 180.0 ! convert to radian
X = U * COS(Angle) * Time
Y = U * SIN(Angle) * Time - g*Time*Time / 2.0
Vx = U * COS(Angle)
Vy = U * SIN(Angle) - g * Time
V = SQRT(Vx*Vx + Vy*Vy)
Theta = ATAN(Vy/Vx) * 180.0 / PI ! convert to degree
WRITE(*,*) 'Horizontal displacement : ', X
WRITE(*,*) 'Vertical displacement : ', Y
WRITE(*,*) 'Resultant velocity : ', V
WRITE(*,*) 'Direction (in degree) : ', Theta
```

```
END PROGRAM Projectile
```

MAKEFILE

```
#
# TX7
#
FTRACE=
OPENMP=
F90=ifort $(FTRACE) $(OPENMP)
NOASSUME=
LOADFLAG=-static
#
# Executable
#
EXEC=/gfs/dk12/pkubota/MPIGlobal/model/exec/ParModel_MPI
OBJ= \
    Constants.o \
    Model.o
model: $(OBJ)
    $(F90) -o $(EXEC) $(LOADFLAG) $(OBJ)
Constants.o : Constants.f90
    $(F90) -c Constants.f90
Model.o : Model.f90 Constants.o
    $(F90) -c $(NOASSUME) Model.f90
.SUFFIXES:
.SUFFIXES: .f90 .o

.f90.o :
    $(F90) -c $<

clean:
    -rm -f $(OBJ)
    -rm -f $(EXEC)
    -rm -f *.mod
```

ftp://ftp1.cptec.inpe.br/pkubota/curso_fortran/Constants.f90
ftp://ftp1.cptec.inpe.br/pkubota/curso_fortran/Model.f90
ftp://ftp1.cptec.inpe.br/pkubota/curso_fortran/Makefile