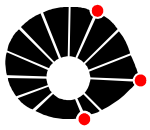


Universidade Estadual de Campinas

Centro Nacional de Processamento de Alto Desempenho
São Paulo

Apostila de Treinamento:

Introdução ao Fortran90

Revisão: 2009

ÍNDICE

Tipografia utilizada na apostila

Na apresentação de alguns comandos do Fortran, foram utilizados símbolos gráficos que identificam, na sintaxe do comando, a característica de ser opcional ou obrigatório:

< *característica* > É **obrigatório** a informação no comando;

[*característica* **]** É **opcional** a informação no comando.

Exemplo: Utilização do comando IF/THEN/ELSE/ENDIF

IF <(expressão lógica**)> THEN**
<bloco de comandos>
...
[ELSE
<bloco de comandos>
...]
END IF

1. A *expressão lógica* do IF é **obrigatória**;
2. O *bloco de comandos* após o comando THEN, é **obrigatório**;
3. O comando ELSE é **opcional**, mas se for utilizado, o *bloco de comandos* após o ELSE, passa a ser **obrigatório**.

```
IF  ( X = 0 ) THEN
     PRINT *, “X=0”
     X=X+1
     Y=10**X
ELSE
     PRINT *, “X é diferente de 0”
     Y=10**X
END IF
```

1 – INTRODUÇÃO

1.1 – História

- **FOR**mula **TRAN**slation System;

  Primeira linguagem considerada de alto nível (“High Level” – próxima a linguagem humana), desenvolvida por **John Backus**, na IBM, em 1954, e comercializada em 1957. Continua sendo muito utilizada nos dias de hoje nas áreas de programação cientifica e aplicações matemáticas. O Fortran, inicialmente, era um interpretador de código digital para o computador IBM 701, sendo originalmente chamado de “Speedcoding”.

  A **primeira geração** de códigos para programação de computadores, era designada de linguagem de máquina ou código de máquina, que na verdade, é a única linguagem que o computador interpreta. São instruções codificadas em seqüências de 0s e 1s (Seqüência binária).

  A **segunda geração** de códigos foi chamada de linguagem “Assembly” (montagem). Esta linguagem torna a seqüência de 0s e 1s em palavras compreensíveis, como “ADD”, “STORE”, “LOAD”. Na verdade, esta linguagem é traduzida para código de máquina por programas chamados “Assemblers” (montadoras).

  A **terceira geração** de códigos foi chamada de “High level language” ou HLL, na qual existem palavras e sintaxe de acordo com a linguagem humana (como palavras em uma sentença). Para que um computador entenda uma HLL, é necessário um compilador que traduza o código para “Assembly” ou para código de máquina.

  John Backus chefiou a equipe de pesquisadores da IBM que inventou o Fortran, no “Watson Scientific Laboratory – NY”. Esta equipe não inventou a idéia de HLL ou a idéia de compilar uma linguagem para código de máquina, mas o Fortran foi uma das primeiras HLL.

  O Fortran está com mais de 40 anos, sendo constantemente modificado e atualizado, e se mantêm no topo das linguagens para computadores nas áreas de programação científica e industrial.

- **Fortran I** (1954-1957)

  O compilador Fortran I manteve o recorde de traduzir um código, por mais de 20 anos.

- **Fortran II** (1958)

  Capacidade de compilar módulos de programas, não executáveis, para serem “link editados” com outros programas.

- **Fortran III** (1958) - Não saiu do laboratório.

- **Fortran IV** (1961) ou **Fortran66** (1966)

  Implementação dos comandos COMMON e EQUIVALENCE, que permitiram o compartilhamento de código entre outros programas e sub-rotinas. Foi o primeiro compilador oficialmente padronizado.

- **Fortran77** (1977)

  Foi padronizado utilizando o conceito de programação estruturada. Padrão: ANSI X3 e ISO/IECJTC1/SC22/WG5

- **Fortran90** (1980)

  Atualização do Fortran77 que levou 12 anos para ser efetuada. Vários recursos do Fortran90 se aproximam aos existentes na linguagem C (Alocação dinâmica de memória, apontadores e orientação ao objeto).

- **HPF** (1990) - High Performance Fortran – Fortran90 para ambientes com memória distribuída;

- **Fortran95** (1995)

1.2 – Fortran77

Algumas necessidades em programação, definiram o Fortran77 como uma linguagem obsoleta em relação às linguagens atuais:

- Formato fixo:

    - Linhas de instruções, começavam na posição 7 e iam até a 72;
    - Somente letras maiúsculas;
    - Nomes de variáveis, até 6 caracteres.

- Impossibilidade de determinar operações paralelas;

  É uma situação crítica, pois o Fortran é considerado com uma linguagem de alta performance, no entanto, até o padrão 77 não existia nenhuma instrução que permitisse ao usuário definir regiões do programa que pudessem executar em paralelo, utilizando os recursos de mais de um processador, como por exemplo, o compartilhamento de endereços de memória.

- Não é possível a alocação dinâmica de memória;

  No Fortran77, o programador é obrigado a declarar vetores com o maior tamanho possível para reservar memória durante a compilação, ou seja, não era possível utilizar conjuntos de dados temporários durante a execução.

- Não possui representação numérica portável;

  Diversos ambientes computacionais criaram extensões do Frotran77 para melhorar a precisão numérica em suas arquiteturas, o que tornava o código “não portável” para outra arquitetura. No Fortran90, as diversas idéias de precisão numérica foram padronizadas, melhorando a portabilidade.

- Não era possível definir uma nova estrutura de dados pelo usuário.

- Não possui recursão explícita;

  Não era possível chamar uma função dentro de outra função. Não havia recursão!

1.3 – Fortran90

- Formato livre:

    - 132 caracteres por linha;
    - Maiúsculas e minúsculas;
    - Nomes até 31 caracteres;
    - Mais de um comando por linha.

- Novas instruções que permitem execuções de partes do programa em paralelo.

    - PARALLEL DO (Definição de "loops" paralelos);
    - PARALLEL SECTION;
    - Operações de "ARRAYS" em paralelo (SUM, MAX, etc). O "ARRAY" é dividido em diversos vetores, para ser distribuído por entre diversos processos.

- Novas instruções que permitem a alocação dinâmica de memória (ALLOCATABLE, ALLOCATE);

- Possibilidade de definição de novas estruturas de dados (Comando de declaração TYPE);

```
TYPE COORDS_3D
       REAL :: x, y, z
END TYPE COORDS_3D
TYPE(COORDS_3D) :: pt1, pt2
```

- Maior flexibilidade e portabilidade na definição da precisão numérica de uma variável (Parâmetro KIND);

```
INTEGER x           FORTRAN IBM → Precisão de 4 bytes
                    FORTRAN CRAY → Precisão de 8 bytes

INTEGER(KIND=4)  x
```

- Recursividade de funções e rotinas;

- Estruturas de controle:

    - DO...ENDDO
    - DO…WHILE
    - SELECT CASE
    - EXIT
    - CYCLE
    - IF…THEN…ELSE…ENDIF

- Nova orientação - Desligar a definição automática de variáveis como reais ou inteiras.

    - IMPLICIT NONE

1.4 – Exemplo de Programa Fortran90

```
MODULE Triangle_Operations
  IMPLICIT NONE
CONTAINS
FUNCTION Area(x,y,z)
  REAL :: Area ! function type
  REAL, INTENT( IN ) :: x, y, z
  REAL :: theta, height
  theta = ACOS((x**2+y**2-z**2)/(2.0*x*y))
  height = x*SIN(theta); Area = 0.5*y*height
END FUNCTION Area
END MODULE Triangle_Operations


PROGRAM Triangle
  USE Triangle_Operations
  IMPLICIT NONE
  REAL :: a, b, c, Area
  PRINT *, 'Welcome, please enter the&
          &lengths of the 3 sides.'
  READ *, a, b, c
  PRINT *, 'Triangle''s area: ', Area(a,b,c)
END PROGRAM Triangle
```

- Possui duas estruturas principais de programação FORTRAN: MODULE e PROGRAM;
    - MODULE é muito utilizado para centralizar funções e rotinas do usuário que serão utilizadas por diversos programas;
    - PROGRAM é a estrutura principal de qualquer programa FORTRAN.

- Uma estrutura auxiliar: FUNCTION, que pode ser definida dentro da estrutura MODULE ou PROGRAM;

- Comando opcional de declaração IMPLICIT NONE – utilizado uma única vez para cada estrutura principal, para desligar a definição automática das variáveis;

- Comando de declaração USE – Especifica a utilização de rotinas ou funções definidas por uma estrutura MODULE

- Comando de declaração REAL – especifica as variáveis reais do programa;

- Comando de execução PRINT – Imprime os resultados de variáveis em uma determinada saída;

- Comando de execução READ – Lê valores de uma determinada entrada e os atribui às variáveis especificadas;

2 – ELEMENTOS DO FORTRAN90

2.1 – Características de Codificação

O Fortran90 suporta o formato livre (“Free form”) de codificação em relação ao formato fixo (“Fixed form”) do Fortran77, o que facilitou a programação em Fortran. De qualquer forma, o Fortran90 aceita todas as características e sintaxe do Fortran77. As principais características são:

- 132 caracteres por linha;

- Alfanumérico: a-z, A-Z, 0-9, _ Os comandos e nomes de variáveis podem utilizar letras maiúsculas ou minúsculas, misturadas ou não;

```
IMPLICIT NONE = implicit none = ImPlicit noNe
```

- **!** Caractere de início de comentário. Pode ser colocado em qualquer posição da linha, sendo que, tudo que estiver a direita do caractere será considerado comentário;

- **&** Caractere de continuação de linha. Colocado no final da linha, indica que o comando continua na próxima linha. Em caso de continuação de “strings”, esse caractere pode ser utilizado na próxima linha para indicar a posição exata da continuação do “string” e evitar brancos desnecessários;

```
PRINT *, “Hoje é o primeiro dia do curso de &
        &Introdução ao Fortran90”
```

- **;** Caractere de separação de comandos. Vários comandos podem estar na mesma linha;

```
PROGRAM Teste; REAL a; END PROGRAM
```

- Símbolos aritméticos:

+	Adição
-	Subtração
*****	Multiplicação
/	Divisão
******	Potenciação

2.2 – Regras de Codificação

- **“Brancos”** não são permitidos:
  - “Palavras-chave” (Comandos, parâmetros, opções)

```
INTEGER :: nome1          Certo
INT EGER :: nome1         Errado
```

  - “Nomes” (Varáveis e arquivos)

```
REAL :: valor_total       Certo
REAL :: valor total       Errado
```

- **“Brancos”** são permitidos:
  - Entre “palavras-chave”
  - Entre “nomes” e “palavras-chave”

```
INTEGER FUNCTION val(x)       Certo
INTEGERFUNCTION val(x)        Errado
INTEGER FUNCTIONval(x)        Errado
```

- Nomes de variáveis e rotinas:
  - Podem possuir até 31 caracteres

```
INTEGER :: essa_variável_possui_mais_de_31_letras   Errado
```

  - Devem começar com letra. Maiúsculas ou minúsculas são equivalentes

```
REAL  ::  a1        Certo
REAL  ::  1a        Errado
```

- Podem continuar com letras, dígitos ou “_”

```
CHARACTER  ::  atoz Certo
CHARACTER  ::  a-z  Errado
CHARACTER  ::  a_z  Certo
```

- Uso de Comentários

```
PROGRAM Teste
!
! Definição das variáveis
!
REAL :: aux1 ! Variável auxiliar
```

2.3 – Compilação no Ambiente do CENAPAD-SP

O Ambiente do CENAPAD-SP possui dois ambientes principais para programação e execução de programas, cada um com seus próprios compiladores e ferramentas de execução:

- Ambiente IBM/AIX

  - **Compilador Fortran77**: xlf, f77, fort77, g77 extensão: .f , .F
  - **Compilador Fortran90**: xlf90, f90 extensão: .f , .f90
  - **Fortran95**: xlf95 extensão: .f , .f95

  Compilação **xlf90 <opções> <nome do programa com extensão>**

- Ambiente INTEL/Linux

  - **Compilador Fortran77 ou 90**: ifort extensão: .f , .F , .f90

  Compilação **ifort <opções> <nome do programa com extensão>**

- Opções genéricas de compilação:

-o	Especifica o nome do arquivo executável (Default: **a.out**);
-O, -O1, -O2, -O3	Nível de otimização do código;
-c	Não gera o arquivo executável, somente o arquivo objeto;
-g	Gera um arquivo executável com informações para depuração;
-L<path>	Localização das bibliotecas para serem integradas ao arquivo (“linkedição”);
-l<biblioteca>	Especifica uma determinada biblioteca;

- Exemplos de compilações:

```
xlf90   cofigo.f   -o   teste   -O3
```

Arquivo executável: teste

```
xlf90   cena.f    -c    -L/usr/lib/scalapack  -lscalapack
```

Arquivo objeto: cena.o

```
ifort   salto.f   -o  salto  -O3   -L/home/kusel   -lbib1
```

Arquivo executável: salto

```
ifort   parceiro.f  -o  par  -g  -O
```

Arquivo executável para ser depurado: par

EXERCÍCIO 1- Compilação e Execução

1. Caminhe para o diretório ~/curso/fortran/ex1.Utilizando um editor de texto, edite o programa que soluciona uma equação do 2º grau (**quadsol.f90)**

```
cd  ~/curso/fortran/ex1
nano  quadsol.f90            Editores: vi, pico ou emacs
```

```
PROGRAM QES
   IMPLICIT NONE
   INTEGER :: a, b, c, D
   REAL :: Part_Real, Part_imag
   PRINT*, ''Entre com os valores de a, b, c''
   READ*, a, b, c
   IF (a /= 0) THEN
      D = b*b - 4*a*c                                  !Calculo do discriminante
      IF (D == 0) THEN                                 !Uma raiz
          PRINT*, ''Raiz é '', -b/(2.0*a)
      ELSE IF (D > 0) THEN                             !Raizes reais
          PRINT*, ''Raizes são '',(-b+SQRT(REAL(D)))/(2.0*a),&
                          ''e '',  (-b-SQRT(REAL(D)))/(2.0*a)
      ELSE                                             !Raizes complexas
          Part_Real = -b/(2.0*a)
          Part_Imag = (SQRT(REAL(-D))/(2.0*a))
          PRINT*, ''1a. Raiz'', Prat_Real, ''+'', Part_Imag, ''i''
          PRINT*, ''2a. Raiz'', Part_Real, ''-'', Part_Imag, ''i''
      END IF
   ELSE                                                ! a == 0
      PRINT*, ''Não é uma equação quadrática''
   END IF
   END PROGRAM QES
```

2. Compile e execute o programa. Verifique se ele executa corretamente para os valores abaixo:

```
ifort  quadsol.f90  –o  quadsol  –O3
./quadsol

(a) a = 1 b = -3 c = 2
(b) a = 1 b = -2 c = 1
(c) a = 1 b = 1  c = 1
(d) a = 0 b = 2  c = 3
```

3. Copie quadSol.f para novoquadsol.f90.

```
cp quadsol.f90 novoquadsol.f90
```

4. Edite esse novo arquivo e declare uma nova variável real de nome “parte2a”.

5. Na seção executável do código, defina a nova variável igual ao valor de 1/(2.0*a).

```
parte2a=1/(2.0*a)
```

6. Aonde aparecer a expressão 1/(2.0*a), substitua pela nova variável.

EXERCÍCIO 2- Reestruturação de programa

1. Caminhe para o diretório ~/curso/fortran/ex2. Reescreva o programa **basic_reform.f90** de uma maneira que fique mais compreensível. O Programa transforma temperaturas em graus Fahrenheits (ºF) para graus Celsius (ºC).

```
cd  ~/curso/fortran/ex2
nano  basic_reform.f90        Editores: vi, pico ou emacs
```

```
PROGRAM MAIN;INTEGER::degreesfahrenheit&
  ,degreescentigrade;READ*,&
  degreesfahrenheit;degreescentigrade&
  =5*(degreesfahrenheit-32)/9;PRINT*,&
  degreesCENtiGrAde;END
```

2. Compile e execute o programa.

```
ifort  basic_reform.f90  –o  basic  –O3
./basic
```

2.4 – Estrutura e Organização de Programas

O Fortran possui algumas regras bem definidas para a estruturação e organização dos programas e estabelece a seguinte ordem:

1. **Cabeçalho de definição:**

```
PROGRAM,  FUNCTION,  SUBROUTINE,  MODULE,  DATA
```

- É necessário definir a estrutura que será codificada;
- Não é obrigatório especificar o programa principal com o comando PROGRAM, mas é recomendado;
- Só pode haver uma estrutura PROGRAM;
- Pode haver mais de uma estrutura FUNCTION, SUBROUTINE e MODULE;
- Toda estrutura deve finalizar com o comando END.

2. **Comandos de Declaração:**

```
IMPLICIT NONE, REAL, INTEGER, CHARACTER, COMPLEX, PARAMETER, DATA
```

- É recomendado que se defina todas as variáveis que serão usadas no programa e, se for necessário, as suas características iniciais;
- É recomendado que use o comando IMPLICIT NONE para desligar a definição automática das variáveis.

3. **Comandos de Execução:**

```
READ, PRINT, FORMAT, IF-ENDIF, DO-ENDDO,  comando de atribuição
```

- Comandos que definem a seqüência lógica de execução do programa.

2.5 – Tipo de Dado

Todo tipo de dado possui um nome, um conjunto válido de valores, um significado dos valores e um conjunto de operadores. No Fortran90 existem três classes de objetos de dados e cada classe irá definir um ou mais tipos de dados.

- **Dado Caractere**

  Para a definição de variáveis caracteres, com um ou mais valores alfanuméricos.

```
CHARACTER :: sexo         ! Variável com 1 caractere
CHARACTER(LEN=12) :: nome ! Variável com 12 caracteres
```

- **Dado Lógico "Boolean"**

  Para a definição de variáveis lógicas, que só podem assumir os valores: Verdadeiro (True) ou Falso (False).

```
LOGICAL :: solteira       ! Verdadeiro ou Falso ?
```

- **Dado Numérico**

  Para a definição de variáveis numéricas com valores inteiros, reais ou complexos.

```
REAL :: pi        ! Valor numérico com casas decimais
INTEGER :: idade  ! Valor numérico inteiro
COMPLEX :: a      ! Valor numérico com parte real e parte imaginária
                  ! (x + iy)
```

2.6 – Constantes

- Constante é um objeto com valor fixo

```
+12345              Valor numérico Inteiro
1.0                 Valor numérico Real
-6.6E-06            Valor numérico Real
.FALSE.             Valor Lógico
"Curso Fortran"     Valor Caractere
```

- Observações:

  - Números **Reais** possuem ponto decimal e/ou o símbolo expoente e/ou sinais + ou -;
  - Números **Inteiros** não possuem ponto decimal e são representados por uma seqüência de dígitos com o sinal + ou -;
  - Só existem dois valores **Lógicos**: **.FALSE.** e **.TRUE. ;**
  - Valores **caracteres** são delimitados por **"** ou **'**

2.7 – Tipo de Dado Implícito

Se num programa, uma variável for referenciada sem ter sido declarada, então o tipo implícito será atribuído (REAL ou INTEGER). É uma característica automática de toda implementação Fortran. A atribuição se baseia na primeira letra do nome de uma variável:

- I, J, K, L, M ou N, será definida como **Inteiro (INTEGER)**;
- Qualquer outra letra será definida como **Real (REAL)**;

Tipo de dado implícito é potencialmente perigoso e deve ser evitado, podendo ser modificado ou desligado (Fortran90), obrigando ao usuário definir todas as variáveis que serão utilizadas no programa. Para isso, utiliza-se o comando:

```
IMPLICIT <tipo> ( <lista de letras>, ... )
```

```
     IMPLICIT REAL(A-H,O-Z),INTEGER(I-N)
     IMPLICIT NONE        ! Definição implícita desligada
```

Exemplo de problema ocorrido (“bug”) em um programa Fortran77:

```
           DO 30 I = 1.1000
              ...
        30 CONTINUE
```

- O “loop” no Fortran77 utiliza a **,** para indicar o intervalo de valores;
- Neste caso, devido ao **.** e devido a definição automática de variáveis, o compilador Fortran77 entendeu como a definição da variável real DO30I como o valor 1.1, ao invés de executar o conteúdo do “loop” 1000 vezes.

O Comando IMPLICIT deve ser utilizado logo após a definição da estrutura do programa.

```
PROGRAM teste
     IMPLICIT NONE
          ...
```

2.8 – Declaração de Variáveis

```
<tipo> [(opções,...), <atributos,...> ::] <variáveis> [=valor]

     <tipo>                REAL, INTEGER, COMPLEX, LOGICAL, CHARACTER

     <opções>              LEN=,KIND=

     <atributos>           PARAMETER,  DIMENSION, ALLOCATABLE, SAVE, INTENT;

     ::                    Não é obrigatório, a menos que se especifique um
                           atributo ou um valor inicial para a variável;

     <variáveis>           Nome das variáveis, separadas por “,”

     =valor                Valor inicial da variável
```

- Opções

  LEN= É utilizada para definir o tamanho de variáveis do tipo caractere (Default=1 byte ou 1caractere);

  KIND= É utilizada para definir a precisão numérica de variáveis do tipo inteiro ou real (Default=4 bytes ou precisão simples);

  OBS: A notação utilizando o ***** ,padrão do Fortran77, para indicar o tamanho das variáveis, ainda é aceito pelo Fortran90.

- Atributos

  PARAMETER Determina que um valor seja fixado para a variável;

  DIMENSION Determina a que a variável irá compor um conjunto de dados, em várias dimensões, com vários elementos em cada dimensão;

  ALLOCATABLE Determina que o número de elementos de uma dimensão da variável será alocado durante a execução do programa;

  SAVE Determina que o tipo e valor de uma variável seja global, estático, visível por todo o programa;

  INTENT Utilizado na declaração de variáveis em funções e subrotinas, serve para identificar se a variável é de entrada ou saída.

- Exemplos:

```
-  Sintaxes alternativas:

REAL :: x  ou  REAL x  ou  REAL*4 x  ou  REAL(4) x  ou  REAL(KIND=4) :: x

REAL(KIND=8) y  ou  REAL(8) y  ou DOUBLE PRECISION y

REAL, DIMENSION(10,10) :: y, z  ou  REAL y(10,10), z(10,10)

INTEGER i,j ou INTEGER(KIND=4) i,j

CHARACTER(LEN=10) :: nome  ou  CHARACTER*10 nome

-  Outros exemplos:

INTEGER :: k=4

LOGICAL ptr

CHARACTER sexo

CHARACTER(LEN=32) :: str

CHARACTER(LEN=10), DIMENSION(10,10) ::  vetor
CHARACTER(LEN=8) :: cidade = "Campinas"

INTEGER :: pi = +22/7

LOGICAL :: wibble = .TRUE.

REAL :: a = 1., b = 2

CHARACTER*6 :: you_know = "y'know"

INTEGER :: a, b, c, d

LOGICAL, DIMENSION(2) :: bool
```

Observações:

- O tipo DOUBLE PRECISION pode ser substituído por REAL(8)
- Só existem dois valores para o tipo LOGICAL, que devem vir entre pontos: .true. ou .false.
- As sintaxes alternativas de declaração de variáveis são devido a necessidade de manter a compatibilidade com as versões mais antigas do Fortran.

2.9 – Declaração de Constantes

Constantes simbólicas, que não podem mudar de valor durante a execução de um programa são definidas pelo comando **PARAMETER** (Fortran77) ou, na declaração da variável, utilizando o atributo **PARAMETER** (Fortran90).

- No comando PARAMETER, a variável deve ter sido declarada antes de receber o valor constante;

```
INTEGER pre

PARAMETER (pre=252)
```

- No atributo PARAMETER, a variável é declarada e ao mesmo tempo, recebe um valor constante;

```
REAL, PARAMETER  ::  pi=3.14159
```

- Na declaração de constantes caracteres, pode-se utilizar a opção LEN=* em conjunto com o atributo PARAMETER, indicando que o valor constante atribuído a variável, determinará o tamanho dessa variável caractere;

```
CHARACTER(LEN=*), PARAMETER :: n1='Paulo', n2='Francisco'

               n1   Variável caractere de 5 caracteres
               n2   Variável caractere de 9 caracteres
```

- Recomenda-se utilizar o atributo PARAMETER;

- O atributo PARAMETER determina que o valor da variável não poderá mudar durante a execução do programa.

2.10 – Inicialização de Variáveis

- Quando um programa é executado, o conteúdo das variáveis declaradas, são normalmente indefinidos, mas na maioria das extensões Fortran, assume o valor zero. É possível, e recomendado, atribuir um valor inicial a uma variável e altera-la no decorrer da execução do programa.

```
REAL :: x=1.0E02, y=1.005

INTEGER :: i=5, j=100

CHARACTER(LEN=5) :: luz='Amber'

LOGICAL :: a=.TRUE., b=.FALSE.
```

- É possível declarar uma variável como sendo o resultado de uma **expressão aritmética** entre outras variáveis que já foram declaradas e iniciadas, mas com o atributo PARAMETER.

- No geral, as funções internas do Fortran, **não podem** ser utilizadas em expressões que iniciam uma variável, mas isso depende da implementação do Fortran. No ambiente CENAPAD-SP, o compilador Fortran90 da IBM (xlf90), não permite o uso de funções, já o compilador Fortran90 da Intel (ifort), permite o uso de funções.

```
REAL, PARAMETER :: pi=3.14159

REAL, PARAMETER :: radius=3.5

REAL :: circo=2*pi*radius  ! expressão

REAL :: pi_2=4.*atan(1.0)  ! Uso da função arcotangente.
                           ! Depende do compilador
```

EXERCÍCIO 3 – Erro na declaração de variáveis

1. Caminhe para o diretório ~/curso/fortran/ex3, tente compilar o programa **declara.f90 .** Verifique e corrija os erros na declaração das variáveis. Analise com atenção as mensagens de erro.

```
cd  ~/curso/fortran/ex3

ifort declara.f90 –o decl
./decl
```

2. Edite, corrija os erros e tente compilá-lo novamente. Verifique e tente entender os resultados.

```
nano declara.f90          Editores: vi, pico ou emacs
```

EXERCÍCIO 4 - Declaração de Variáveis

1. Caminhe para o diretório ~/curso/fortran/ex4, crie um programa em fortran90 (**variavel.f90**) que apenas declare as seguintes variáveis da tabela abaixo:

```
cd  ~/curso/fortran/ex4
nano variavel.f90                Editores: vi, pico ou emacs
```

Nome da Variável	Status	Tipo	Valor Inicial
pe	Variável	Inteiro	-
milhas	Variável	Real	-
cidade	Variável	Caractere (20 letras)	-
local	Constante,fixa	Caractere	Campinas
E_aonde_nasceu	Constante	Lógica	Falso
seno_meio	Constante	Real	Sin(0.5)

2. Compile e execute o programa. Verifique se há erros de compilação.

```
ifort variavel.f90  –o  var
./var
```

2.11 – Expressões

- Expressões são construídas com pelo menos um operador (**+ , - , * ,** / **,** // **, ** ,etc.**) e, com pelo menos um operando.

```
X+1                 Expressão numérica (Adição)

“campo”//campo      Expressão caractere (Concatenação)

A .GT. B            Expressão lógica
```

- O tipo de uma expressão deriva do tipo dos operandos;

- Operandos podem ser: expressões, números, caracteres, funções;

2.11.1 – Operador de ATRIBUIÇÃO (=)

- Normalmente uma expressão é utilizada em conjunto com um operador de atribuição **“=“**, que irá definir ou atribuir um valor a um novo objeto.

```
a = b

c = SIN(0.7)*12.7

nome = iniciais//sobrenome

logi = (a.EQ.b.OR.c.NE.d)
```

OBS: Os operandos a esquerda e a direita do sinal de igualdade não necessitam ser do mesmo tipo.

2.11.2 – Operadores NUMÉRICOS

- Exponencial (**) (Avaliado da direita para esquerda)

```
10**2
a**b
```

- Multiplicação (*) e Divisão (/) (Avaliado da esquerda para direita)

```
10*7/4
a*b/c
```

- Adição (+) e Subtração (-) (Avaliado da esquerda para direita)

```
7+8-3
a+b-c
```

OBS: Os operandos podem ser: constantes, variáveis escalares ou vetoriais, com exceção do expoente que necessariamente deve ser escalar.

2.11.3 – Operadores RELACIONAIS

- São utilizados em expressões lógicas, entre dois operandos, retornando um valor lógico (.TRUE. ou .FALSE.) .
- Os operadores relacionais, como mnemônicos, são sempre utilizados entre dois pontos, maiúsculo ou minúsculo.

.GT.	>	Maior que
.GE.	>=	Maior igual
.LE.	<=	Menor igual
.LT.	<	Menor que
.NE.	/=	Não é igual a
.EQ.	==	Igual a

```
Para i=20 e j=10, então

a = i  .GT.  j      a=.true.

b = i  .EQ.  j      b=.false.
```

2.11.4 - Operadores LÓGICOS

- São utilizados em expressões lógicas, com um ou dois operandos, retornando um valor lógico (.TRUE. ou .FALSE.) :

.AND. → .TRUE. Se ambos os operandos forem .TRUE.
.OR. → .TRUE. Se pelo menos um operando for .TRUE.
.NOT. → .TRUE. Se o operando for .FALSE.
.EQV. → .TRUE. Se os operandos possuírem o mesmo valor
.NEQV.→ .TRUE. Se os operandos possuírem valores diferentes

Se **T=.TRUE.** e **F=.FALSE.** então

```
T .AND. F → .FALSE.      F .AND. F → .FALSE.
T .OR.  F → .TRUE.       F .OR.  F → .FALSE.
T .EQV. F → .FALSE.      T .NEQV.F → .TRUE.
```

2.11.5 – Operador CARACTERE (//)

- Uma variável caractere, pode ser representada como um vetor de caracteres, podendo ser manipulada conforme a sintaxe de vetores.

vetor(posição inicial **:** posição final)

```
CHARACTER(LEN=*), PARAMETER :: string= 'abcdefgh'

string(1:1) → 'a'
string(2:4) → 'bcd'
string(6:7) → 'fg'
string(1: ) → 'abcdefgh'
string( :1) → 'a'
```

- Utilizado para efetuar a concatenação **"//"**, somente de variáveis caracteres.

```
CHARACTER(LEN=*), PARAMETER :: string='abcdefgh'

a=string//string(3:5)  →  'abcdefghcde'
b=string(1:1)//string(2:4)  →  'abcd'
estado='São Paulo'
cidade='Campinas'
endereco=estado//"-"//cidade//"-"//"Brasil" → 'São Paulo-Campinas-Brasil'
```

2.11.6 – Precedência de Operadores

( ) ** *,/ +,- //

- Toda expressão que vier entre parêntesis, será avaliada primeiro;
- Em expressões aritméticas, **com o mesmo nível de avaliação**, o que vier da esquerda para direita, será avaliado primeiro, com exceção do expoente.

```
(a + b)/c            diferente de    a+b/c
(a*b)/c              igual a         a*b/c
a/b*c                diferente de    a/(b*c)
x=a+b/5.0-c**d+1*e   equivale a   x=((a+(b/5.0))-(c**d))+(1*e)
```

OBS: Dois operadores não podem ficar adjacentes.

```
1*-1      Errado
1*(-1)    Correto
```

EXERCÍCIO 5 – Expressões

1. Caminhe para o diretório ~/curso/fortran/ex5. Edite o programa **area_circulo.f90**

```
cd  ~/curso/fortran/ex5
nano area_circulo.f90            Editores: vi, pico ou emacs
```

2. O programa está incompleto. Acrescente na linha das reticências o que é solicitado.

   ...Declaração de variáveis...
   ...Expressão para cálculo da área e volume...

3. Fórmulas para serem codificadas:

   Área do Círculo: **area = πr^2**

   Volume da esfera: **volume = $\frac{4\pi r^3}{3}$**

4. Compile e execute o programa.

```
ifort area_circulo.f90  –o  area
./area
```

5. Verifique se ele executa corretamente para os valores: **2, 5, 10, –1**

3 – COMANDOS DE CONTROLE DO FLUXO DA EXECUÇÃO

Toda linguagem de programação estruturada necessita de artifícios que possibilitem o controle da execução de comandos. Comandos que podem alterar o fluxo de execução de um programa, repetir determinadas tarefas e direcionar a entrada e saída dos dados.

- Comandos de execução condicional: **IF... , IF...THEN...ENDIF , IF...THEN...ELSE...ENDIF, IF...THEN...ELSEIF...ENDIF**

  O comando IF analisa uma expressão que, se o resultado for verdadeiro, executa os comandos que vierem após o THEN, se for falso , executa os comandos que vierem após o ELSE. O laço da condição finaliza com o comando ENDIF;

- Comandos de iteração repetitiva: **DO...ENDDO, DO WHILE...ENDDO**

  O comando DO permite a execução repetitiva de um bloco de comandos (“loops”). O número de iterações pode ser uma constante, variável ou uma expressão, desde que resultem para um valor constante. O Fortran aceita “loops” encadeados (um “loop” dentro de outro “loop”);

- Comandos de múltipla escolha: **SELECT CASE**

  O comando SELECT permite a execução de comandos baseado no valor que uma expressão pode assumir. É uma construção equivalente ao IF...THEN...ELSE...ENDIF;

- Comando de salto: **GOTO**

  Direciona a execução do programa para uma linha de comando identificada por um número. Bastante poderoso, mas deve ser evitado ou utilizado com cuidado em situações que necessitam uma ação de emergência;

- Exceções de comandos de I/O: **ERR=, END=, EOR=**

  São opções de comandos de I/O que funcionam como o comando GOTO, em determinadas situações de emergência (erro geral de I/O, erro de fim de arquivo encontrado e erro de fim de registro encontrado), direciona a solução do problema para um determinada posição do programa;

3.1 – Comando IF

- É a forma mais básica de execução condicional; determina a execução de um **único comando**, se uma expressão lógica for verdadeira, caso contrário a execução passa para a próxima linha.

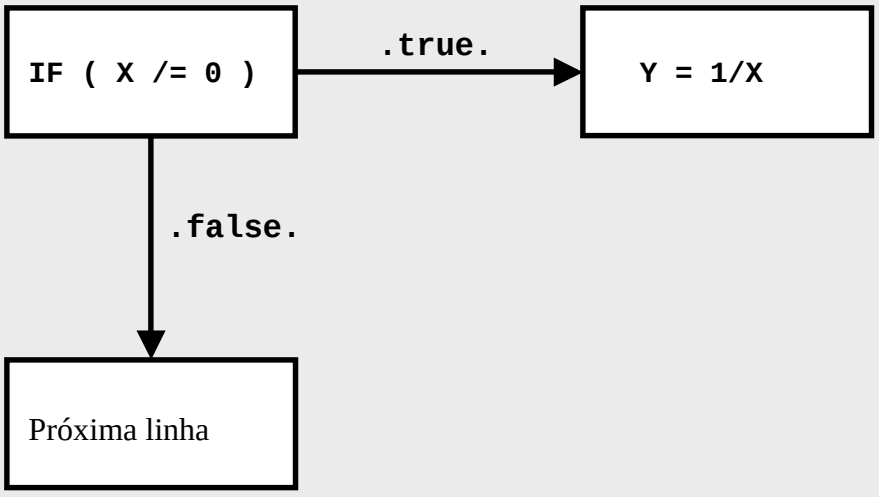
IF <**(**expressão lógica**)**> <**comando**>

Exemplos:

```
IF (cidade .eq. “UBATUBA”)  estado=”SAO PAULO”

IF (x > 20) y=10

IF (I  .NE. 0 .AND. J .NE. 0) K=1/(I*J)

IF (I /=0 .and. j /=0) k=1/(i*j)

IF (a.eq.b.and.c.ne.a) d=b
```

Diagrama de execução:

```
IF ( X /= 0) Y=1/X
```



- Importante:
  - Somente um comando pode ser executado se a condição for verdadeira;
  - O comando deve vir na mesma linha do comando IF.

3.2 – Comando IF...THEN...END IF

- Determina a execução de um bloco de comandos se uma condição lógica for verdadeira.

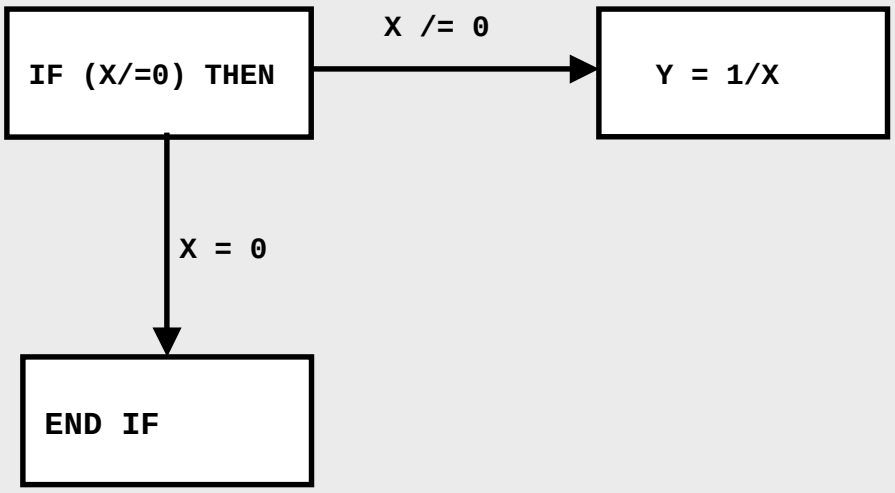
IF <(expressão lógica**)> THEN**
<bloco de comandos>
...
END IF

Exemplo:

```
IF  ( X == 0 ) THEN
     PRINT *, “X=0”
     X=X+1
     Y=10**X
END IF
```

Diagrama de execução:

```
  IF ( X /= 0) THEN
     Y=1/X
  END IF
```



- Importante:

  - O bloco de comandos deve iniciar na linha seguinte ao do comando IF...THEN;
  - O comando IF...THEN, finaliza com o comando END IF ou ENDIF.

3.3 – Comando IF...THEN...ELSE...END IF

- Determina a execução de um bloco de comandos se uma condição lógica for verdadeira ou falsa. No entanto, o bloco de comandos para condição falsa, é opcional.

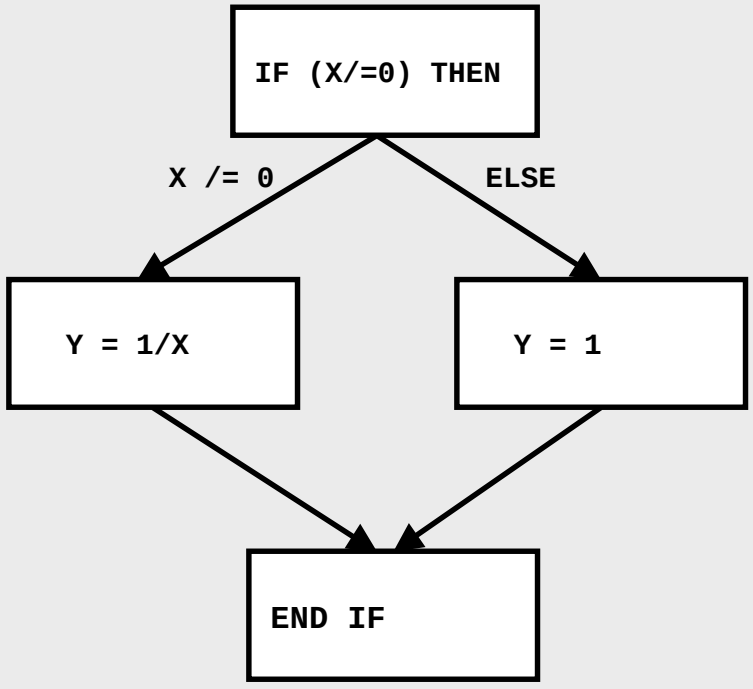
IF <(expressão lógica**)> THEN**
<bloco de comandos>
...
[ELSE
<bloco de comandos>
...]
END IF

Exemplo:

```
IF  ( X == 0 ) THEN
     PRINT *, "X=0"
     X=X+1
     Y=10**X
ELSE
     PRINT *, "X é diferente de 0"
     Y=10**X
END IF
```

Diagrama de execução:

```
IF ( X /= 0) THEN
   Y=1/X
ELSE
     Y=1
END IF
```



- Importante:

  - O bloco de comandos deve iniciar na linha seguinte ao do comando IF...THEN e na linha seguinte do comando ELSE;
  - O bloco de comandos do THEN, finaliza com o comando ELSE.
  - O comando IF...THEN...ELSE, finaliza com o comando END IF ou ENDIF.

3.4 – Comando IF...THEN...ELSEIF...END IF

- Determina a execução recursiva de vários comandos IFs dentro da condição lógica do IF principal, através do comando ELSEIF.

IF <(expressão lógica**)> THEN**
<bloco de comando>
...
[ELSEIF <(expressão lógica**)> THEN**
<bloco de comandos>
...]
[ELSE
<bloco de comandos>
...]
END IF

Exemplo:

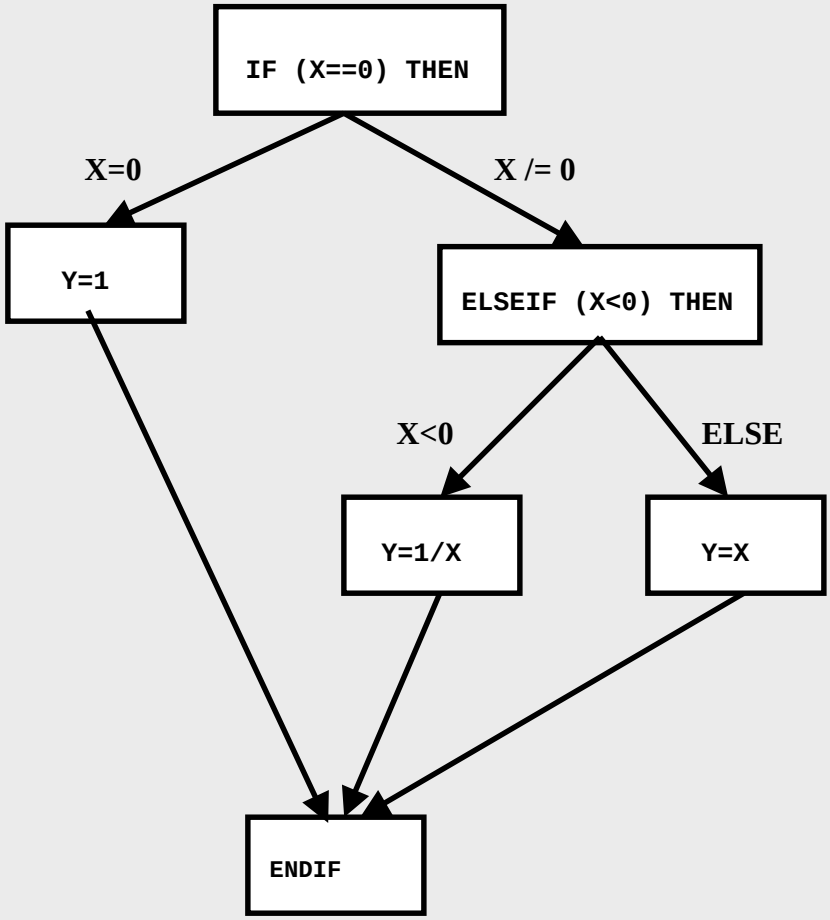
```
IF  ( X == 0 ) THEN
     PRINT *, “X=0”
     X=X+1
     Y=10**X
ELSEIF ( X < 0 ) THEN
        PRINT *, “X é um número negativo”
        Y=1/10**X
ELSE
     PRINT *, “X é um número positivo”
     Y=10**X
END IF
```

- Importante:

  - Podem existir vários ELSEIFs dentro da lógica de um IF;
  - Não existe ENDIF para ELSEIF, pois ele está ligado á lógica do IF principal;
  - Pode existir um IF dentro do bloco de comandos de um ELSEIF, neste caso, existe o ENDIF;

Diagrama de execução:

```
IF ( X == 0) THEN
   Y=1
ELSEIF ( X < 0 ) THEN
     Y=1/X
ELSE
     Y=X
END IF
```



3.5 - Comando IF...THEN...ELSEIF...END IF Identificados

- Determina a execução recursiva de comandos se uma condição lógica for verdadeira ou falsa em vários blocos de IF's identificados por um nome. A identificação dos IFs é simplesmente **"perfumaria"**, com a intenção de apresentar um código mais limpo e claro.

<nome>: IF <(expressão lógica**)> THEN**
<bloco de comando>
...
[ELSEIF <(expressão lógica**)> THEN <nome>**
<bloco de comandos>
...]
[ELSE <nome>
<bloco de comandos>
...]
END IF <nome>

Exemplo:

```
 outa: IF (a .NE. 0) THEN outa
           PRINT*, "a não é igual a 0"
       ina: IF (c .NE. 0) THEN ina
              PRINT*,"a não é igual a 0 e c não é igual a 0"
           ELSE ina
              PRINT*,"a não é igual a 0 mas c é igual a 0"
           ENDIF ina
       ELSEIF (a .GT. 0) THEN outa
              PRINT*,"a é maior que 0"
       ELSE outa
           PRINT*,"a deve ser menor que 0"
       ENDIF outa
```

EXERCÍCIO 6 – Comando IF

1. Caminhe para o diretório ~/curso/fortran/ex6. Edite o programa **triangulo.f90**

```
cd  ~/curso/fortran/ex6
nano triangulo.f90            Editores: vi, pico ou emacs
```

2. Caminhe para o diretório ~/cursos/fortran/ex6. Edite o programa **triangulo.f**

3. Esse programa solicita que se digite três valores inteiros que poderão definir os três lados de um triângulo Eqüilátero, Isósceles, Escaleno, ou não formar um triângulo.

4. Detalhe muito importante para a lógica do programa:

   Se três valores formam um triângulo, então 2 vezes o maior valor tem que ser menor que a soma de todos os três valores, ou seja, a seguinte expressão tem que ser verdadeira para que um triângulo exista.

   (2*MAX(lado1,lado2,lado3) < lado1+lado2+lado3)

5. Substitua as linhas com reticências pela lógica de programação que irá determinar que tipo de triângulo será formado. Analise com atenção o resto do programa para perceber como montar os comandos. Em um determinado instante, a expressão acima será utilizada.

6. Compile e execute o programa várias vezes, informando com os seguintes valores:

   - 1 1 1
   - 2 2 1
   - 1 1 0
   - 3 4 5
   - 3 2 1
   - 1 2 4

3.6 - Comando de “LOOP” Condicional - DO–EXIT-END DO

- “Loop” consiste de um bloco de comandos que são executados ciclicamente, infinitamente. É necessário um mecanismo condicional para sair do “loop”. O bloco de comandos que é executado ciclicamente é delimitado pelos comandos **DO...END DO** e o comando **EXIT,** que determina a saída do “loop”.

DO
...
IF <(expressão lógica**)> EXIT**
...
END DO

Exemplo:

```
i = 0
DO
   i = i + 1
   IF (i .GT. 100)  EXIT
   PRINT*, “I é”, i
END DO
PRINT*, “Fim do loop. I = “, i
```

3.7 - Comando de “LOOP” Cíclico Condicional DO–CYCLE-EXIT-END DO

- “Loop” cíclico que possui um mecanismo condicional para sair e iniciar o “loop” novamente. O comando **CYCLE** determina que a ação retorne para o início do “loop”.

DO
...
IF <(expressão lógica)> CYCLE
IF <(expressão lógica)> EXIT
...
END DO

Exemplo:

```
i = 0
DO
   i = i + 1
   IF (i >= 50 .AND. I <= 59)  CYCLE
   IF (i .GT. 100) EXIT
   PRINT*, “I é”, i
END DO
PRINT*, “Fim do loop. I = “, i
```

3.8 - “LOOPs” Identificados

- Neste caso, a identificação dos “loops”, funciona como uma posição de retorno para execução novamente do “loop”, que foi identificado no comando EXIT ou CYCLE.

Exemplo:

```
1     READ *,a
2     c=a;d=a-2;
3     outa:DO
4           READ *,b
5      inna:DO
6            IF (a .GT. b) EXIT outa     ! Pula para linha 12
7            IF (a .EQ. b) CYCLE outa    ! Pula para linha 4
8            IF (c .GT. d) EXIT inna     ! Pula para linha 10
9            IF (c .EQ. a) CYCLE         ! Pula para linha 6
10          ENDDO inna
11          d=a+b
12         ENDDO outa
```

3.9 - Comando DO-WHILE

- “loop” com a lógica do **“Faça enquanto”**, ou seja, condiciona a execução dos comandos dentro do “loop”, somente se, e enquanto a expressão for verdadeira. A expressão que condiciona o “loop” tem que vir entre parênteses; quando a expressão for falsa o “loop” não será mais executado.

DO WHILE <(expressão lógica)>
...
END DO

Exemplo:

```
READ *, “Salário=?”,salario
DO WHILE ( salario .LE. 5000 )
       salario=salario*1.05
END DO
```

3.10 - Comando DO iterativo

- "loops" que possuem um número fixo de ciclos, determinados pelo programador.

DO <variável>=<expressão1>, <expressão2> [,<expressão3>]
...
END DO

expressão1 → Valor inicial
expressão2 → Valor final
expressão3 → Valor de incremento

Exemplos:

```
DO  i1=1, 100, 2
       ... ! i1 será: 1,3,5,7...
       ... ! 50 iterações
END DO


DO  i2=1, 30
       ... ! i2 será: 1,2,3...30
       ... ! 30 iterações
END DO


READ *,m,n,x
DO  i3=m, n, x
       ...
END DO


DO  i4=85.5, 0, -0.5
       ... ! i4 será: 85.5,85,84.5,84...
       ... ! 171 iterações
END DO
```

3.11 - Comando SELECT CASE-CASE-END SELECT

- Construção similar ao IF, muito útil quando o valor analisado na expressão lógica, possuir diversos valores.

SELECT CASE <(expressão)>
CASE <(seleção)>
<comando>
CASE <(seleção)>
<comando>
...
CASE DEFAULT
<comando>
END SELECT

OBS: 1 - A seleção de valores, nos comandos CASE, pode representar **uma lista de valores**, quando se utiliza os caracteres especiais: **,** e/ou **:**

(6**,** 10**,** 100) → Valores iguais a 6, 10 ou 100
(10**:**65,67**:**98) → Valores entre 10 e 65, inclusive, ou entre 67 e 98 inclusive
(100**:**) → Valores maiores ou iguais a 100

2 - Inúmeros comandos CASE podem existir em um único SELECT CASE

3 - Só existe um CASE DEFAULT em um SELECT CASE

Exemplo:

```
SELECT CASE (I)
     CASE (1)
          PRINT*, "I é igual a 1"
     CASE (2:9)
          PRINT*, "I é maior ou igual a 2 e menor ou igual a 9"
     CASE (10:)
          PRINT*, "I é maior ou igual a 10"
     CASE DEFAULT
          PRINT*, "I é menor ou igual a 0"
END SELECT
```

3.12 – Operações entre Tipos de Dados

- As operações aritméticas entre valores, deve ser feita entre objetos do mesmo tipo de dado, pois as operações entre objetos diferentes irá converter um dos valores para o tipo de dado de maior precedência. A ordem de precedência dos tipos de dados é (**do menor para o maior**):

INTEGER → REAL → DOUBLE PRECISION → COMPLEX

```
INTEGER*REAL=REAL

REAL*INTEGER=REAL

DOUBLE PRECISION*REAL=DOUBLE PRECISION

COMPLEX*REAL=COMPLEX

REAL*DOUBLE PRECISION*INTEGER=DOUBLE PRECISION
```

3.13 – Divisão por Inteiros

- Ocorrem confusões em relação aos resultados quando da divisão entre números inteiros. Atenção, normalmente, o resultado é um valor inteiro.

```
REAL :: a, b, c, d, e

a = 1999/1000          a = 1
b = -1999/1000         b = -1
c = (1999+1)/1000      c = 2
d = 1999.0/1000        d = 1.999
e = 1999/1000.0        e = 1.999
```

3.14 - Procedimentos Internos do Fortran90

- Em linguagens de programação, normalmente, algumas tarefas são executadas com muita freqüência dentro de um programa, tornando-se repetitiva e dispendiosa a definição dessas tarefas sempre que necessário. O Fortran90 possui internamente, em torno de, 113 procedimentos que são chamados de procedimentos internos, pertencentes a biblioteca do Fortran, e executadas como funções. Existem diversos tipos de funções (Fortran Intel):

Matemáticas

```
ACOS, ACOSD, ACOSH, ASIN, ASIND, ASINH, ATAN, ATAN2, ATAN2D, ATAND, ATANH, COS, COSD, COSH,
COTAN, COTAND, EXP, LOG, LOG10, SIN, SIND, SINH, SQRT, TAN, TAND, TANH.

COS(x), SIN(x)              Cosseno e Seno, x em radianos;
ACOS(x), ASIN(x)            Arcocoseno e Arcoseno, x em radianos;
ACOSD(x), ASIND(x)          Arcocoseno e Arcoseno, x em graus;
ATAN(x)                     Arcotangente, x em radianos;
EXP(x)                      Exponencial: e^x
LOG(x)                      Logaritmo Natural;
LOG10(x)                    Logaritmo base 10;
SQRT(x)                     Raiz quadrada;
```

Numéricas

```
ABS, AIMAG, AINT, AMAX0, AMIN0, ANINT, CEILING, CMPLX, CONJG, DBLE, DCMPLX, DFLOAT, DIM,
DNUM, DPROD, DREAL, FLOAT, FLOOR, IFIX, ILEN, IMAG, INT, INUM, JNUM, MAX, MAX1, MIN, MIN1,
MOD, MODULO, NINT, QCMPLX, QEXT, QFLOAT, QNUM, QREAL, RAN, REAL, RNUM, SIGN, SNGL, ZEXT,
EXPONENT, FRACTION, NEAREST, RRSPACING, SCALE, SET_EXPONENT, SPACING, DIGITS, EPSILON,
HUGE, MAXEXPONENT, MINEXPONENT, PRECISION, RADIX, RANGE, SIZEOF, TINY, DOT_PRODUCT, MATMUL,
MCLOCK, SECNDS.

ABS(x)                      Valor absoluto;
INT(x)                      Converte x para um valor inteiro;
REAL(x)                     Converte x para um valor real;
DBLE(x)                     Converte x para um valor de precisão dupla;
MAX(x1,x2,...,xn)           Valor máximo entre os valores x1 e xn;
MIN(x1,x2,...,xn)           Valor mínimo entre os valores x1 e xn;
MOD(a,p)                    Resto da divisão a/p;
```

Caracteres

```
ADJUSTL, ADJUSTR, INDEX, LEN_TRIM, REPEAT, SCAN, TRIM, VERIFY, ACHAR, CHAR,
IACHAR, ICHAR, IARG, IARGC, LEN, NARGS, NUMARG, LGE, LGT, LLE, LLT.

ADJUSTL(str)          Alinha o valor de str pela esquerda;
ADJUSTR(str)          Alinha o valor de str pela direita;
LEN(str)              Tamanho da variável str (Número de caracteres);
REPEAT(str,i)         Repete o valor de str, i vezes
TRIM(str)             Remove brancos a direita da variável str;
IACHAR(str)           Retorna o código do caractere str na tabela ASCII;
ACHAR(x)              Retorna o caractere identificado pelo código x na tabela ASCII;


       IACHAR('C')    67
       ACHAR(67)      C
```

Conjunto

```
MERGE, PACK, SPREAD, UNPACK, ALLOCATED, LBOUND, SHAPE, SIZE, UBOUND, MAXLOC, MINLOC,
CSHIFT, EOSHIFT, RESHAPE, TRANSPOSE, ALL, ANY, COUNT, MAXVAL, MINVAL, PRODUCT.
```

Outras

```
ASSOCIATED, BADDRESS, CACHESIZE, EOF, ERF, ERFC, FP_CLASS, IARGPTR, INT_PTR_KIND, ISNAN,
LOC, LOGICAL, MALLOC, MULT_HIGH, NULL, PRESENT, TRANSFER.
```

3.15 - Comando PRINT

- Comando que mostra o valor de variáveis na saída padrão (Tela do vídeo). Cada comando PRINT inicia uma nova linha.

  PRINT <formato> ,<imp1>,<imp2>,<imp3>, ...

  <formato> * Formato livre

```
Exemplo:

   PROGRAM uno
      IMPLICIT NONE
      CHARACTER(LEN=*), PARAMETER :: nome_completo = “Mauricio Silva”
      REAL :: x, y, z
      LOGICAL :: varlog
      x = 1; y = 2; z = 3
      varlog = (y .EQ. X)
      PRINT*, nome_completo
      PRINT*,”X=  “,x,” Y = “,y,” Z = “,z
      PRINT*,”Variável lógica: “,varlog
   END PROGRAM uno

   Mauricio Silva
   X = 1.000 Y = 2.000 Z = 3.000
   Variável lógica: F
```

3.16 - Comando READ

- Comando que lê valores da entrada padrão (teclado), e os armazena nas variáveis especificadas pelo comando READ.

  READ <formato> ,<imp1>,<imp2>,<imp3>, ...

  <formato> * Formato livre

```
Exemplo:

    PROGRAM leitura
      CHARACTER(LEN=15) :: nome
      REAL :: x,y,z
      PRINT *,”Entre com o nome?”
      READ *, nome
      PRINT*
      PRINT *,”Entre com os valores de x,y,z?”
      READ *, x, y, z
    END PROGRAM

Entre com o nome?
Joao Felipe

Entre com os valores de x,y,z?
10 20 30
```

EXERCÍCIO 7 – DO

1 – Sugere-se que a população dos Estados Unidos possa ser modelada de acordo com a fórmula:

$$P(t) = \frac{19727300}{1 + e^{-0.03134(t - 1913.25)}}$$

2 - Caminhe para o diretório ~curso/fortran/ex7. Edite o programa **pop.f90** . Esse programa calcula a população dos EEUU a cada 10 anos, entre 1790 e 2000. Acrescente ao programa o "loop" e a formula para visualizar os valores.

EXERCÍCIO 8 – SELECT CASE

1 – Caminhe para o diretório ~curso/fortran/ex8. Edite o programa **ingresso.f90**. Esse programa determina o preço de um ingresso a partir do número da cadeira escolhida:

CADEIRAS	**PREÇO**
50	R$ 50,00
100 – 140 e 200 – 240	R$ 25,00
300 – 340	R$ 20,00
400 – 440	R$ 15,00
Acima de 500	R$ 10,00
Qualquer outro valor	Cadeira não existe

2 – Substitua nas **reticências** a estrutura de um **SELECT CASE** que determinará o preço do ingresso.

3 – Compile e execute o programa diversas vezes para verificar se está certo.

4 – Altere o programa, de maneira que, fique em "loop" solicitando o número da cadeira, até ser digitado 0 que determina o fim do programa.

EXERCÍCIO 9 – Funções Matemáticas

1 – Caminhe para o diretório ~/curso/fortran/ex9. Edite o programa **PontoNoCirculo.f90** e altere o que for necessário para executar o programa corretamente. Esse programa calcula as coordenadas **x,y** de um ponto no círculo, tendo como valores de entrada o raio, **r** e o ângulo teta, **θ** em graus.

Fórmulas:

$$\theta(radianos) = \frac{\theta(graus)}{180} \times \pi$$

$$\pi = \arctan(1) \times 4$$

$$\text{sen}(\theta) = \frac{y}{r}$$

$$\cos(\theta) = \frac{x}{r}$$

Teste com os valores abaixo:

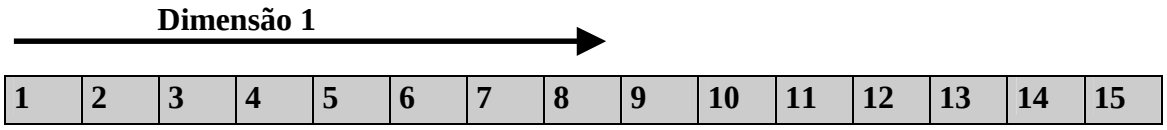
1. r = 12 θ = 77º
2. r = 1000 θ = 0º
3. r = 1000 θ = 90º
4. r = 20 θ = 100º
5. r = 12 θ = 437º

4 – CONJUNTO DE DADOS

Em programação, normalmente é necessário manipular uma grande quantidade de dados para efetuar determinadas tarefas: A média de um conjunto de números; a análise do resultado de testes; a ordenação de uma lista de números ou nomes; a solução de um sistema de equações lineares. Para eliminar a necessidade de definição de centenas de variáveis, pode-se usar o recurso de definição de conjuntos de dados ou “Arrays”, ou seja, variáveis com mais de um elemento, como um vetor de dados ou uma matriz de dados, que serão armazenados na memória e acessados individualmente de acordo com a sua posição espacial, definida pelas dimensões do conjunto de dados.

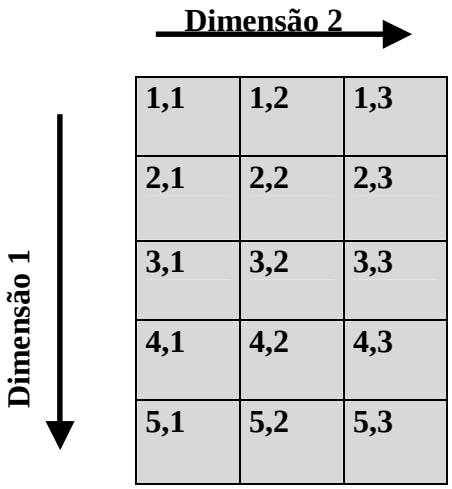
- Conjunto de 1 dimensão (Vetor)

  Vetor com 15 elementos:



- Conjunto de 2 dimensões (Matriz)

  Matriz de 15 elementos - 5 x 3:



4.1- Declaração de um Conjunto de Dados

- A definição de um conjunto de dados se faz durante a declaração das variáveis utilizando-se o atributo DIMENSION (Fortran90), o comando DIMENSION (Fortran77) ou definindo variáveis com índices.

```
REAL, DIMENSION(100)       :: R
REAL                       :: T(10,10)
REAL, DIMENSION(15)        :: X
REAL                       :: T(5,3)
```

- O Fortran90 permite definir até 7 dimensões;

```
REAL, DIMENSION(2,1,3,4,8,3,5)  :: Y
```

- O Fortran90 permite definir os limites inferior (LBOUND) e superior (UBOUND) de cada dimensão de um conjunto de dados, utilizando o caractere **" : "** para separar os limites. Caso não exista esse caractere, o valor informado será sempre o limite superior e o limite inferior será sempre 1. É possível limites com valores negativo;

```
REAL, DIMENSION(1:10,1:10) :: S
REAL, DIMENSION(-10:-1)    :: X
REAL, DIMENSION(1:5,1:3)   :: Y,Z
```

- O valor dos limites pode ser uma variável do tipo inteira ou uma expressão aritmética que resolva para um valor inteiro;

```
INTEGER, PARAMETER              :: lda = 5
REAL, DIMENSION(0:lda-1)        :: Y
REAL, DIMENSION(1+lda*lda,10)   :: Z
```

- O tamanho de um conjunto de dados é igual à multiplicação dos elementos em cada dimensão;

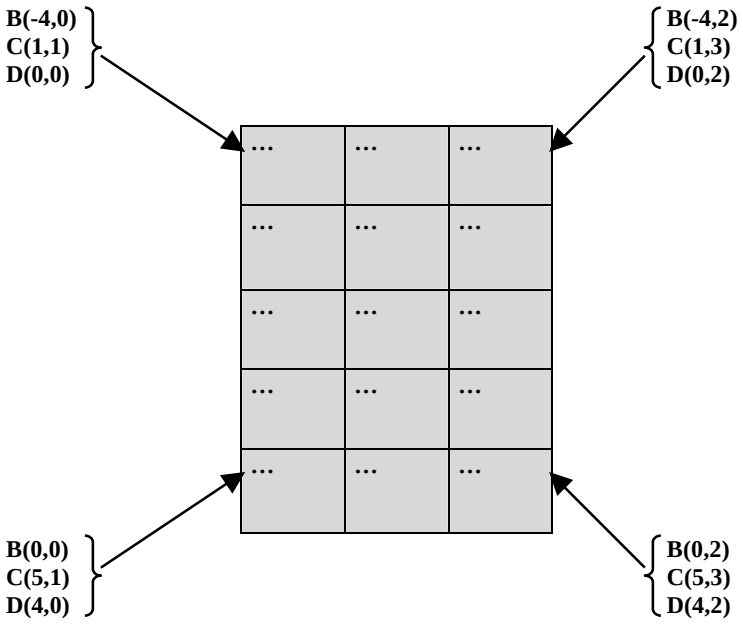
- O Fortran90 possui a definição da aparência (SHAPE) de um conjunto de dados, que é definido como sendo o número de elementos em cada dimensão;

```
REAL, DIMENSION(0:20, -5:40, -10:-1)

     3 Dimensões       Dimensão 1  limite inferior(LBOUND)=0
                                   limite superior(UBOUND)=20
                                   Número de elementos da dimensão=21

                       Dimensão 2  limite inferior(LBOUND)=-5
                                   limite superior(UBOUND)=40
                                   Número de elementos da dimensão=46

                       Dimensão 3  limite inferior(LBOUND)=-10
                                   limite superior(UBOUND)=-1
                                   Número de elementos da dimensão=10

     Tamanho do conjunto de dados = 21 x 46 x 10 = 9660 elementos

     Aparência do conjunto de dados SHAPE=(21,46,10)
```

4.2 - Visualização de um Conjunto de Dados

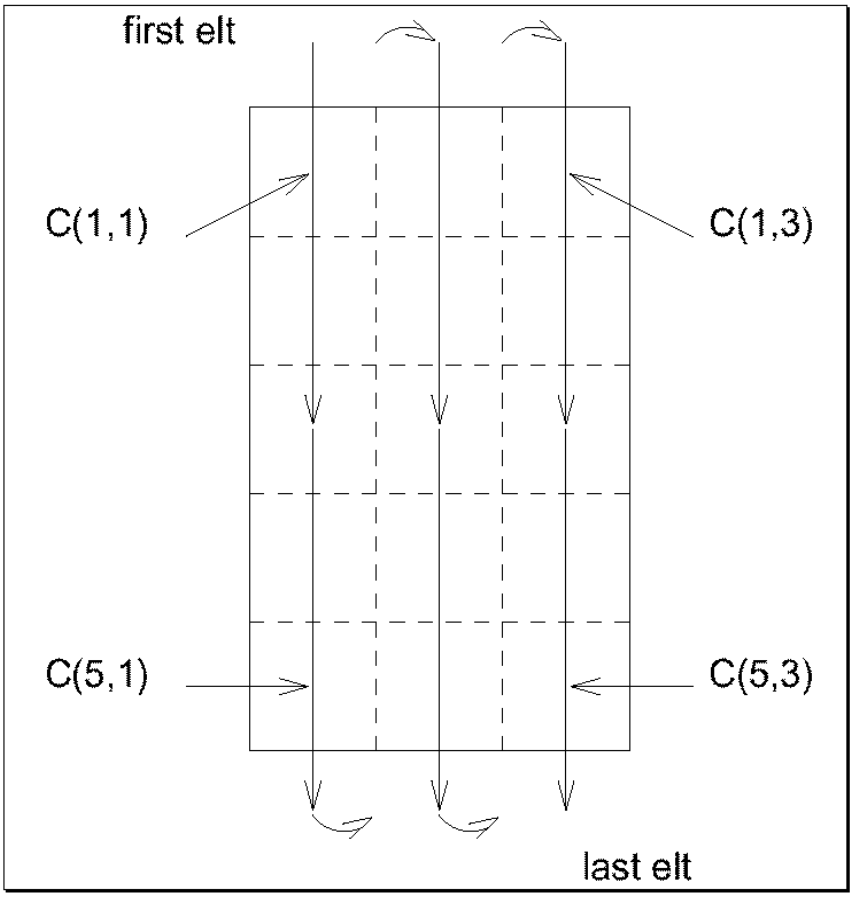
```
REAL, DIMENSION(15)          :: A
REAL, DIMENSION(-4:0,0:2)    :: B
REAL, DIMENSION(5,3)         :: C
REAL, DIMENSION(0:4,0:2)     :: D
```

A(1)	A(2)	A(3)	...	...	...	...	...	...	...	...	...	...	A(14)	A(15)



4.3 - Organização do Conjunto de Dados

- O Fortran90 não especifica como os conjuntos de dados serão organizados na memória; não existe associação de memória, isso permite que as implementações do Fortran90 tenham liberdade de otimização de acordo com a arquitetura do ambiente. No entanto, para leitura de dados e impressão de dados , a ordem por coluna, será utilizada.



4.4 – Sintaxe de Utilização de um Conjunto de Dados

- Iniciar os elementos de um conjunto **(/ ... /)**

```
INTEGER, DIMENSION(4) :: mat = (/2,3,4,5/)
```

- Operação aritmética entre todo o conjunto de dados:

```
A = 0.0
B = C + D      B,C,D devem possuir a mesma aparência(SHAPE).
```

- Operação aritmética entre alguns elementos do conjunto de dados:

```
A(1) = 0.0
B(0,0) = A(3) + C(5,1)
```

- Operação aritmética entre seções de elementos de um conjunto de dados:

```
A(2:4) = 0.0
B(-1:0,1:0) = C(1:2, 2:3) + 1.0
```

- Operações aritméticas com procedimentos internos do Fortran90:

```
B=SIN(C) + COS(D)   ⇔   B(i,j)=SIN(C(i,j)) + COS(D(i,j))

D=MAX(A,B,C)        ⇔   D(i,j)=MAX( A(i,j), B(i,j), C(i,j) )
```

- Seções de um conjunto de dados:

```
A([<limite inferior>:<limite superior>][:<incremento>])



A(:)          Representa todo o conjunto de dados;
A(3:9)        Representa os elementos A(3) até A(9) de 1 em 1;
A(8:3:-1)     Representa os elementos A(8) até A(3) de –1 em –1;
A(m:)         Representa os elementos A(m) até o limite superior;
A(::2)        Representa os elementos do conjunto, de 2 em 2;
```

4.5 – Leitura e Impressão dos Elementos de um Conjunto de Dados

- Considere A como um conjunto de dados de duas dimensões

Leitura:

```
INTEGER, DIMENSION(2,3) :: A
READ*, A                        OBS: 6 elementos serão lidos na ordem por colunas;

1 2 3 4 5 6


A(1,1)=1    A(1,2)=3    A(1,3)=5
A(2,1)=2    A(2,2)=4    A(2,3)=6
```

Impressão:

```
PRINT*, A                       OBS: 6 elementos serão impressos na ordem por colunas;

1 2 3 4 5 6
```

Exemplos:

1	**4**	**7**
2	**5**	**8**
3	**6**	**9**

```
PRINT*, 'Elemento da Matriz     =', a( 3 , 2 )
PRINT*, 'Secção da Matriz       =', a( : , 1 )
PRINT*, 'Sub-Matriz             =', a( :2 , :2 )
PRINT*, 'Toda Matriz            =', a
PRINT*, 'Matriz Transposta              =',TRANSPOSE(a)
```

```
Elemento da Matriz      = 6
Secção da Matriz        = 1 2 3
Sub-Matriz              = 1 2 4 5
Toda Matriz             = 1 2 3 4 5 6 7 8 9
Matriz Transposta       = 1 4 7 2 5 8 3 6 9
```

4.6 – Funções de Tratamento de Conjuntos de Dados

```
REAL, DIMENSION(-10:10,23,14:28) :: A
```

- **LBOUND(***ARRAY[,DIM]***)**

  Identifica o limite **inferior** das dimensões de um conjunto de dados.

```
LBOUND(A)           Resultado é um conjunto com (/-10,1,14/)
LBOUND(A,1)         Resultado é um escalar com valor –10
```

- **UBOUND(***ARRAY[,DIM]***)**

  Identifica o limite **superior** das dimensões de um conjunto de dados.

```
UBOUND(A)           Resultado é um conjunto com (/10,23,28/)
UBOUND(A,3)         Resultado é um escalar com valor 28
```

- **SHAPE(***ARRAY***)**

  Identifica qual é a aparência de um conjunto de dados.

```
SHAPE(A)            Resultado é um conjunto com (/21,23,15/)
SHAPE((/4/))        Resultado é um conjunto com (/1/)
```

- **RESHAPE(***ARRAY,SHAPE***)**

  Altera a aparência de um conjunto de dados. Muito útil na declaração do conjunto, com os valores especificados.

```
REAL, DIMENSION(3,3) :: unida = RESHAPE((/1,0,0,0,1,0,0,0,1/),(/3,3/))
```

- **SIZE(***ARRAY[,DIM]***)**

  Identifica o numero de elementos de um conjunto de dados.

```
SIZE(A,1)           21
SIZE(A)             7245
```

- **MAXVAL(***ARRAY[,DIM]***), MINVAL(***ARRAY[,DIM]***)**

  Retorna o valor máximo/mínimo entre todos os elementos ou entre os elementos de uma dimensão de um conjunto de dados.

$$X = \begin{pmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{pmatrix}$$

```
MAXVAL(X)=6  MAXVAL(X,DIM=1)=(/2,4,6/) MAXVAL(X,DIM=2)=(/5,6/)
MINVAL(X)=1  MINVAL(X,DIM=1)=(/1,3,5/) MINVAL(X,DIM=2)=(/1,2/)
```

- **SUM(***ARRAY[,DIM]***)**

  Retorna a soma de todos os elementos ou entre os elementos de uma dimensão de um conjunto de dados.

  $\mathbf{X} = \begin{pmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{pmatrix}$ `SUM(X)=21 SUM(X,DIM=1)=(/3,7,11/) SUM(X,DIM=2)=(/9,12/)`

- **PRODUCT(***ARRAY[,DIM]***)**

  Retorna o produto de todos os elementos ou entre os elementos de uma dimensão de um conjunto de dados.

  $\mathbf{X} = \begin{pmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{pmatrix}$ `PRODUCT(X)=720 PRODUCT(X,DIM=1)=(/2,12,30/) PRODUCT(X,DIM=2)=(/15,48/)`

- **DOT_PRODUCT(***ARRAY1,ARRAY2***)**

  É a soma do produto interno de dois vetores de dois conjuntos de dados, em apenas uma dimensão.

  `DP=DOT_PRODUCT(A,B)` ⇔ `DP=A(1)*B(1)+A(2)*B(2)+…` ⇔ `DP=SUM(A*B)`

- **MATMUL(***ARRAY1,ARRAY2***)**

  O resultado é a tradicional multiplicação de dois conjuntos de dados de no máximo duas dimensões (matrizes), podendo ser um dos conjuntos de dados, de uma dimensão.

  `MATMUL(X,Y)` ⇔ `SUM(X(i,:)*Y(:,j))`

4.7 - Alocação Dinâmica de Conjuntos de Dados

O Fortran90 permite a alocação dinâmica de memória para conjunto de dados. Quando for necessário, deve-se utilizar uma combinação de atributo, comando e função que irá disponibilizar a memória para o programa.

- Declaração do conjunto de dados: Atributo - **ALLOCATABLE**

```
INTEGER, DIMENSION( : ), ALLOCATABLE :: idade          ! 1D
REAL, DIMENSION( : , : ), ALLOCATABLE :: velo          ! 2D
```

- Alocar a memória durante a execução do programa: Comando - **ALLOCATE**

  ALLOCATE(*ARRAY(DIM)[,STAT=<variável>]***)**

```
READ*, isize
ALLOCATE(idade(isize), STAT=err)
IF (err /= 0) PRINT*, “idade : Falhou a alocação de memória”

ALLOCATE(velo(0:isize-1,10), STAT=err)
IF (err /= 0) PRINT*, “velo : Falhou a alocação de memória”
```

- Verificar se existe memória alocada: Função – **ALLOCATED**

  ALLOCATED(*ARRAY***)**

- Liberar a memória alocada: Comando **DEALLOCATE**

  DEALLOCATE(*ARRAY[,STAT=<variável>]***)**

```
IF (ALLOCATED(idade)) DEALLOCATE(idade, STAT=err)
```

- OBS: O espaço de memória reservado para um conjunto de dados, permanece alocado até ser efetuado um DEALLOCATED ou, até o fim da execução do programa.

EXERCÍCIO 10 – Definição de um Conjunto de Dados

1 – Analise as declarações abaixo e identifique para cada uma o que é solicitado:

REAL, DIMENSION(1:10) :: ONE

Quantas dimensões? _____

Limite(s) inferior(es)?_____

Limite(s) Superior(es)?_____

Tamanho do conjunto?_____

REAL, DIMENSION(2,0:2) :: TWO

Quantas dimensões? _____

Limite(s) inferior(es)?_____

Limite(s) Superior(es)?_____

Tamanho do conjunto?_____

INTEGER, DIMENSION(-1:1,3,2) :: THREE

Quantas dimensões? _____

Limite(s) inferior(es)?_____

Limite(s) Superior(es)?_____

Tamanho do conjunto?_____

REAL, DIMENSION(0:1,3) :: FOUR

Quantas dimensões? _____

Limite(s) inferior(es)?_____

Limite(s) Superior(es)?_____

Tamanho do conjunto?_____

OBS: A solução está no diretório ~/curso/fortran/ex10

EXERCÍCIO 11 – Funções de Características de um Conjunto de Dados

1 – Dado a seguinte declaração de um conjunto de dados:

INTEGER, DIMENSION(-1:1,3,2) :: A

2 – Escreva um pequeno programa no diretório ~/curso/fortran/ex11 , que possua algumas funções de conjuntos que identificam:

1. O número total de elementos em A;
2. A aparência de A (Função SHAPE);
3. O limite inferior da dimensão 2;
4. O limite superior da dimensão 3.

OBS: A solução está no arquivo solucao.f90

EXERCÍCIO 12 – Funções de Operações de Conjunto de Dados

1 – Caminhe para o diretório ~/curso/fortran/ex12. Edite o programa **manipulacao.f90** que efetua algumas operações de conjuntos, acrescente na posição das reticências, o comando ou função do fortran apropriado para a posição.

1. Declare os conjuntos A e X como constantes no programa e duas variáveis escalares, M e N;

$$A = \begin{pmatrix} -4 & 5 & 9 \\ 6 & -7 & 8 \end{pmatrix} \qquad X = \begin{pmatrix} 1,5 \\ -1,9 \\ 1,7 \\ -1,2 \\ 0,3 \end{pmatrix}$$

2. Utilize uma função de conjunto de dados para definir o valor de M e N, como sendo o número de elementos do conjunto A, em cada dimensão. Imprima esses valores;

3. Imprima o conjunto A por linhas;

4. Imprima a soma do produto entre os elementos das colunas de A;

5. Imprima o produto da soma dos elementos das linhas de A;

6. Imprima a soma dos quadrados dos elementos de X;

7. Imprima a média dos elementos de X;

8. Imprima o maior valor absoluto de X;

9. Imprima o maior valor absoluto da soma das colunas de A.

OBS: A solução está no arquivo solucao.f90

EXERCÍCIO 13 – Uso de um Conjunto de Dados

1 – O salário recebido por alguns funcionários de uma empresa foi:

10500, 16140, 22300, 15960, 14150, 12180, 13230, 15760, 31000

e a posição hierárquica de cada funcionário é, respectivamente:

1, 2, 3, 2, 1, 1, 1, 2, 3

2 – Caminhe para o diretório ~/curso/fortran/ex13. Edite o programa **MatrizSalario.f90 .** Codifique o que é solicitado nas reticências. Esse programa calcula o custo total que a companhia terá com o incremento de 5%, 4% e 2% para as categorias 1, 2 e 3 respectivamente.

5 - SEÇÕES PRINCIPAIS DE PROGRAMAS

5.1 – Seções de um Programa

O Fortran90 possui duas seções principais de programa, que podem ser compilados independentes um do outro.

- **PROGRAM**

  Seção principal do programa aonde a execução inicia e finaliza. Pode conter vários procedimentos como subrotinas e funções.

- **MODULE**

  Seção do programa que pode conter novas declarações e procedimentos podendo ser anexado ao programa principal.

O Fortran90 possui dois tipos de definição de procedimentos, que devem estar dentro do contexto do programa principal ou do módulo.

- **SUBROUTINE**

  A subrotina é um programa com parâmetros de entrada e saída, comandos de declaração e lógica de execução, que pode ser chamada de dentro do programa principal com o comando CALL.

```
CALL <subrotina> ([parâmetro1, parâmetro2,...])
```

- **FUNCTION**

  A função é similar a uma subrotina, podendo ter vários parâmetros de entrada, comandos de declaração e lógica de execução, no entanto, retorna apenas um único valor que é armazenado no nome da função. A função pode ser executada dentro de um outro comando.

```
PRINT*, "Resultado da Função é:", f(x)
```

5.2 - Programa Principal: PROGRAM

[PROGRAM [<*nome do programa*>**]]**
...
! Comandos de declaração
...
! Comandos executáveis
...
[CONTAINS
<*definição dos procedimentos internos*>**]**
END [PROGRAM [<*nome do programa*>**]]**

- O comando PROGRAM é opcional, assim como o nome do programa, mas é uma boa prática sempre usá-lo;
- O programa principal pode conter comandos de declaração, comandos executáveis e procedimentos internos: subrotinas e funções, definidas pelo usuário. Esses procedimentos são definidos e separados do resto do programa pelo comando CONTAINS;
- O programa principal pode definir inúmeros procedimentos internos, mas estes, não podem definir novos procedimentos;
- O programa principal tem que ser finalizado com o comando END.

Exemplo:

```
PROGRAM main
     IMPLICIT NONE
     REAL :: x
     READ*, x
     PRINT*, SQRT(x)     ! Função interna
     PRINT*, Negative(x) ! Função do usuário
CONTAINS
     REAL FUNCTION Negative(a)
          REAL, INTENT(IN) :: a
          Negative = -a
     END FUNCTION Negative
END PROGRAM Main
```

5.3 – Seções Internas: Procedimentos

- Procedimentos internos, procedimentos "intrínsicos" – O compilador Fortran90 possui cerca de 113 procedimentos entre subrotinas e funções, que são utilizadas para resolver uma determinada necessidade do programa.

- Sempre se questione se é necessário criar um código para resolver alguma tarefa do seu programa. Existem dezenas de bibliotecas de rotinas, algumas de domínio público, que facilitam a solução de determinados problemas. Normalmente, os códigos dessas rotinas já foram bastante depurados e otimizados.

Bibliotecas comerciais:

NAG - Numerica**l A**lgorithms **G**roup - **www.nag.co.uk**

IMSL - International **M**athematics and **S**tatistics **L**ibrary - **www.vni.com/products/imsl**

ESSL - Engineering and **S**cientific **S**ubroutine **L**ibrary - **www-03.ibm.com/systems/p/software/essl/index.html**

Bibliotecas de domínio público:

BLAS – Basic **L**inear **A**lgebra **S**ubroutine **- www.netlib.org/blas**

LAPACK – **L**inear **A**lgebra **PACK**age - **www.netlib.org/lapack**

Diversas bibliotecas para Fortran: www.fortranlib.com/freesoft.htm

Compiladores Fortran "freeware"

Fortran da GNU: **gcc.gnu.org/fortran**

Fortran da Intel: **software.intel.com/en-us/articles/non-commercial-software-download**

Conversor de Fortran para C: **www.netlib.org/f2c**

5.3.1 - Procedimentos: SUBROUTINE

SUBROUTINE <nome> [(<argumentos "dummy">)]
<Declaração dos argumentos associados ou "dummy">
<Declaração local dos objetos>
...
<Comandos executáveis>
END [SUBROUTINE [<nome>]]

- Para se definir uma subrotina usa-se a estrutura **SUBROUTINE – END SUBROUTINE**;
- Para se usar uma subrotina usa-se o comando **CALL <nome da subrotina>**;
- Uma subrotina "enxerga" todas as variáveis declaradas no programa principal;
- Uma subrotina pode incluir a chamadas a outras subrotinas.
- Subrotinas são posicionadas entre o comando **CONTAINS** e o comando **END PROGRAM**;
- O Fortran90 permite execução recursiva de subrotinas, ou seja, subrotina que chama ela mesma; basta acrescentar o comando **RECURSIVE** antes de **SUBROUTINE**.

Exemplo:

```
PROGRAM algo
    IMPLICIT NONE
    REAL, DIMENSION(100) :: numeros
     ...
    CALL ImprimeNum(numeros)
     ...
CONTAINS
    SUBROUTINE ImprimeNum(num)
     REAL, DIMENSION(:), INTENT(IN) :: num
     PRINT*,"Esses são os números", num
    END SUBROUTINE ImprimeNum
END PROGRAM algo
```

OBS: A variável "dummy": *num*, assume a dimensão da variável *números*, definida no programa principal.

5.3.2 - Procedimentos: FUNCTION

[<Tipo da Função>] FUNCTION <nome> [(<argumentos>)]
Declaração dos argumentos "dummy"
Declaração dos objetos
Comandos executáveis
Comando de atribuição do resultado
END [FUNCTION [<nome>]]

- Função funciona sobre o mesmo princípio da subrotina, com a diferença de que a função retorna um único resultado;
- Uma função é definida usando-se a estrutura **FUNCTION – END FUNCTION**;
- Pra usar uma função, basta 'chamá-la' pelo nome;
- Função pode ser definida na área de declaração de variáveis quando se identifica o tipo da função;
- O tipo da função pode ser especificado quando for definida a função, ou dentro da definição da função, pois o nome da função receberá o resultado;
- O Fortran90 permite execução recursiva de funções, ou seja, função que chama ela mesma; basta acrescentar o comando **RECURSIVE** antes de **FUNCTION**.

Exemplo Alternativa 1

```
PROGRAM algo
  IMPLICIT NONE
     ...
  PRINT*, F(a,b)
     ...
  CONTAINS
  REAL FUNCTION F(x,y)
     REAL, INTENT(IN) :: x,y
     F=SQRT(x*x + y*y)
  END FUNCTION F
END PROGRAM algo
```

Exemplo Alternativa 2

```
PROGRAM algo
  IMPLICIT NONE
     ...
  PRINT*, F(a,b)
     ...
  CONTAINS
  FUNCTION F(x,y)
     REAL :: F
     REAL, INTENT(IN) :: x,y
     F=SQRT(x*x + y*y)
  END FUNCTION F
END PROGRAM algo
```

5.3.3 – Detalhes de Procedimentos

- Argumentos associados ou **"dummy"**

Considere o comando:

```
PRINT*, F(a,b)
```

E a definição da função:

```
REAL FUNCTION F(x,y)
```

Os argumentos do programa principal: ***a*** e ***b***, estão associados aos argumentos "dummy": ***x*** e ***y***, da função F. Se os valores dos argumentos "dummy" forem alterados durante a execução da função, então os valores associados, também serão modificados.

- Definição dos **Objetos Locais**

Na definição do procedimento abaixo:

```
SUBROUTINE madras(i,j)
        INTEGER, INTENT(IN) :: i,j
        REAL :: a
        REAL, DIMENSION(i,j) :: x
                …
END SUBROUTINE madras
```

a e ***x*** são conhecidos como objetos locais, e ***x***, pode ter uma aparência e tamanho diferente a cada chamada do procedimento. Isso significa que, os objetos locais:

1. são iniciados cada vez que o procedimento é chamado;
2. são eliminados quando o procedimento finaliza sua execução;
3. não mantêm seus valores entre chamadas, a não ser que na declaração, sejam iniciados com um determinado valor (REAL :: a=0);
4. não utilizam o mesmo espaço de endereçamento de memória do programa principal.

- Atributo **INTENT**

Esse atributo da declaração de objetos locais é utilizado para facilitar a compilação e por razões de otimização, mas a sua utilização não é essencial.

INTENT(IN)	Objeto de entrada, é enviado pelo programa principal e não pode ser modificado durante a execução do procedimento;
INTENT(OUT)	Objeto de saída, o seu valor é definido pelo procedimento que é devolvido ao programa principal;
INTENT(INOUT)	Objeto de entrada e saída, enviado pelo programa principal, pode ser modificado pelo procedimento e então devolvido ao programa.

Exemplo:

```
SUBROUTINE ex(arg1,arg2,arg3)
     REAL, INTENT(IN) :: arg1
     INTEGER, INTENT(OUT) :: arg2
     CHARACTER, INTENT(INOUT) :: arg3
     REAL r
     r=arg1*ICHAR(arg3)
     arg2=ANINT(r)
     arg3=CHAR(MOD(127,arg2))
END SUBROUTINE ex
```

1. **arg1** não é modificado no procedimento;
2. o valor de **arg2** não é utilizado até ser definido pelo procedimento;
3. arg3 é utilizado e o seu valor redefinido.

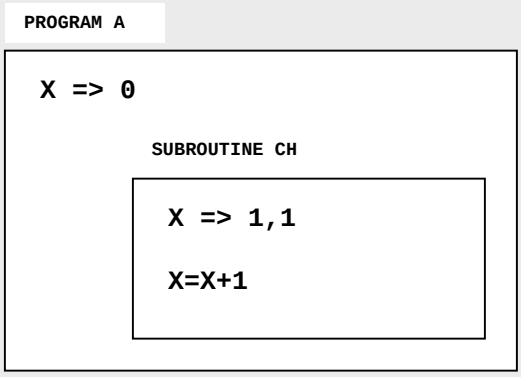
- Atributo **SAVE**

Variáveis definidas com o atributo SAVE são conhecidas como objetos estáticos e por sua vez possuem armazenamento estático não sendo reiniciadas a cada chamada do procedimento. Com exceção das variáveis "dummy" e das variáveis definidas com o atributo PARAMETER, as variáveis declaradas possuem, implícito, o atributo SAVE, desde que possuem um valor inicial na declaração.

Exemplo 1 (Sem o atributo SAVE)

```
PROGRAM A
   REAL ::  X
   CALL CH()
   PRINT *,"Programa principal x=",X
   CALL CH()
   PRINT *,"Programa principal x=",X
CONTAINS
   SUBROUTINE CH
         REAL :: X
         X=X+1
         PRINT*,"Subrotina x=",X
   END SUBROUTINE
END
```

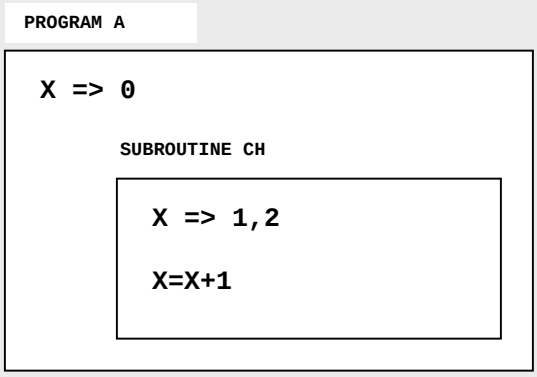
```
Subrotina x=   1.000000
Programa principal x=  0.0000000E+00
Subrotina x=   1.000000
Programa principal x=  0.0000000E+00
```



Exemplo 2 (Com o atributo SAVE)

```
PROGRAM A
   REAL ::  X
   CALL CH()
   PRINT *,"Programa principal x=",X
   CALL CH()
   PRINT *,"Programa principal x=",X
CONTAINS
   SUBROUTINE CH
         REAL,SAVE :: X
! ou     REAL :: X=0.0
         X=X+1
         PRINT*,"Subrotina x=",X
   END SUBROUTINE
END
```

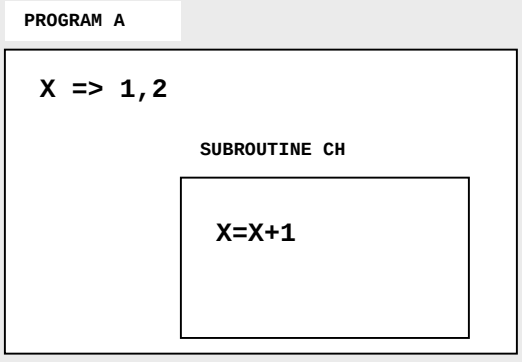
```
Subrotina x=   1.000000
Programa principal x=  0.0000000E+00
Subrotina x=   2.000000
Programa principal x=  0.0000000E+00
```



Exemplo 3 (Sem a definição de X na subrotina)

```
PROGRAM A
   REAL ::  X
   CALL CH()
   PRINT *,"Programa principal x=",X
   CALL CH()
   PRINT *,"Programa principal x=",X
CONTAINS
   SUBROUTINE CH
         X=X+1
         PRINT*,"Subrotina x=",X
   END SUBROUTINE
END
```

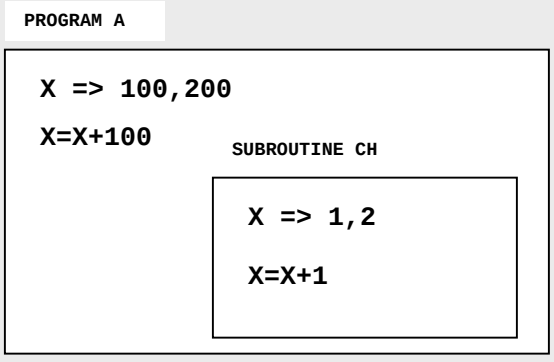
```
Subrotina x=   1.000000
Programa principal x=   1.000000
Subrotina x=   2.000000
Programa principal x=   2.000000
```



Exemplo 4 (Vriáveis X indenpendentes)

```
PROGRAM A
   REAL ::  X
   X=X+100
   CALL CH()
   PRINT *,"Programa principal x=",X
   X=X+100
   CALL CH()
   PRINT *,"Programa principal x=",X
CONTAINS
   SUBROUTINE CH
         REAL,SAVE :: X
         X=X+1
         PRINT*,"Subrotina x=",X
   END SUBROUTINE
END
```

```
Subrotina x=   1.000000
Programa principal x=   100.0000
Subrotina x=   2.000000
Programa principal x=   200.0000
```



5.4 – Seção Especial: MODULE

- MODULE é uma estrutura de programa Fortran90, independente do programa principal, podendo ser compilado e utilizado por diversos programas como se fosse um procedimento externo;

- Normalmente, um MODULE é criado quando se percebe que o código pode ser utilizado em diversas situações diferentes;

- Um MODULE pode conter:

    1 – Declaração global de objetos;
    2 – Diversos procedimentos SUBROUTINE e/ou FUNCTION;
    3 – Definição de tipos de dados (TYPE);

- Um MODULE é utilizado no programa principal através do comando USE;

- Sintaxe:

```
MODULE <nome>
        <Comandos de declaração>
   [ CONTAINS
        <Definição de procedimentos>]
END [ MODULE [ <nome> ]]
```

Exemplo:

Definição do módulo - Declara as variáveis X, Y e Z como variáveis reais globais.

```
MODULE global
    REAL, SAVE :: X, Y, Z
END MODULE global
```

Utilização do módulo:

```
PROGRAM teste
   USE global
   IMLPICIT NONE
     ...
END PROGRAM teste
```

EXERCÍCIO 14 – Subrotina

1 – Caminhe para o diretório: ~/curso/Fortran/ex14. Edite o arquivo **subrotina.f90** e adicione a subrotina de acordo com a descrição do problema.

Esse programa possui uma rotina interna que retorna, como primeiro argumento, a soma de dois números reais.

Subrotina summy(arg1, arg2, arg3)

arg1 variável com resultado
arg2 variável com 1º número
arg3 variável com 2º número

arg1=arg2+arg3

O programa principal deverá chamar a rotina três vezes e imprimir o resultado:

- Números: 2.6 e 3.1
- Números: 6.1 e 9.2
- Números: 0.1 e 0.555

EXERCÍCIO 15 – Função

1 – Caminhe para o diretório: ~/curso/Fortran/ex15. Edite o arquivo **funcao.f90** e adicione a função de acordo com a descrição do problema.

Esse programa possui uma função interna que retorna a soma de dois números reais, fornecido pelos argumentos.

Função real summy(arg1,arg2)

Arg1 variável com 1º número
Arg2 variável com 2º número

summy=arg1+arg2

O programa principal deverá chamar a rotina quatro vezes e imprimir o resultado:

- Números: 1.0 e 2.0
- Números: 1.0 e -1.0
- Números: 0.0 e 0.0
- Números: 1.0E54 e 9192652.0

EXERCÍCIO 16 – Procedimentos Internos

1 – Caminhe para o diretório: ~/curso/Fortran/ex16. Edite o arquivo **NumeroRandomico.f90** e substitua pelo comando adequado nas reticências.

Esse programa chama uma função que por sua vez chama duas vezes um procedimento interno do Fortran90 (subrotina), RANDOM_NUMBER(r), para obter um número randômico. Esse número vai permitir e simular um arremesso de dados.

2-Mesmo após substituir as reticências pelos comandos adequados, provavelmente haverá erros na compilação (Problema de recursividade). Leia com atenção os prováveis erros e tente corrigi-los.

EXERCÍCIO 17 – Definição de um Módulo

1 – Caminhe para o diretório: ~/curso/Fortran/ex17. Edite o programa **DefModulo.f90**

Este programa cria um módulo que possui a definição de duas funções, uma para calcular a média, e uma para calcular o desvio padrão, de um vetor real de tamanho indefinido. O módulo também registra quantas vezes as funções foram utilizadas.

2 – Substitua as linhas com reticências com o comando adequado a definição de um módulo.

3 - Somente compile o programa, apenas para gerar o objeto e o módulo:

ifort –c DefModulo.f90

OBS: A compilação do módulo é apenas para gerar o código objeto, utiliza-se apenas a opção: **-c** . Após a compilação, serão criados dois arquivos, o objeto com o mesmo nome do programa, mas com extensão **.o** e o módulo, com o nome dado na definição, com a extensão **.mod**

EXERCÍCIO 18 – Uso de um Módulo

1 – Caminhe para o diretório ~/curso/Fortran/ex18. Edite o programa **UsoModulo.f90**

2 – Substitua as linhas com reticências com o comando adequado ao uso de um módulo.

3 – Compile o programa apenas gerando o objeto:

ifort –c UsoModulo.f90

4 – Copie o **objeto (.o)** e o módulo **(.mod)**, gerado no exercício anterior, para este diretório. Crie o executável "linkeditando" os dois objetos, da seguinte maneira:

ifort –o Teste_modulo UsoModulo.o DefModulo.o

OBS: Tudo isso pode ser feito em um único passo, desde que os arquivos do exercício anterior já tenham sido copiados.

ifort UsoModulo.f90 DefModulo.o –o Teste_modulo

5 – Execute o programa com os seguintes valores de entrada:

3.0 17.0 –7.56 78.1 99.99 0.8 11.7 33.8 29.6

1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0 11.0 12.0 13.0 14.0

6 – TRATAMENTO DE ARQUIVOS

6.1 - ENTRADA / SAÍDA

- O Fortran90 possui uma variedade de recursos para a manipulação de arquivos de dados com opções de I/O (Input/Output), que permitem diferentes tipos de arquivos se conectarem ao programa principal para leitura e/ou gravação dos registros de dados;

- ***Registro*** é uma seqüência de valores numéricos ou uma seqüência de caracteres, sendo que existem três tipos de registros no Fortran:

- Formatado	Existe uma regra para a disposição da seqüência de dados;
- Não Formatado	Não existe uma regra para a seqüência de dados, sendo o caractere branco o limite entre uma seqüência e outra;
-Fim de arquivo	(ENDFILE)Registro especial que determina o fim do arquivo.

- ***Arquivo*** é uma seqüência de registros;

- Em Fortran90, um arquivo é conectado durante a execução do programa através de uma ***unidade lógica,*** que é definida por um número inteiro e positivo, no comando de abertura do arquivo. Na maioria dos compiladores esse número está limitado entre 1 e 999;

- Cada ***unidade lógica*** pode possuir diversas propriedades para a manipulação dos arquivos:

- Arquivo	Nome do arquivo que será conectado;
- Ação	Modo de acesso ao arquivo: **read** (leitura) **write** (gravação) **read/write** (leitura e gravação);
- Status	Status do arquivo: **old** (arquivo já existe) **new** (arquivo novo) **replace** (sobreposição de arquivo);
- Método de acesso	Modo de leitura dos registros: **sequential** (sequêncial), registro após registro, desde o início do arquivo; **direct** (direto), acessa pelo número do registro no arquivo.

6.2 - Comando OPEN

- É utilizado para conectar um arquivo a uma unidade lógica e definir algumas características de conexão.

OPEN ([UNIT=]<inteiro>
[, FILE=<"arquivo">]
[, ERR= <label>]
[, IOSTAT=<ivar>]
[, outros parâmetros])

- O único parâmetro obrigatório o valor inteiro que irá identifica a ***unidade lógica***, mas é aconselhável informar o nome do arquivo e mais dois parâmetros para analisar possíveis erros.

UNIT= Valor inteiro, qualquer, que especifica a ***unidade lógica*** para um arquivo;

FILE= Especifica, entre aspas, o nome do arquivo que será conectado;

ERR= Valor inteiro, que especifica uma posição lógica de controle, no programa, aonde será analisado o erro que ocorreu na abertura do arquivo. **OBS: Especificar esse parâmetro, evita que o Fortran cancele a execução do programa, caso o erro ocorra.**

IOSTAT= Variável inteira que irá armazenar o código do erro na abertura de arquivo. Um valor igual a zero significa que não houve erro.

- **Outros parâmetros**, muito utilizados:

STATUS= Especifica o **status** do arquivo:

'OLD'	O arquivo já existe;
'NEW'	O arquivo não existe;
'REPLACE'	O arquivo será sobreposto;
'SCRATCH'	O arquivo é temporário e será apagado quando fechado (CLOSE);
'UNKNOW'	Desconhecido ("default", assumirá OLD ou NEW);

ACCESS= Especifica o método de acesso:

'DIRECT'	Acesso direto a registros individuais. É obrigado a usar a opção RECL;
'SEQUENTIAL'	Acesso sequêncial, linha por linha ("default");

ACTION= Especifica o modo de acesso ao arquivo:

'READ'	Somente leitura;
'WRITE'	Somente gravação;
'READWRITE'	Leitura e gravação;

RECL= Especifica uma expressão, que resolva para um valor inteiro, que irá determinar o tamanho do registro, somente quando o modo de acesso for direto.

Exemplos:

```
PROGRAM arquivo
       CHARACTER(LEN=40) :: FILNM
       DO I=1,4
             FILNM = ''
             PRINT *, 'Entre com o nome do arquivo.'
             READ *, FILNM
             OPEN (UNIT=1, FILE=FILNM, STATUS='OLD', IOSTAT=IERR, ERR=100)
             PRINT*,'Abrindo arquivo: ', FILNM

                   ...

             CLOSE (UNIT=1)
             STOP
100          IF (IERR .EQ. FOR$IOS_FILNOTFOU) THEN  ! Código 29 – Arquivo não existe
                   PRINT *, 'Arquivo: ', FILNM, ' não existe. '
             ELSE IF (IERR .EQ. FOR$IOS_FILNAMSPE) THEN
                   PRINT *, 'Arquivo: ', FILNM, ' com problemas, entre novamente:'
             ELSE
                   PRINT *, 'Erro indefinido! Código =', IERR
                   STOP
             END IF
       END DO
       PRINT *, 'Arquivo não encontrado. Corrija o problema e execute novamente! '
END PROGRAM arquivo
```

```
OPEN(17,FILE='saida.dat',ERR=10,STATUS=REPLACE',&
       ACCESS='SEQUENTIAL',ACTION='WRITE')
```

```
OPEN(14,FILE='entra.dat',ERR=10,STATUS='OLD',&
       RECL=1024, ACCESS='DIRECT',ACTION='READ')
```

6.3 - Comando READ

- O comando READ transfere os dados de um arquivo externo, de acesso seqüencial ou direto, para a lógica de execução do programa.

READ([UNIT=]<inteiro>
[,[FMT]=<formato ou label>]
[,ERR=<label>]
[,IOSTAT=<ivar>]
[,END=<label>]
[,ADVANCE=<modo>]
[,EOR=<label>]
[,REC=<expressão inteira>]
[,SIZE=<ivar>]) <lista de variáveis>

- O único parâmetro realmente obrigatório é a ***unidade lógica*** para identificar de qual arquivo está sendo feito à leitura, no entanto, se for uma leitura em um arquivo com os dados formatados, o parâmetro de formato (FMT), também será necessário.

UNIT= Valor inteiro, qualquer, que especifica a ***unidade lógica*** para um arquivo. O símbolo ***** representa a unidade padrão, neste caso, o monitor. A unidade pode indicar um arquivo externo **ou uma variável caractere criada no programa;**

FMT= Especifica o formato da leitura dos dados. A especificação do formato tem vir, primeiro, entre aspas, e segundo, entre parênteses, ou, pode ser feita uma indicação de uma posição lógica no programa, aonde se encontra um comando FORMAT com a descrição do formato;

FMT='(I4)' **FMT=200**

ERR= Valor inteiro, que especifica uma posição lógica de controle no programa, aonde será analisado o erro que ocorreu na leitura do arquivo. **OBS: Especificar esse parâmetro, evita que o Fortran cancele a execução do programa, caso o erro ocorra.**

IOSTAT= Variável inteira que irá armazenar o código do erro na abertura de arquivo. Um valor igual a zero significa que não houve erro.

END= Valor inteiro, que especifica uma posição lógica de controle no programa, aonde será analisado o erro de **fim de arquivo**.

REC= Especifica uma expressão, que resolva para um valor inteiro, que irá determinar o número do registro. Utilizado somente quando o modo de acesso for direto.

ADVANCE= Parâmetro que especifica ('YES' ou 'NO') se cada comando READ deve, ou não, iniciar a leitura em um novo registro. O padrão é: ADVANCE='YES', se for utilizado o parâmetro para não avançar a leitura, então, será obrigatório o arquivo ser conectado no modo de acesso seqüencial e a descrição do formato de leitura, no parâmetro FMT.

EOR= Valor inteiro, que especifica uma posição lógica de controle no programa, aonde será analisado o erro **fim de registro**. Este parâmetro só é utilizado no comando READ e quando o parâmetro ADVANCE='NO' for utilizado;

SIZE= Variável inteira que irá armazenar o número de caracteres lidos pelo comando READ e, somente quando o parâmetro ADVANCE='NO' tiver sido utilizado.

Exemplo 1:

```
PROGRAM ler
501 FORMAT(3I5)
    INTEGER A,B,C
    DO
       READ(*,501,END=50,ERR=90) A,B,C
       IF(A=0 .OR. B=0 .OR. C=0) THEN
                PRINT *,"Um dos lados e igual a zero !"
                STOP
       ELSE
                S = (A + B + C) / 2.0
         AREA = SQRT( S * (S - A) * (S - B) * (S - C))
         PRINT *, A, B, C, AREA
       ENDIF
    END DO
50  PRINT *, "Fim do programa!"
    STOP
90  PRINT *,"Entrada de dado errada!"
    STOP
END PROGRAM
```

Exemplo 2:

Arquivo:control_file.txt

```
pi 3.1415
invalid 5.7
vector 0 1 1 2 3
```

Arquivo:control_file.f90

```
! JASON BLEVINS <JRBLEVIN@SDF.LONESTAR.ORG>
! DURHAM, MAY 6, 2008
PROGRAM controle
  IMPLICIT NONE
  CHARACTER(LEN=100) :: buffer, label
  INTEGER :: pos
  INTEGER, PARAMETER :: fh = 15
  INTEGER :: ios = 0
  INTEGER :: line = 0
  REAL :: pi
  INTEGER, DIMENSION(5) :: vector
  OPEN(fh, FILE='controle.txt')
  DO WHILE (ios == 0)
     READ(fh, '(A)', IOSTAT=ios) buffer
     IF (ios == 0) THEN
        line = line + 1
        pos = SCAN(buffer, '     ')
        label = buffer(1:pos)
        buffer = buffer(pos+1:)
        SELECT CASE (label)
        CASE ('pi')
            ! Leitura de uma unidade interna: o registro armazenado em buffer
           READ(buffer, *, IOSTAT=ios) pi
           PRINT *, 'READ pi: ', pi
        CASE ('vector')
            ! Leitura de uma unidade interna: o registro armazenado em buffer
           READ(buffer, *, IOSTAT=ios) vector
           PRINT *, 'READ vector: ', vector
        CASE DEFAULT
           PRINT *, 'SKIPPING INVALID LABEL AT LINE', line
        END SELECT
     END IF
  END DO
END PROGRAM controle
```

Outros exemplos:

```
      READ(14,FMT='(3(F10.7,1x))',REC=exp) a,b,c

      READ(*, '(A)', ADVANCE='NO',EOR=12,SIZE=nch) str
```

6.4 - Comando WRITE

- O comando WRITE transfere os dados para um arquivo externo, de acesso seqüencial ou direto, de acordo com a lógica de execução do programa.

WRITE([UNIT=]<inteiro>
[,[FMT]=<formato ou label>]
[,ERR=<label>]
[,IOSTAT=<ivar>]
[,ADVANCE=<modo>]
[,REC=<expressão inteira>]
[,SIZE=<ivar>]) <lista de variáveis>

- O único parâmetro realmente obrigatório é a ***unidade lógica*** para identificar para qual arquivo está sendo feito à gravação, no entanto, se for uma gravação em um arquivo com os dados formatados, o parâmetro de formato (FMT), também será necessário.

UNIT= Valor inteiro, qualquer, que especifica a ***unidade lógica*** para um arquivo. O símbolo ***** representa a unidade padrão, neste caso, o monitor;

FMT= Especifica o formato da gravação dos dados. A especificação do formato tem vir, primeiro, entre aspas, e segundo, entre parênteses, ou, pode ser feita uma indicação de uma posição lógica no programa, aonde se encontra um comando FORMAT com a descrição do formato;

ERR= Valor inteiro, que especifica uma posição lógica de controle no programa, aonde será analisado o erro que ocorreu na gravação do arquivo. **OBS: Especificar esse parâmetro, evita que o Fortran cancele a execução do programa, caso o erro ocorra.**

IOSTAT= Variável inteira que irá armazenar o código do erro na abertura de arquivo. Um valor igual a zero significa que não houve erro.

REC= Especifica uma expressão, que resolva para um valor inteiro, que irá determinar o número do registro. Utilizado somente quando o modo de acesso for direto.

ADVANCE= Parâmetro que especifica (‘YES’ ou ‘NO’) se cada comando WRITE deve, ou não, iniciar a gravação em um novo registro. O padrão é: ADVANCE=’YES’, se for utilizado o parâmetro para não avançar a gravação, então, será obrigatório o arquivo ser conectado no modo de acesso seqüencial e a descrição do formato de gravação, no parâmetro FMT.

Exemplo 1:

```
PROGRAM divisores
C     Este programa acha os divisores de uma valor inteiro informado.
C     O divisor é salvo em um arquivo.
      INTEGER n, k, d(10)
      OPEN (UNIT = 1, FILE = "div.txt")
      PRINT *, "Informe um valor inteiro positivo :"
      READ *, n
      WRITE (1,*) "Divisores de ", N, " :"
      k = 0
      DO i = 1, n
            IF (MOD(n,i) .EQ. 0) THEN
                  k = k + 1
                  d(k) = i
            END IF
            IF (k .EQ. 10) THEN
                  WRITE (1,5) (d(j), j = 1, 10)
                  k = 0
            END IF
      END DO
      WRITE (1,5) (d(j), j = 1, k)
  5   FORMAT (10I7)
      CLOSE (1)
      PRINT *, "Os divisores estão salvos no arquivo 'div.txt' "
      END
```

Arquivo div.txt

```
Divisores de       100000  :
      1      2      4      5      8     10     16     20     25     32
     40     50     80    100    125    160    200    250    400    500
    625    800   1000   1250   2000   2500   3125   4000   5000   6250
  10000  12500  20000  25000  50000 100000
```

Outros exemplos:

WRITE(17,FMT='(I4)',IOSTAT=stat, ERR=10) val

WRITE(*, '(A)', ADVANCE='NO') 'Amarelo'

6.5 – “loops” Inseridos nos comandos READ/WRITE

- A sintaxe de “loop” subentendido (“Implied-DO-list”), geralmente é utilizado em operações de INPUT/OUTPUT para um conjunto de dados. Possui a seguinte forma:

(<lista de variáveis>, <variável loop>=<expr>,<expr>[,expr])

Exemplos:

```
INTEGER :: j
REAL, DIMENSION(10) :: A
READ (*,*)  ( A(j), j=1,10 )
WRITE (*,*) ( A(j), j=10,1,-1 )



INTEGER :: i, j
REAL, DIMENSION(10,10) :: B
WRITE (*,*) (( B(I,J), I=1,10 ),  J=1,10 )



DO I = 1, 5
   WRITE(*,1) (A(I,J), J=1,10)
END DO
1 FORMAT (10I6)



( ( A(I,J) , J = 1,3 ) , B(I), I = 6,2,-2 )


A(6,1), A(6,2), A(6,3), B(6), A(4,1), A(4,2),
A(4,3), B(4), A(2,1), A(2,2), A(2,3), B(2)
```

6.6 - Descritores de Edição

- O Fortran possui vários descritores de edição de formatos, que permite ler, escrever e imprimir dados em diversas maneiras possíveis.

- Os dados, usualmente, são armazenados na memória no formato binário. Por exemplo, o número inteiro ***6***, deve ser armazenado como ***0000000000000110***, aonde ***0s*** e ***1s*** representam dígitos binários. Os registros de dados em arquivos formatados, consistem de caracteres; quando um dado é lido de um registro, ele precisa ser convertido de caractere para uma representação interna de processamento e vice-versa.

- A especificação de formato fornece a informação necessária para determinar como essa conversão deva ser realizada. A especificação de formato é basicamente uma lista de descritores de edição, divididos em três categorias: Descritores de dados, descritores de controle e descritores de cadeia de caracteres (“strings”).

Descritores de dados

A*[w]*	Descreve dados do tipo **caractere**. O tamanho ***w*** do campo é opcional.
I*w*	Descreve dados do tipo **inteiro**; **w** indica o número de dígitos.
F*w.d*	Descreve dados do tipo **real**; ***w*** indica o número total de dígitos e ***d*** o número de decimais.
E*w.d*	Descreve dados do tipo **real** com **expoente**; ***w*** indica o número total de dígitos e ***d*** o número de decimais.
L*w*	Descreve dados do tipo **lógico**;***w*** indica o número de caracteres no campo lógico.
B*w*	Descreve dados do tipo **inteiro** em **base binária**; ***w*** indica o número de dígitos no campo binário.
O*w*	Descreve dados do tipo **inteiro** em **base octal**; ***w*** indica o número de dígitos no campo octal.
Z*w*	Descreve dados do tipo **inteiro** em **base hexadecimal**; ***w*** indica o número de dígitos no campo hexadecimal.

Descritores de Controle

BN	Na leitura de dados, ignora os brancos a esquerda de campos numéricos.
BZ	Na leitura de dados, trata os brancos a esquerda de campos numéricos, como zeros.
T*n*	Posiciona a leitura ou gravação na posição ***n***.
*[n]***X**	Pula ***n*** espaços em branco.
[r]/	Finaliza o registro atual e pula para o início do próximo registro (o ***r*** significa repetição da ação).
:	Para de processar o formato se não possuir mais variáveis para utilizar a sequência editada.
$	Na **gravação** de dados, se o primeiro caractere for um ***branco*** ou **+**, este símbolo, elimina o caractere de fim de linha (***cr*** ou ***lf***), mantendo a continuação da gravação, na mesma linha.

Descritores de cadeia de caracteres (“strings”)

*n***H***texto*	Grava o texto, sem precisar colocar entre aspas, com ***n*** caracteres, no registro de saída.
‘texto’	Grava o texto entre aspas simples ou apóstrofes.
“texto”	Grava o texto entre aspas duplas.

6.7 – Formatação dos Dados (FORMAT/FMT=)

- A formatação é um recurso necessário quando da leitura, gravação e impressão dos dados. É utilizado nos comandos: READ. WRITE e PRINT, através de uma expressão caractere de formato ou a indicação de uma posição lógica no programa com um comando de formatação;

  FMT — Nos comandos READ e WRITE, utiliza-se o parâmetro FMT= para indicar a expressão caractere com o formato, ou, a posição lógica do programa aonde se encontra o comando de formatação. No comando PRINT, só é possível informar a posição lógica, do comando de formatação;

  FORMAT — Comando que descreve os formatos dos dados. Sempre deve vir rotulado, ou seja, com um número a esquerda do comando, indicando uma posição lógica dentro do programa;

- A expressão caractere com a formatação dos dados é uma sequência de descritores de edição;

Exemplo 1:

```
        WRITE(17, FMT='(2X, 2I4, 1X, "Nome ", A7)')11, -195, "Paulo"
        WRITE(*, FMT=10)-1.05133, 333356.0
   10   FORMAT('vals',2(F15.6,2X))
```

Registro na unidade 17

				1	1	-	1	9	5		N	o	m	e		P	a	u	l	o		

Registro na tela

V	a	l	s						-	1	.	0	5	1	3	3	0				3	3	3	3	5	6	.	0	0	0	0	0	0		

Exemplo 2:

```
        WRITE(*,FMT='(2X,2(I4,1X),"Nome ",A4,F13.5,1X,E13.5)')&
              77778,3,'ABCDEFGHI',14.45,14.5666666
```

		*	*	*	*					3		N	o	m	e		A	B	C	D					1	4	.	4	5	0	0	0				0	.	1	4	5	6	7	E	+	0	2

Exemplo 3:

```
        PRINT 10, (X(I), I = 1, N )
        10 FORMAT( 'X1:', I2 : ' X2:', I2 : ' X3:', I3 )

        Quando N=1

        X1: 1

        Sem o caractere ":" o resultado seria:

        X1: 1 X2:   X3:
```

Exemplo 4:

Formato	Valor especificado	Resultado
F8.5,F8.5	1234567812345.67	123.45678, 12345.67
E9.1,F9.3	734.432E8123456789	734.432E8, 123456.789

Exemplo 5:

```
        PRINT 25
        25 FORMAT (T51,'COLUNA 2',T21,'COLUNA 1')
```

```
                          2                                                 5
                          1                                                 1
                          C O L U N A  1                                    C O L U N A  2
```

Exemplo 6:

Arquivo: FOR002.DAT

```
001 0101 0102 0103 0104 0105
002 0201 0202 0203 0204 0205
003 0301 0302 0303 0304 0305
004 0401 0402 0403 0404 0405
005 0501 0502 0503 0504 0505
006 0601 0602 0603 0604 0605
007 0701 0702 0703 0704 0705
008 0801 0802 0803 0804 0805
009 0901 0902 0903 0904 0905
010 1001 1002 1003 1004 1005
```

```
PROGRAM ex
       INTEGER I, J, A(2,5), B(2)
       OPEN (unit=2, access='sequential', file='FOR002.DAT')
       READ (2,100) (B(I), (A(I,J), J=1,5),I=1,2)  ❶
100    FORMAT (2(I3, X, 5(I4,X), /)) ❷
       WRITE (6,999) B, ((A(I,J),J=1,5),I=1,2) ❸
999    FORMAT (' B is ', 2(I3, X), '; A is', / 1(' ', 5(I4, X)))
       READ (2,200) (B(I), (A(I,J), J=1,5),I=1,2) ❹
200    FORMAT (2(I3, X, 5(I4,X), :/) )
       WRITE (6,999) B, ((A(I,J),J=1,5),I=1,2) ❺
       READ (2,300) (B(I), (A(I,J), J=1,5),I=1,2) ❻
300    FORMAT ((I3, X, 5(I4,X)))
       WRITE (6,999) B, ((A(I,J),J=1,5),I=1,2) ❼
       READ (2,400) (B(I), (A(I,J), J=1,5),I=1,2) ❽
400    FORMAT (I3, X, 5(I4,X) )
       WRITE (6,999) B, ((A(I,J),J=1,5),I=1,2) ❾
END PROGRAM ex
```

❶ Este comando READ lerá: B(1); A(1,1) até A(1,5); B(2); A(2,1) até A(2,5). O primeiro registro a ser lido será o 001 que inicia o processo de leitura;

❷ No comando FORMAT indicado, processa dois registros com o formato **I3, X, 5(I4, X)**. O caractere (**/**) força a leitura para o segundo registro, após o dado A(1,5) ser processado. **A leitura para, no início de um terceiro registro, após o dado A(2,5) ser processado**;

❸ Este comando WRITE mostra o resultado na tela (unidade padrão 6):

```
B is 1 2 ; A is
 101 102 103 104 105
 201 202 203 204 205
```

❹ Este comando READ começa a leitura no registro 004. O caractere (**/**) força a leitura para o próximo registro, após o dado A(1,5) ser processado. **O caractere (:), para a leitura, após o dado A(2,5) ser processado, antes do caractere (/) forçar a leitura de um novo registro**;

❺ Este comando WRITE mostra o resultado na tela (unidade padrão 6):

```
B is 4 5 ; A is
 401 402 403 404 405
 501 502 503 504 505
```

❻ Este comando READ começa a leitura no registro 006. Após o dado A(1,5) ser processado, **ocorre a reversão do formato para continuar a leitura, ou seja, o formato é reiniciado; a colocação dos parênteses no início da sequência dos descritores, causa a leitura do próximo registro, iniciando o processo de formatação no parênteses da esquerda, antes do I3**;

❼ Este comando WRITE mostra o resultado na tela (unidade padrão 6):

```
B is 6 7 ; A is
 601 602 603 604 605
 701 702 703 704 705
```

❽ Este comando READ começa a leitura no registro 008. Após o dado A(1,5) ser processado, **ocorre a reversão do formato para continuar a leitura, ou seja, o formato é reiniciado; a colocação dos parênteses no início da sequência dos descritores, causa a leitura do próximo registro, iniciando o processo de formatação no parênteses da esquerda, antes do I4**;

❾ Este comando WRITE mostra o resultado na tela (unidade padrão 6):

```
B is 8 90 ; A is
 801 802 803 804 805
 9010 9020 9030 9040 100
```

6.8 - Outros comandos de I/O

- **CLOSE[([UNIT=]unit[,ERR=])][unit]**

  Finaliza a conexão entre a unidade lógica e o arquivo;

```
CLOSE(17,ERR=99)
```

- **REWIND[([UNIT=]unit[,ERR=])][unit]**

  Posiciona a leitura, da unidade lógica, no primeiro registro do arquivo;

```
REWIND 14
```

- **BACKSPACE[([UNIT=]unit[,ERR=])][unit]**

  Retorna a leitura em um registro e posiciona na primeira coluna;

```
BACKSPACE(UNIT=14)
```

- **ENDFILE[([UNIT=]unit[,ERR=])][unit]**

  Força a gravação de um "registro" indicando o *fim de arquivo (EOF)*, sem fechar a conexão com a unidade lógica, e se posiciona após esta marca de *fim de arquivo*. OBS: Não é permitido gravar registros após a marca de *fim de arquivo*. Se for preciso ler ou gravar algum dado, será necessário executar um REWIND ou BACKSPACE para voltar a uma posição antes desta marca.

```
WRITE (14, *) x
ENDFILE 14
REWIND 14
READ (14, *) y
```

- **DELETE([UNIT=]unit[,REC=][,ERR=])**

  Elimina, da unidade lógica especificada em modo de acesso direto, o registro informado no parâmetro REC=

```
DELETE(17,REC=20)
```

6.9 - Comando DATA

- Comando que permite iniciar uma ou várias listas de variáveis, de diversas maneiras.

- Muito utilizado para iniciar conjuntos de dados (“arrays”), devido ao recurso de representar o conjunto de dados como um “loop” subentendido.

DATA <lista de variáveis1>/<valores1>/[,<lista de variáveis2> /<valores2>/,...]

lista de variáveis variáveis ou conjuntos de dados;

valores valores das variáveis, separados por vírgulas. Pode-se usar o operador ***** para representar repetição de valores;

Exemplo 1:

```
      INTEGER :: count, I, J
      REAL :: inc, max, min
      CHARACTER(LEN=5) :: luz
      LOGICAL :: vermelho, azul, verde
      DATA count/0/, I/5/, J/100/
      DATA inc, max, min/1.0E-05, 10.0E+05, -10.0E+05/
      DATA luz/’Clara’/
      DATA vermelho/.TRUE./, azul, verde/.FALSE.,.FALSE./
```

Exemplo 2:

```
      REAL :: matriz(100,100)
      DATA matriz(1,1:100)/100*1.0/       ! Primeira linha
      DATA matriz(100,1:100)/100*1.0/     ! Última linha
      DATA matriz(2:99,1)/98*1.0/         ! Primeira coluna
      DATA matriz(2:99,100)/98*1.0/       ! Última coluna
      DATA matriz(2:99,2:99)/9604*0.0/    ! Interior do conjunto
```

Exemplo 3:

```
      DATA matriz(1,1:100)/50*1.0,50*2.0/ ! Primeira linha
```

Exemplo 4:

```
      DIMENSION A(10,10)
      DATA A/100*1.0/                              ! Iniciar por nome do conjunto
      DATA A(1,1), A(10,1), A(3,3) /2*2.5, 2.0/ ! Iniciar por elemento do conjunto
      DATA ((A(I,J), I=1,5,2), J=1,5) /15*1.0/ ! Iniciar por “loops” subentendidos



      REAL,DIMENSION(100,100) :: diag

      ! Iniciar os elementos da diagonal

      DATA (diag(i,i),i=1,100)/100*5.0/

      ! Iniciar os elementos do triângulo superior e inferior à diagonal

      DATA ((diag(i,j),diag(j,i),j=i+1,100),i=1,100)/9900*0.0/
```

7 – DEFINIÇÃO DE TIPO DE DADOS

- É muito vantajoso declarar alguns objetos (variáveis) como uma estrutura agregada de outros tipos de dados, como:

  Coordenadas: (x,y,z)
  Endereços: nome, rua, número, CEP, etc.

- O Fortran90 permite a composição de tipos de dados com os tipos de dados implícitos da linguagem; são chamados de Tipo Derivado e são definidos através do comando TYPE.

7.1 – Definição: Estrutura TYPE

TYPE [[, atributos] ::] <nome>
<comandos de declaração ou componentes do tipo>
...
END TYPE [type-name]

Exemplo:

```
! DERIVED.F90
PROGRAM deriva
     TYPE pessoa
          INTEGER idade
          CHARACTER (LEN = 20) nome
     END TYPE pessoa
          …
END PROGRAM
```

7.2 – Comando de Declaração: TYPE

- Após a definição do tipo de dados, pode-se utilizá-lo na declaração das variáveis, utilizando-se o comando de declaração TYPE.

TYPE (nome) [, atributos ::] lista de variáveis

Exemplo:

```
! DERIVED.F90
PROGRAM deriva
     TYPE pessoa
          INTEGER idade
          CHARACTER (LEN = 20) nome
     END TYPE pessoa
     REAL :: x
     INTEGER :: valor
     TYPE (pessoa) :: jorge
     TYPE (pessoa) :: ana
          …
END PROGRAM
```

7.3 – Utilização de um Novo Tipo de Dados

- Uma variável definida por um tipo de dados derivado, possui componentes na sua estrutura, que são identificados pelo símbolo **%**, quando da sua utilização.

  <variável definida>%<componente definido>

- Outra forma, é especificar os componentes do novo tipo de dados entre parênteses, na ordem que foram definidos os componentes.

  variável definida = tipo definido (componente1, componente2,...)

Exemplo:

```
! DERIVED.F90
PROGRAM deriva
     TYPE pessoa
          INTEGER idade
          CHARACTER (LEN = 20) nome
     END TYPE pessoa
     REAL :: x
     INTEGER :: valor
     TYPE (pessoa) :: jorge
     TYPE (pessoa) :: ana
     jorge = pessoa( 33, 'Jorge Alberto' )
     ana%idade = 56
     ana%nome = 'Ana Aparecida'
     WRITE (*,*) jorge
     WRITE (*,*) ana
END
```

8 – COMANDOS DE EXCEÇÃO

8.1 – Comando GOTO

- Comando que transfere a execução, imediatamente, para outra posição no programa, através de uma "*label*" numérico. É muito útil, mas deve ser usado com muito cuidado e somente em casos excepcionais.

GOTO <label> ou **GO TO <label>**

Exemplo:

```
      INTEGER in
10    PRINT *, 'Entre com um número de 1 a 10: '
      READ *, in
      SELECT CASE (in)
            CASE (1:10)
                  EXIT
            CASE DEFAULT
                  PRINT *, 'Número errado! Tente de novo.'
                  GOTO 10
            END SELECT
```

8.2 – Comando RETURN

- Utilizado em subrotinas e funções, este comando transfere a execução para a última linha de um procedimento, ou seja, finaliza a execução do procedimento e retorna para o programa principal.

RETURN

Exemplo:

```
      SUBROUTINE sub(ierror)
            INTEGER, INTENT(OUT) :: ierror
                  ...
            ALLOCATE(A(100),STAT=ierror)
            IF (ierror>0) THEN
                  PRINT*, 'memory fault'
                  RETURN
            END IF
                  ...
      END SUBROUTINE
```

8.3 - STOP

- Comando que causa a parada imediata da execução do programa.

STOP [”texto”]

Exemplo:

```
      OPEN(1,FILE='file1.dat', status='OLD', ERR=100)
            ...
100   STOP 'Ocorreu um erro na abertura do arquivo!'
      END
```

9 - RECOMENDAÇÕES DE CODIFICAÇÃO

- Sempre utilize o comando de declaração **IMPLICT NONE**, ou seja, sempre declare todas as variáveis que forem utilizadas no programa;

- Comandos, procedimentos internos e as definidas pelo programador, devem ser colocadas em maiúsculas;

  OBS: Não é obrigatório! Apenas uma recomendação.

- Variáveis e constantes, em minúsculas;

  OBS: Não é obrigatório! Apenas uma recomendação.

- Cada comando deve ser posto numa linha;

- Codifique com recuos;

- Acrescente comentários às linhas (**!**).

EXERCÍCIO 19 – I/O

1 - Caminhe para o diretório ~/curso/Fortran/ex19. Edite o programa **Write_io.f.** Substitua as reticências pelo comando adequado.

Esse programa solicita dados para serem digitados e os grava em um arquivo.

2 – Compile e execute o programa, testando com os seguintes valores:

```
Blair        94.  97.  97.  94.
Major         2.   6.   6.   5.
Ashdown      49.  28.  77.  66.
END           0.   0.   0.   0.
```

3 - Edite o programa **Read_io.f.** Substitua as reticências pelo comando adequado.

4 - Compile e execute o programa.

EXERCÍCIO 20 – I/O

1 - Caminhe para o diretório ~/curso/fortran/ex20. Edite o programa **io_spec.f90** Substitua as reticências pelos comandos de I/O, solicitados nas linhas de comentário.

Este programa grava e lê um arquivo com um único registro.

2 – Compile e execute o programa. Verifique se o último resultado está de acordo com os valores:

1 2 3 4 5 6 7 8 -1 -2

EXERCÍCIO 21 – Formatação

1 – Dado o comando abaixo:

READ(*,'(F10.3,A2,L10)') A,C,L

Como será representado o valor de A (REAL), o valor de C (CHARACTER de tamanho 2) e o valor de L LOGICAL logical) para os seguintes valores de dados? (OBS: b significa espaço em branco.)

bbb**5.34**bbb**NO**b**.TRUE.**
5.34bbbbbb**Y**bb**F**bbbbb
b**6**bbbbbb**3211**bbbbbb**T**
bbbbbbbbbbbbbbbbbb**F**

2 - Caminhe para o diretório ~/curso/Fortran/ex21. Edite o programa **IOFormatado.f.** Substitua as reticências pelo comando adequado.

Esse programa gera um arquivo com linhas de cabeçalho e linhas de dados, sendo: NAME (até 15 caracteres), AGE (até 3 dígitos), HEIGHT (em metros 4 posições e 2 decimais) e o FONE (4 dígitos inteiros).

```
                        Height
   Name          Age    (metres)    Tel. No.
   ----          ---    --------    --------
Bloggs J. G.      45      1.80        3456
Clinton P. J.     47      1.75        6783
```

EXERCÍCIO 22 – Derivação de Tipo de Dados

1 - Caminhe para o diretório ~/curso/fortran/ex22. Edite o programa **type.f90.** Substitua as reticências pelo comando adequado.

Esse programa cria um novo tipo de dado, que configura um elemento da tabela periódica, com três definições: o símbolo, o número atômico e a massa atômicas.

2 – Compile e execute o programa.

10 - REFERÊNCIAS

1 - IBM XL Fortran for AIX User's Guide Version 8 Release 1

2 - IBM XL Fortran for AIX Language Reference Version 8 Release 1

3 - INTEL Fortran Language Reference

4 - The University of Liverpool – Fortran 90 Programming
Dr. A.C. Marshall

5 - Fortran 90 Handbook - Complete ANSI / ISO Reference - Intertext Publications McGraw-Hill Book Company
Jeanne C. Adams
Walter S. Brainerd
Jeanne T. Martin
Brian T. Smith
Jerrold L. Wagener

6 - Introduction to Fortran 90 for Scientists and Engineers
Brian D Hahn Department of Applied Mathematics - University of Cape Town