

Universidade Estadual de Campinas

Introdução ao Fortran90

Ricardo Almeida de Mendonça Küsel

Versão: Abril/2008



Centro Nacional de Processamento de Alto Desempenho - SP

CONTEÚDO

CONTEÚDO	
INTRODUÇÃO	
1-HISTÓRIA	04
2-FORTRAN 77	05
3-FORTRAN 90	06
ELEMENTOS DO FORTRAN 90	
4-EXEMPLO DE PROGRAMA	07
5-RECOMENDAÇÕES DE CODIFICAÇÃO	08
6-CARACTERÍSTICAS DE CODIFICAÇÃO	09
7-REGRAS DE CODIFICAÇÃO	10
8-COMPILAÇÃO NO AMBIENTE CENAPAD-SP	11
9-EXERCÍCIO 1- Compilação e Execução	12
10-EXERCÍCIO 2- Reestruturação de programa	13
11-ESTRUTURA DE PROGRAMAS	14
12-TIPOS DE DADOS	14
13-CONSTANTES	15
14-TIPO IMPLÍCITO	15
15-DECLARAÇÃO NUMÉRICA E LÓGICA	16
16 - DECLARAÇÃO CARACTERE	16
17-DECLARAÇÃO DE CONSTANTES	17
18-INICIALIZAÇÃO DE VARIÁVEIS	17
19-EXERCÍCIO 3 - Declaração de Variáveis	18
20-EXPRESSÕES	19
21-Operador de ATRIBUIÇÃO	19
22-Operadores NUMÉRICOS	20
23-Operadores RELACIONAIS	20
24-Operadores LÓGICOS	21
25-Operador CARACTERE	21
26-Precedência de Operadores	21
27-EXERCÍCIO 4 – Expressões	22
CONSTRUÇÕES DE CONTROLE DE EXECUÇÃO	
28-COMANDOS DE CONTROLE DE FLUXO	23
29-Comando IF	24
30-Comando IF...THEN...END IF	25
31-Comando IF...THEN...ELSE...END IF	26
32-Comando IF...THEN...ELSEIF...END IF	27
33-Comando IF...THEN...ELSEIF...END IF Identificado	28
34-EXERCÍCIO 5 - Comando IF	29
35-Comando DO–EXIT-END DO “LOOP” Condicional	30
36-Comando DO–CYCLE-EXIT-END DO “LOOP” Cíclico Condicional	30
37-“LOOPS” Identificados	31
38-Comando DO-WHILE	31
39-Comando DO iterativo	31
40-Comando SELECT-CASE	32
41-DIVISÃO POR INTEIROS	32
42-EXERCÍCIO 6 – SELECT CASE	33

43-PROCEDIMENTOS INTRÍNSECOS	34
44-Funções de CONVERSÃO	34
45-Funções MATEMÁTICAS	34
46-Funções NUMÉRICAS	35
47-Funções CARACTERES	35
48-EXERCÍCIO 7 – Funções Matemáticas	36
49-Comando PRINT	37
50-Comando READ	37
MATRIZES	
51-DEFINIÇÃO DE MATRIZES	38
52-DECLARAÇÃO DE MATRIZES	39
53-SÍNTEXE DE MATRIZES	40
54-SECÇÕES DE MATRIZES	40
55-IMPRESSÃO E LEITURA DE MATRIZES	41
56-FUNÇÕES DE MATRIZES	42
57-ALOCAÇÃO DE MATRIZES	43
58-EXERCÍCIO 8 – DEFINIÇÃO DE MATRIZES	44
59-EXERCÍCIO 9 – FUNÇÕES DE MATRIZES	45
60-EXERCÍCIO 10 – USO DE MATRIZES	45
SECÇÕES PRINCIPAIS DE PROGRAMAS	
61-SECÇÕES DE UM PROGRAMA FORTRAN	46
62-PROGRAMA PRINCIPAL	47
63-PROCEDIMENTOS	48
64-Procedimentos: SUBROUTINE	48
65-Procedimentos: FUNCTION	49
66-EXERCÍCIO 11 – SUBROTINA	50
67-EXERCÍCIO 12 – FUNÇÃO	50
68-EXERCÍCIO 13 – PROCEDIMENTOS	51
69-SECÇÃO DE PROGRAMA: MODULE	52
70-EXERCÍCIO 14 – DEFINIÇÃO DE UM MÓDULO	53
71-EXERCÍCIO 15 – USO DE UM MÓDULO	53
ENTRADA E SAÍDA	
72-ENTRADA / SAÍDA	54
73-Comando OPEN	55
74-Comando READ	56
75-Comando WRITE	57
76-Comando FORMAT/FMT=	58
77-Descriores de Formatos	58
78-Outros comandos de I/O	58
79-Comando DATA	59
80-EXERCÍCIOS 15 – I/O	60
81-EXERCÍCIOS 16 – FORMATAÇÃO	60
82-REFERÊNCIAS	61

1-HISTÓRIA

- **FORmula TRANslation System;**

Primeira linguagem considerada de alto nível (“High Level” – próxima a linguagem humana), inventada por **John Backus**, na IBM, em 1954 e comercializada em 1957. Continua sendo muito utilizada nos dias de hoje nas áreas de programação científica 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 chamada de linguagem de máquina ou código de máquina, que na verdade, é a única linguagem que o computador entende, uma seqüência de 0s e 1s, interpretadas como instruções.

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”. 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 à primeira HLL.

O Fortran possui hoje, mais de 40 anos, constantemente sendo modificado e atualizado, e se mantém 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)

Era um Fortran II “limpo” e com implementação dos comandos COMMON e EQUIVALENCE, que permitiram o compartilhamento de código entre outros programas e sub-rotinas.

- **Fortran77** (1977)

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

- **Fortran90** (1980)

Levou 12 anos para ser atualizado. Os recursos do Fortran90 se equiparam aos existentes no C (Alocação dinâmica de memória, apontadores e orientação ao objeto).

- **HPF** (1990)

High Performance Fortran – Fortran90 para ambientes com memória distribuída;

- **Fortran95** (1995)

2-FORTRAN 77

O Fortran77 foi considerado obsoleto em relação às linguagens atuais e aos recursos existentes. Razões:

- Formato fixo:
 - Linhas da posição 7 a 72;
 - Somente letras maiúsculas;
 - Nomes até 6 caracteres.
- Impossibilidade de representar operações paralelas intrinsicas;

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 o paralelismo, como compartilhamento de endereços de memória.

- Não é possível a alocação de memória dinâmica;

No Fortran77, o programador é obrigado a declarar vetores com o maior tamanho possível para reservar memória durante a compilação.

- Não possui representação numérica portátil;

A precisão de campos numéricos variava de uma máquina para outra, tornando o código “não portátil”.

- Não possui definição de tipo de dado pelo programador;

Não é possível criar novos formatos a partir dos existentes.

- Não possui recursão explícita;

Não é possível chamar uma função dentro de outra função.

3-FORTRAN 90

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

- Definição de “ARRAYS” paralelos;

Novos recursos na definição de um “ARRAY”, permitem particionar vetores por entre vários processos que compartilham um ambiente de memória compartilhada.

- Alocação de memória dinâmica e apontadores;
- Definição de tipo de dados (Comando KIND);
- Recursividade

Além dos recursos descritos acima, vários outros, melhoraram o Fortran tornando-o mais atual aos recursos existente em outras linguagens:

- Controles de estruturas:
 - DO...ENDDO
 - DO...WHILE
 - SELECT CASE
- Substituição de comandos:
 - COMMON blocks → MODULE
 - EQUIVALENCE → TRANSFER
- Novos comandos:
 - IMPLICIT NONE

4-EXEMPLO DE PROGRAMA

```
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
```

5-RECOMENDAÇÕES DE CODIFICAÇÃO

- Sempre utilize o comando IMPLICIT NONE;
- Comandos, funções intrínsecas e as definidas pelo programador, devem se 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.

6-CARACTERÍSTICAS DE CODIFICAÇÃO

- 132 caracteres por linha;
- Alfanumérico: a-z, A-Z, 0-9, _
- ! Caractere de início de comentário;
- & Caractere de continuação de linha;
- ; Caractere de separação de comandos;
- Símbolos aritméticos:
 - + Adição
 - Subtração
 - * Multiplicação
 - / Divisão
 - ** Potenciação

7-REGRAS DE CODIFICAÇÃO

- “**Branco**s” não são permitidos:

- “Palavras-chave”

INTEGER :: nome1 Certo
INT EGER :: nome1 Errado

- “Nomes”

REAL :: valor_total Certo
REAL :: valor total Errado

- “**Branco**s” 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 ter até 31 caracteres
- Devem começar com letra

REAL :: a1 Certo
 REAL :: 1a Errado

- Podem continuar com letras, dígitos ou “_”

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

8-COMPILAÇÃO NO AMBIENTE CENAPAD-SP

- Ambiente IBM/AIX

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

Na verdade o compilador é um só (**xlf**) , mas com vários “scripts” de execução que possuem as opções de como o compilador deve ser executado, como:

- qlanglvl= Padrão de Fortran
- qsuffix= Sufixo dos programas
- qfree=yes/no Tipo de formatação

- Ambiente INTEL/Linux

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

- Opções básicas de compilação:

- o Nome do executável (Default: **a.out**);
- O, -O1, -O2, -O3 Otimização do código;
- c Não gera executável;
- g Gera informações para depuração;
- L<path> Localização da biblioteca para “linkedição”;
- l<biblioteca> Nome da biblioteca;
- q32 Código para 32bits (somente para AIX);
- q64 Código para 64bits (somente para AIX);

- Exemplos de compilações:

```
xlf prog1.f -o prog
```

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

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

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

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

9-EXERCÍCIO 1- Compilação e Execução

1 – Caminhe para o diretório ~/curso/fortran/ex1. Utilizando um editor de texto, edite o programa abaixo e salve-o com o nome **quadsol.f**

```
% cd ~/curso/fortran/ex1
% vi quadsol.f      OBS: Pode ser o editor 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              ! one 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", Part_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:

```
% xlf90 quadsol.f -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.f.

```
% cp quadsol.f novoquadsol.f
```

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. Você acha que isso é uma boa idéia?

10-EXERCÍCIO 2- Reestruturação de programa

1 – Caminhe para o diretório ~/curso/fortran/ex2. Reescreva o programa **basic_reform.f** de uma maneira que fique mais compreensível.

11-ESTRUTURA DE PROGRAMAS

O Fortran possui algumas regras bem definidas para ordem dos comandos:

1. Cabeçalho de definição:

PROGRAM, FUNCTION, SUBROUTINE, MODULE ou BLOCK DATA;

Só pode haver um único comando PROGRAM.

Pode haver mais de um FUNCTION, SUBROUTINE e MODULE.

Só pode haver um único BLOCK DATA

2. Comandos de Declaração:

REAL, INTEGER, IMPLICIT, PARAMETER, DATA;

3. Comandos de Execução:

IF-ENDIF, DO-ENDDO, comando de atribuição;

4. Finalização do programa com o comando END;

12-TIPOS DE DADOS

Todo tipo de dado possui um nome, um conjunto válido de valores, um significado dos valores e um conjunto de operadores.

- **Dado Caracter**

CHARACTER :: sex

CHARACTER(LEN=12) :: nome

- **Dado “Boolean”**

LOGICAL :: w

- **Dado Numérico**

REAL :: alt

DOUBLE PRECISION :: pi

INTEGER :: id

13-CONSTANTES

- Constante é um objeto com valor fixo

12345	Número Inteiro
-6.6E-06	Número Real
.FALSE.	Valor Lógico
“Curso Fortran”	Caractere

- Observações:

- Números **Reais** possuem ponto decimal ou o símbolo de expoente;
- 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 ‘

14-TIPO IMPLÍCITO

- Variáveis não declaradas possuem um tipo implícito de dado (Padrão Fortran77):
 - Se a primeira letra da variável começar por I, J, K, L, M ou N, será definida como **Inteiro**;
 - Qualquer outra letra, será do tipo **Real**;
- Tipo de dado implícito é potencialmente perigoso e deve ser evitado com a declaração:

IMPLICIT NONE

Exemplo de problema ocorrido no Fortran77:

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

15-DECLARAÇÃO NUMÉRICA E LÓGICA

- Com o comando IMPLICIT NONE, todas as variáveis devem ser declaradas, da forma:

<tipo> [, <lista de atributos>] :: <lista de variáveis> [=<valor>]

Em Fortran90 pode se definir atributos quando se declara uma variável.

Lista de atributos PARAMETER, DIMENSION;

:: Não é obrigatório, a menos que se especifique um atributo;

Exemplos:

```
REAL                                :: x
INTEGER                            :: i, j
LOGICAL                            :: ptr
REAL, DIMENSION(10,10)        :: y, z
INTEGER                            :: k=4
```

16 - DECLARAÇÃO CARACTERE

- As declarações de caracteres são similares as declarações numéricas. Pode se declarar um caractere ou um conjunto de caracteres:

<tipo>[(LEN=<tamanho>)] [, <lista de atributos>] :: <lista de variáveis> [=<valor>]

Exemplos:

```
CHARACTER(LEN=10)                        :: nome
CAHARCTER                                :: sexo
CHARACTER(LEN=32)                        :: str
CHARACTER(LEN=10), DIMENSION(10,10)    :: vetor
```


17-DECLARAÇÃO DE CONSTANTES

- Um valor constante, conhecido como parâmetro, pode ser definido pelos comandos de declaração utilizando o atributo **PARAMETER** ou pelo próprio comando **PARAMETER**;

Exemplos:

INTEGER pre Fortran77

PARAMETER (pre=252) Fortran77

REAL, PARAMETER :: pi=3.14159 Fortran90

CHARACTER(LEN=*), PARAMETER :: n1='Paulo', n2='Francisco' Fortran90

Valores caracteres podem assumir o seu próprio tamanho utilizando-se (LEN=*);

- É recomendado que se use a forma padrão do Fortran90, com atributos;
- O atributo PARAMETER deve ser usado, quando se tiver certeza que a variável só poderá assumir um único valor.

18-INICIALIZAÇÃO DE VARIÁVEIS

- Pode-se atribuir um valor inicial a uma variável e altera-la no decorrer da execução do programa.

Exemplos:

REAL :: x, y=1.005

INTEGER :: i=5, j=100

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

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

REAL, PARAMETER :: pi=3.14159

REAL :: radius=3.5

REAL :: circo=2*pi*radius (**expressão**)

OBS: Em geral, **funções não podem** ser utilizadas em expressões que inicializam uma variável.

19-EXERCÍCIO 3 - Declaração de Variáveis

1 – Caminhe para o diretório ~/curso/fortran/ex3, crie um programa em fortran90 (**variavel.f**) que apenas declare as seguintes variáveis:

Nome	Status	Tipo	Valor Inicial
Pe	Variável	Inteiro	Não possui
Milhas	Variável	Real	Não possui
Cidade	Variável	Caractere (até 20 letras)	Não possui
Local	Constante	Caractere	Campinas
Aonde_nasceu	Constante	Lógica	Falso
Seno_meio	Constante	Real	Sin(0.5)=0.47942554

20-EXPRESSÕES

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

Exemplos:

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;

21-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.

Exemplos:

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.

22-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 variáveis, constantes, escalares ou vetores, com exceção do expoente que necessariamente deve ser escalar.

23-Operadores RELACIONAIS

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

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

Exemplos:

a = i .GT. J

IF (i .EQ. J) c=d

24-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

Exemplos: 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.

25-Operador CARACTERE

- Utilizado para efetuar a concatenação “//”, somente de variáveis caracteres.

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

string(1:1) → 'a'
string(2:4) → 'bcd'

a=string//string(3:5) → 'abcdefghcde'
b=string(1:1)//string(2:4) → 'abcd'

26-Precedência de Operadores



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

Exemplos:

(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)

27-EXERCÍCIO 4 – Expressões

1-Caminhe para o diretório ~/curso/fortran/ex4. Edite o programa **area_circulo.f**

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-Área do Círculo: **area = $\mathbf{B} r^2$**

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

5-Compile e execute o programa. Verifique se ele executa corretamente para os valores de 2, 5, 10 e -1

28-COMANDOS DE CONTROLE DE FLUXO

- Toda linguagem de programação estruturada necessita de artifícios que possibilitem a execução condicional de comandos. Esses comandos normalmente alteram o fluxo de execução de um programa.

- Comandos de execução condicional: IF... , IF...THEN...ELSE...END IF

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.

- Comandos de iteração repetitiva: DO...END DO, DO WHILE...END DO

O comando DO permite a execução repetitiva de um bloco de comandos.

- Comandos de múltipla escolha: SELECT CASE

O comando SELECT permite a execução de comandos baseado no valor que uma expressão pode ter.

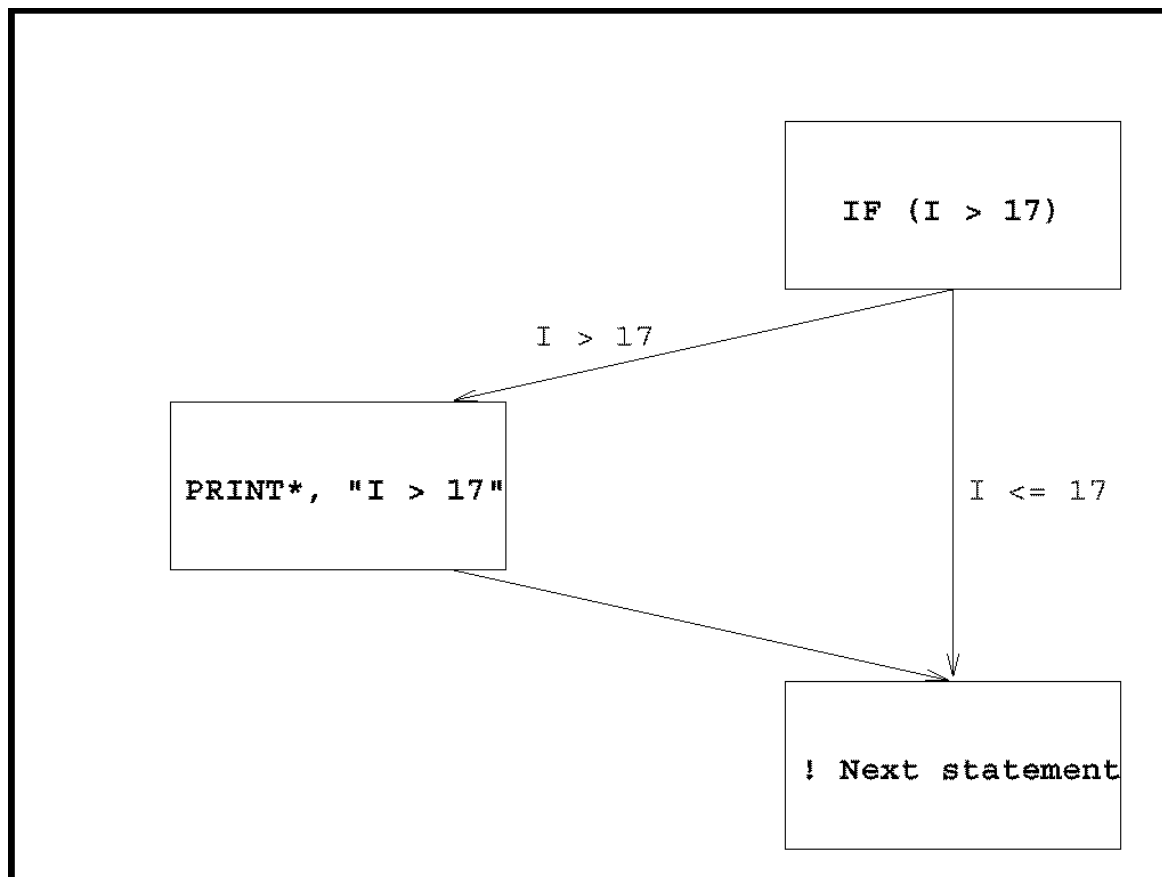
29-Comando IF

- Determina a execução de um único comando se uma condição lógica for verdadeira:

IF (<expressão lógica>) <comando>

Exemplos:

```
IF (I > 17) PRINT*, "I > 17"
```



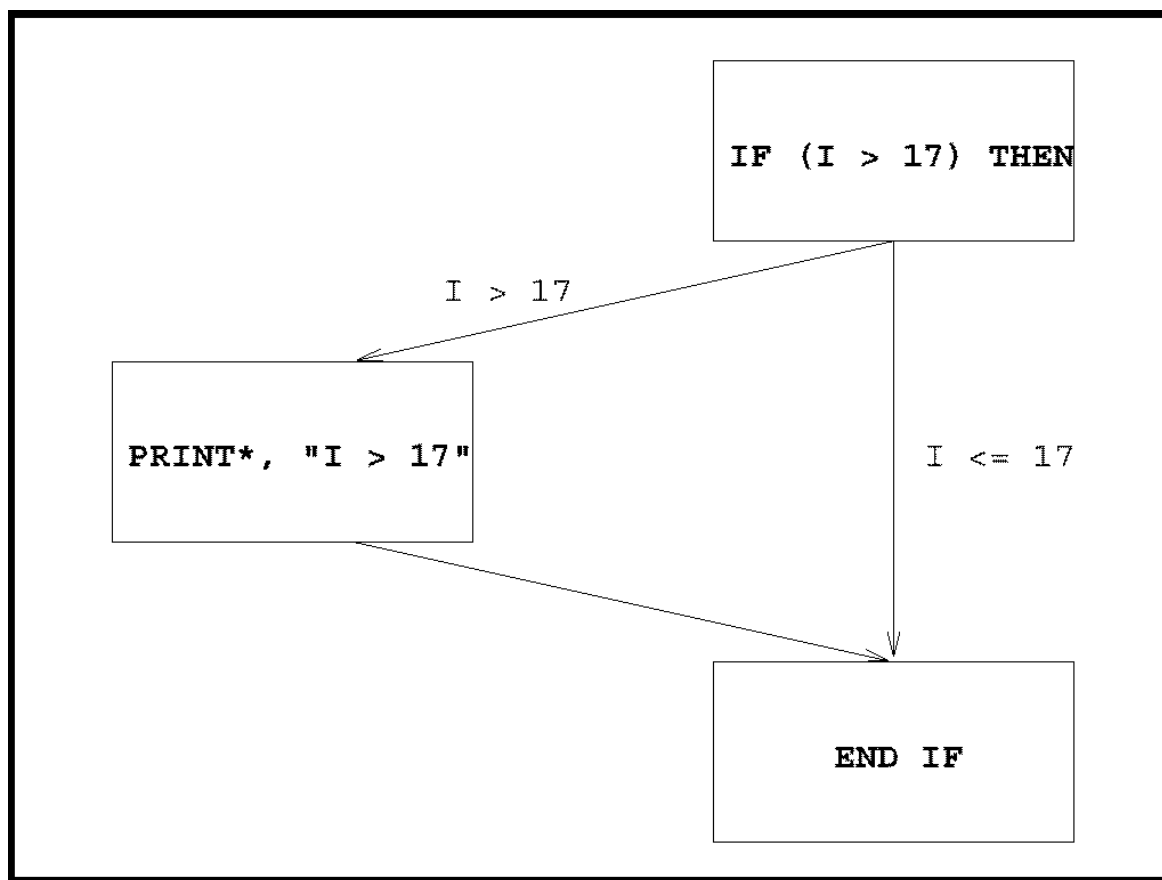
30-Comando IF...THEN...END IF

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

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

Exemplos:

```
IF (I > 17) THEN
  PRINT *, "I > 17"
END IF
```



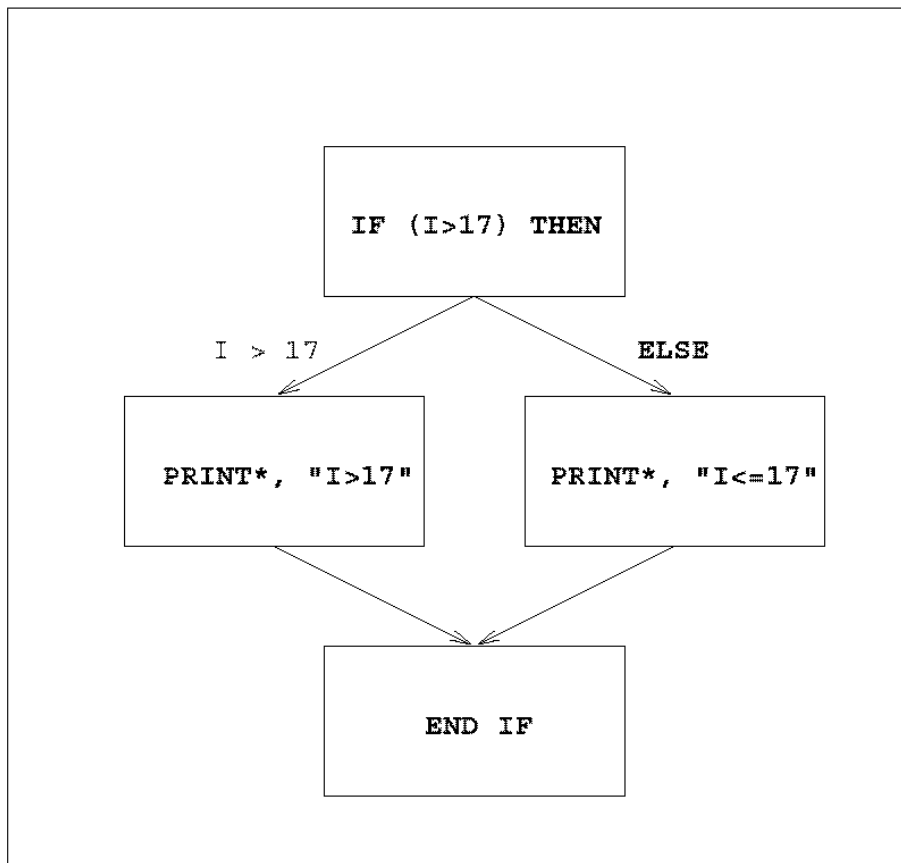
31-Comando IF...THEN...ELSE...END IF

- Determina a execução de um bloco de comandos se uma condição lógica for verdadeira ou falsa:

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

Exemplos:

```
IF (I > 17) THEN
  PRINT *, "I > 17"
ELSE
  PRINT *, "I <= 17"
END IF
```



32-Comando IF...THEN...ELSEIF...END IF

- Determina a execução recursiva de comandos se uma condição lógica for verdadeira ou falsa em vários blocos de IF's

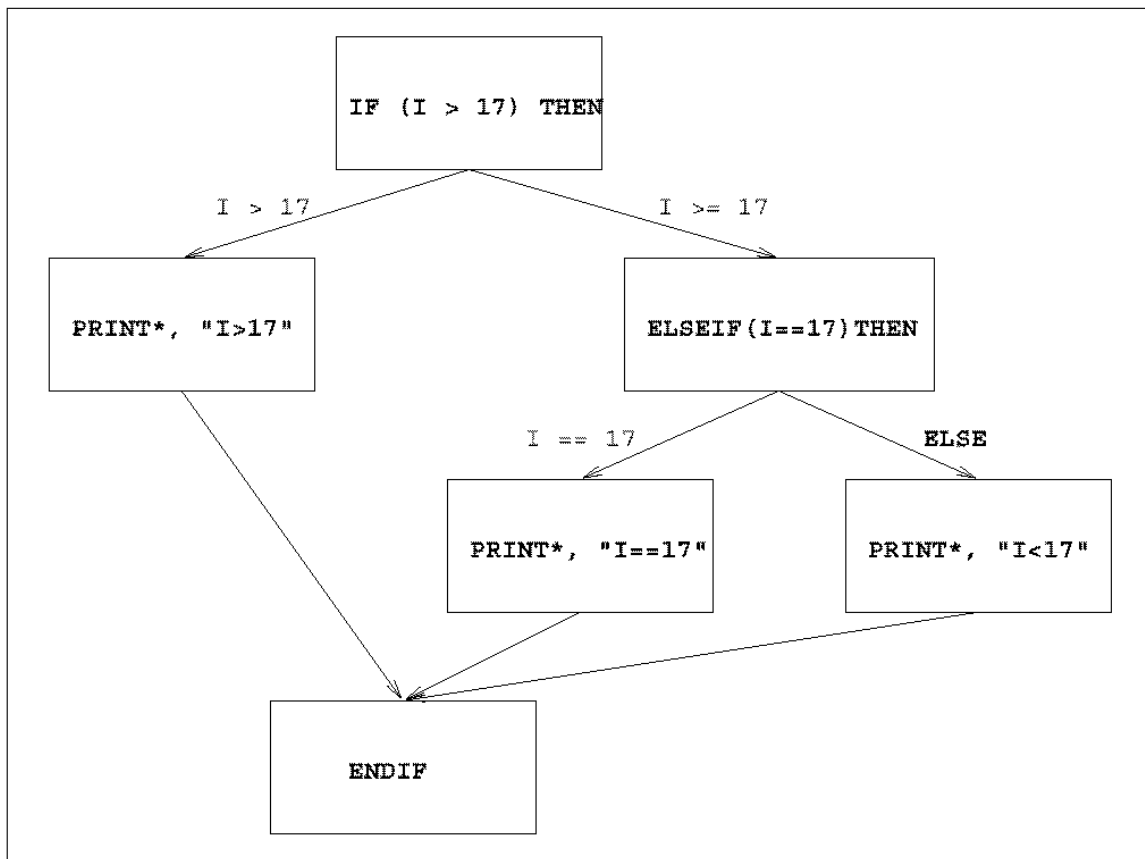
```

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

Exemplos:

```

IF (I > 17) THEN
    PRINT *, "I > 17"
ELSEIF (I == 17) THEN
    PRINT *, "I == 17"
ELSE
    PRINT*, "I < 17"
END IF
  
```



33-Comando IF...THEN...ELSEIF...END IF Identificado

- 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>

```

Exemplos:

```

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

```

34-EXERCÍCIO 5 - Comando IF

1 – Caminhe para o diretório ~/cursos/Fortran/ex5. Edite o programa **triangulo.f**

2 – Esse programa solicita que se digite três valores inteiros que poderão formar os três lados de um triângulo Equilátero, Isósceles, Escaleno, ou não formar um triângulo.

3 – 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.

$$(2*\text{MAX}(\text{lado1},\text{lado2},\text{lado3}) < \text{lado1}+\text{lado2}+\text{lado3})$$

4 – 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.

5 – Compile e execute o programa várias vezes entre com os seguintes valores:

- (1,1,1)
- (2,2,1)
- (1,1,0)
- (3,4,5)
- (3,2,1)
- (1,2,4)

35-Comando DO–EXIT-END DO “LOOP” Condicional

- “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 pelo comando **DO...END DO** e o comando **EXIT** determina a saída do “loop”.

```

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

```

Exemplos:

```

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

```

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

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

```

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

```

Exemplos:

```

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

```

37-“LOOPS” Identificados

- “loops” recursivos identificados.

Exemplo:

```

1  outa: DO
2  inna: DO
3      ...
4      IF (a .GT. b) THEN EXIT outa           ! Pula para linha 10
5      IF (a .EQ. b) THEN CYCLE outa         ! Pula para linha 1
6      IF (c .GT. d) THEN EXIT inna          ! Pula para linha 9
7      IF (c .EQ. a) THEN CYCLE              ! Pula para linha 2
8  END DO inna
9  END DO outa

```

38-Comando DO-WHILE

- “loops” que condiciona a sua execução antes de executar o bloco de comandos. “Faça enquanto”. A condição é testada no topo do “loop”.

```

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

```

Exemplo:

```

DO WHILE ( salario .LE. 5000 )
    salario=salario*1.05
END DO

```

39-Comando DO iterativo

- “loops” que possuem um número fixo de ciclos.

```

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

```

```

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

```

Exemplo:

```

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

```

40-Comando SELECT-CASE

- Construção similar ao IF, mas 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: A seleção pode ser representar **uma lista de valores**:

(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 maior ou igual a 100

Exemplo:

```

SELECT CASE (I)
  CASE (1)
    PRINT*, "I==1"
  CASE (2:9)
    PRINT*, "I>=2 and I<=9"
  CASE (10)
    PRINT*, "I>=10"
  CASE DEFAULT
    PRINT*, "I<=0"
END SELECT

```

41-DIVISÃO POR INTEIROS

- Ocorrem confusões em relação aos resultados quando da divisão de números inteiros. 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

42-EXERCÍCIO 6 – SELECT CASE

1 – Caminhe para o diretório ~curso/fortran/ex6. Edite o programa **ingresso.f** . 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

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.

43-PROCEDIMENTOS INTRÍNSECOS

- Em linguagens de programação, normalmente, algumas tarefas são executadas com muita frequência. O Fortran90 possui internamente, em torno de, 113 procedimentos, que são chamadas de funções intrínsecas e executadas como funções:
 - Matemáticas: Trigonométricas, Logarítmicas (SIN e LOG)
 - Numéricas (SUM, MAX)
 - Caracteres (INDEX, TRIM)
 - Transformação (REAL)

44-Funções de CONVERSÃO

- Transformação de tipos de dados

REAL(i)	Converte i para um aproximação de real;
INT(x)	Converte x para um equivalente inteiro;
DBLE(a)	Converte a para precisão dupla;
IACHAR(c)	Retorna o valor de c da tabela ASCII
ACHAR(i)	Identifica o valor i da tabela ASCII

REAL(1)	1.000000
INT(1.7)	1
INT(-0.9999)	0
IACHAR('C')	67
ACHAR(67)	C

45-Funções MATEMÁTICAS

- Algumas:

ACOS(x)	Arcocoseno
COS(x)	Coseno
COSH(x)	Coseno Hiperbólico
EXP(x)	e^x
LOG(x)	Logaritmo natural ou neperiano ln
LOG10(x)	Logaritmo base 10
SQRT(x)	Raiz Quadrada

46-Funções NUMÉRICAS

- Algumas:

ABS(a)	Valor absoluto
INT(a)	Valor inteiro
MAX(a1,a2,...)	Valor máximo
MIN(a1,a2,...)	Valor mínimo
MOD(a,p)	Resto da divisão a/p
REAL(a)	Converte a para REAL
DBLE(x)	Converte x para DOUBLE PRECISION

47-Funções CARACTERES

- Algumas:

ADJUSTL(str)	Alinha pela esquerda
ADJUSTR(str)	Alinha pela direita
LEN(str)	Tamanho do “string”
REPEAT(str,i)	Repete o “string” i vezes
TRIM(str)	Remove brancos a direita

48-EXERCÍCIO 7 – Funções Matemáticas

1 – Caminhe para o diretório ~/curso/fortran/ex7. Edite o programa **PontoNoCirculo.f** 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, **q** em graus.

Lembre-se:

$$q(\text{radianos}) = (q(\text{graus}) / 180) * P$$

$$P = \arctan(1) * 4$$

$$\text{seno}q = y / r$$

$$\text{coseno}q = x / r$$

49-Comando PRINT

- Comando que direciona um dado não formatado para a saída padrão.

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

* Substitui um formato

O comando PRINT sempre inicia uma nova linha.

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
```

50-Comando READ

- Comando que lê um dado não formatado da entrada padrão, o teclado.

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

* Substitui um formato

Exemplo:

```
READ*, nome
READ*, x, y, z
READ*, var1
```

MATRIZES

51-DEFINIÇÃO DE MATRIZES

- Matrizes ou “Arrays” é uma coleção de dados armazenados na memória e acessados individualmente de acordo com a sua posição espacial, definida pelas dimensões da matriz.

- Matriz de 1 Dimensão com 15 elementos:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
----------	----------	----------	----------	----------	----------	----------	----------	----------	-----------	-----------	-----------	-----------	-----------	-----------

- Matriz de 2 dimensões 15 elementos - 5 x 3:

1,1	1,2	1,3
2,1	2,2	2,3
3,1	3,2	3,3
4,1	4,2	4,3
5,1	5,2	5,3

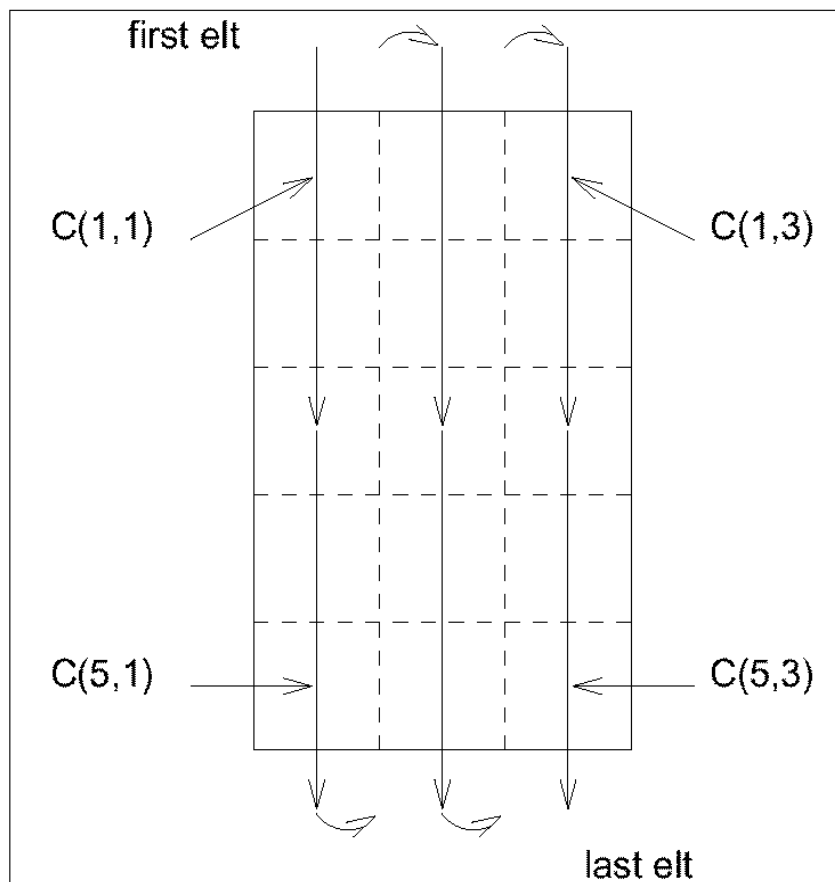
52-DECLARAÇÃO DE MATRIZES

```

REAL, DIMENSION(15)      :: X
REAL, DIMENSION(1:5,1:3) :: Y,Z
REAL                     :: T(5,3)

```

- **Dimensões:** até 7 dimensões;
- **Limites:** O limite inferior e superior ficam separados pelo caractere “:”. Caso não exista esse caractere, o limite inferior será sempre 1 e o limite superior, o informado na definição da matriz. Se não for informado, então a alocação de memória será dinâmica, ou seja, durante a execução do programa;
- **Tamanho:** O número de elementos de uma matriz é igual à multiplicação dos elementos em cada dimensão;
- **Organização:** O Fortran90 não especifica como as matrizes são dispostas em memória, mas para leitura e impressão, a ordem por coluna, será utilizada.



53-SÍNTAXE DE MATRIZES

- Iniciar os elementos de uma Matriz (/ ... /)

INTEGER, DIMENSION(4) :: mat = (/2,3,4,5/)
 REAL, DIMENSION(3,3) :: unida = (/1,0,0,0,1,0,0,0,1/)

- A ação se efetua em toda a Matriz:

A = 0.0
 B = C + D

- A ação se efetua em alguns elementos de uma Matriz:

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

- A ação se efetua para algumas secções de elementos de uma Matriz:

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

54-SECCÕES DE MATRIZES

[<limite1>:<limite2>][<incremento>]

A(:)	Toda matriz
A(3:9)	A(3) até A(9) de 1 em 1
A(8:3:-1)	A(8) até A(3) de -1 em -1
A(m:)	A(m) até limite superior
A(:,2)	Toda matriz de 2 em 2

55-IMPRESSÃO E LEITURA DE MATRIZES

- Considere A como uma matriz de duas dimensões

Impressão:

PRINT*, A

A(1,1), A(2,1), A(3,1),..., A(1,2), A(2,2), A(3,2),...

Leitura:

READ*, A

Deverá informar os dados na ordem acima (**ordem por colunas**)

Exemplos:

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 4 8 3 6 9

56-FUNÇÕES DE MATRIZES

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

- Algumas funções permitem questionar alguns atributos de matrizes.

- LBOUND(SOURCE[,DIM])

Identifica o limite **inferior** das dimensões de uma matriz

LBOUND(A) Resultado é uma matriz (/ -10,1,14/)

LBOUND(A,1) Resultado é um escalar -10

- UBOUND(SOURCE[,DIM])

Identifica o limite **superior** das dimensões de uma matriz

- SHAPE(SOURCE)

Identifica qual é a aparência de uma matriz

SHAPE(A) Resultado é uma matriz (/21,23,15/)

SHAPE((/4/)) Resultado é uma matriz (/1/)

- SIZE(SOURCE[,DIM])

Identifica o numero de elementos de uma dimensão da matriz

SIZE(A,1) 21

SIZE(A) 7245

57-ALOCAÇÃO DE MATRIZES

- O Fortran90 permite a alocação dinâmica de memória. Para isso será necessário utilizar os comandos **ALLOCATABLE**, **ALLOCATE**, **ALLOCATED** e **DEALLOCATE**.

- Na declaração das matrizes - **ALLOCATABLE** :

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

- Alocação de memória - **ALLOCATE** :

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

- Liberação de memória – **ALLOCATED** e **DEALLOCATE**

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

- OBS: O espaço de memória de uma matriz permanece alocado até ser efetuado um **DEALLOCATED** ou até o fim do programa

58-EXERCÍCIO 8 – DEFINIÇÃO DE MATRIZES

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 da matriz? _____

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

Quantas dimensões? _____

Limite(s) inferior(es)? _____

Limite(s) Superior(es)? _____

Tamanho da matriz? _____

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

Quantas dimensões? _____

Limite(s) inferior(es)? _____

Limite(s) Superior(es)? _____

Tamanho da matriz? _____

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

Quantas dimensões? _____

Limite(s) inferior(es)? _____

Limite(s) Superior(es)? _____

Tamanho da matriz? _____

OBS: A solução está no diretório ~/curso/fortran/ex8

59-EXERCÍCIO 9 – FUNÇÕES DE MATRIZES

1 – Dado a seguinte declaração de matriz:

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

2 – Escreva um pequeno programa no diretório ~/curso/fortran/ex9 , que possua algumas funções de matrizes que identifiquem:

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

OBS: A solução está no arquivo solucao.f. Tente fazer sem olhar!!!

60-EXERCÍCIO 10 – USO DE MATRIZES

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/ex10. Edite o programa **MatrizSalario.f** . 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.

SECCÕES PRINCIPAIS DE PROGRAMAS

61-SECCÕES DE UM PROGRAMA FORTRAN

- O Fortran90 possui duas secções principais de programa:

- **PROGRAM**

Secção principal do programa aonde a execução inicia e finaliza. Pode conter vários procedimentos.

- **MODULE**

Secção do programa que pode conter novas declarações e procedimentos, e que pode ser anexado ao programa principal.

- O Fortran90 possui dois tipos de procedimentos:

- **SUBROUTINE**

Um programa com parâmetros de entrada e saída que pode ser chamada de dentro do programa principal com o comando CALL

CALL relatorio(titulo)

- **FUNCTION**

Similar a uma SUBROUTINE, no entanto, retorna um único valor e pode ser executado de dentro de um comando.

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

62-PROGRAMA PRINCIPAL

```

PROGRAM principal
    ! Comandos de declaração
    ! Comandos executáveis
CONTAINS
    SUBROUTINE sub1(...)
        !Comandos executáveis
    END SUBROUTINE sub1
    ! ...
    FUNCTION func(...)
        !Comandos executáveis
    END FUNCTION func
END PROGRAM principal

```

- O comando PROGRAM é opcional, assim como o nome do programa, mas é uma boa prática sempre usá-los.
- O programa principal pode conter declarações, comandos executáveis e procedimentos internos: Sub-rotinas e funções definidas pelo usuário. Esses procedimentos são separados do resto do programa pelo comando CONTAINS.

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

```

63-PROCEDIMENTOS

- Internos: 113 do Fortran90

- Bibliotecas: **NAG Numerical Library, 300+**
 BLAS – Basic Linear Algebra Subroutine
 IMSL
 LAPACK
 SCALAPACK
 Uniras

64-Procédimentos: SUBROUTINE

SUBROUTINE <nome> [(<argumentos>)]
 Declaração dos argumentos “dummy”
 Declaração dos objetos
 Comandos executáveis
END [SUBROUTINE [<nome>]]

- Para se definir uma Sub-rotina usa-se a estrutura SUBROUTINE – END SUBROUTINE;
- Para se usar uma Sub-rotina usa-se o comando CALL;
- Uma Sub-rotina pode ‘enxergar’ todas as variáveis declaradas no programa principal;
- Uma Sub-rotina pode incluir a chamadas a outras sub-rotinas

Exemplo:

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


65-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 de Sub-rotina, com a diferença de que a função retorna um valor;
- 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.

Exemplo:

```

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 FUNTION F
END PROGRAM algo

```

66-EXERCÍCIO 11 – SUBROTINA

1 – Caminhe para o diretório ~/curso/Fortran/ex11. Edite o arquivo **Subrotina.f** e adicione a sub-rotina de acordo com a descrição do problema.

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

Sub-rotina 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

67-EXERCÍCIO 12 – FUNÇÃO

1 – Caminhe para o diretório ~/curso/Fortran/ex12. Edite o arquivo **funcao.f** 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

68-EXERCÍCIO 13 – PROCEDIMENTOS

1 – Caminhe para o diretório ~/curso/Fortran/ex13. Edite o arquivo **NumeroRandomico.f** e substitua pelo comando adequado nas reticências.

Esse programa chama uma função que por sua vez chama a duas vezes a sub-rotina do Fortran90, RANDOM_NUMBER(r), para obter um número randômico e simular um arremesso de dados.

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

69-SECÇÃO DE PROGRAMA: MODULE

MODULE é uma secção de programa Fortran90, independente do programa principal, podendo ser compilado e utilizado por diversos programas como se fosse uma sub-rotina externa;

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

Define classes em orientação a objetos.

- Sintaxe:

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

Exemplo: Declara as variáveis X, Y e Z como variáveis reais globais.

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

Para se utilizar um MODULE usa-se o comando USE

```
PROGRAM teste  
    USE <nome>  
    IMPLICIT NONE  
    ...  
END PROGRAM teste
```

70-EXERCÍCIO 14 – DEFINIÇÃO DE UM MÓDULO

1 – Caminhe para o diretório ~/curso/Fortran/ex14. Edite o programa **DefModulo.f**

Esse programa cria um módulo que possui duas definições de funções que calculam a média e 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:

```
%xlf90 -c DefModulo.f
```

71-EXERCÍCIO 15 – USO DE UM MÓDULO

1 – Caminhe para o diretório ~/curso/Fortran/ex15. Edite o programa **UsoModulo.f**

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

3 - Compile o programa apenas gerando o objeto:

```
%xlf90 -c UsoModulo.f
```

4 – Copie o objeto e o módulo gerado no exercício anterior para este diretório. Crie o executável “linkeditando” os dois objetos:

```
%xlf90 -o Teste_modulo UsoModulo.o DefModulo.o
```

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
```

ENTRADA E SAÍDA

72-ENTRADA / SAÍDA

O Fortran90 possui uma grande variedade de opções de I/O, que permitem diferentes tipos de arquivos se conectarem ao programa principal para leitura e gravação.

Em Fortran90, um arquivo é conectado a uma *unidade lógica* definida por um número.

Cada unidade pode possuir diversas propriedades:

Arquivo	Nome do arquivo conectado;
Ação	Leitura, Gravação, Leitura e Gravação;
Status	old, new, replace;
Método de acesso	Sequencial ou Direto;

73-Comando OPEN

**OPEN([UNIT=]<inteiro>, &
 FILE=<arquivo>, ERR=<rótulo>, &
 STATUS=<status>, ACCESS=<método>, &
 ACTION=<modo>, RECL=<expressão>)**

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

UNIT= Especifica um número de referência ao nome do arquivo;

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

ERR= Especifica uma posição lógica de controle caso haja algum erro na abertura do arquivo;

STATUS= Especifica (entre aspas) 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;

ACCESS= Especifica (entre aspas) o método de acesso:

‘DIRECT’ Acesso direto a registros individuais. É obrigado a usar a opção RECL;

‘SEQUENTIAL’ Acesso seqüencial, linha por linha;

ACTION= Especifica (entre aspas) o modo de acesso ao arquivo:

‘READ’ Somente leitura;

‘WRITE’ Somente gravação;

‘READWRITE’ Leitura e gravação;

RECL= Especifica o tamanho máximo de um registro aberto para acesso direto.

Exemplo:

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

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

74-Comando READ

**READ([UNIT=]<inteiro>, [FMT]=<formato>,&
 IOSTAT=<int-var>, ERR=<rótulo>, &
 END=<rótulo>, EOR=<label>, &
 ADVANCE=<modo>, REC=<expressão>,&
 SIZE=<num-caracteres>) <lista de variáveis>**

- No caso do comando de leitura, algumas características de conexão não podem ser utilizadas juntas.

UNIT= Especifica um número de referência a unidade de leitura (* representa a unidade default);

FMT= Especifica (entre aspas) o formato da leitura dos dados;

ERR= Especifica uma posição lógica de controle caso haja algum erro de leitura;

IOSTAT= Código de retorno. Zero significa sem erros;

END= Especifica uma posição lógica de controle caso ocorra erro de fim de arquivo;

EOR= Especifica uma posição lógica de controle caso ocorra erro de fim de registro;

REC= Especifica o número do registro que deve ser lido no modo de acesso direto;

ADVANCE= Especifica 'YES' ou 'NO', se deve ou não iniciar a leitura em um novo registro;

SIZE= Retorna par uma variável o número de caracteres lidos

Exemplo:

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

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

75-Comando WRITE

**WRITE([UNIT=]<inteiro>, [FMT]=<formato>,&
IOSTAT=<int-var>, ERR=<rótulo>, &
ADVANCE=<modo>, &
REC=<expressão>) <lista de variáveis>**

- No caso do comando de gravação, algumas características de conexão não podem ser utilizadas juntas.

UNIT= Especifica um número de referência a unidade de gravação (* unidade default);

FMT= Especifica (entre aspas) o formato de gravação dos dados;

ERR= Especifica uma posição lógica de controle caso haja algum erro de gravação;

IOSTAT= Código de retorno. Zero significa sem erros;

REC= Especifica o número do registro que deve ser lido no modo de acesso direto;

ADVANCE= Especifica 'YES' ou 'NO', se deve ou não iniciar a leitura em um novo registro;

Exemplo:

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

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

76 - Comando FORMAT/FMT=

Comando que especifica o formato na qual os dados serão lidos ou gravados.

Exemplo:

```
WRITE(17,FMT='(2X,2I4,1X,''nome'',A7)') i, j, str
```

```
READ(14,*) x,y
```

```
WRITE(*, FMT=10) a, b
10 FORMAT('vals',2(F15.6,2X))
```

```
11-195 nome Felipe
Vals      -1.051330  333356.000033
```

77-Descriores de Formatos

- Iw w número de dígitos **inteiro**s
- Fw.d w número de dígitos **reais** e d número de decimais
- Ew.d w dígitos reais e d decimais em **notação científica**
- Lw w número de caracteres **lógicos**
- A[w] w numero de **caracteres**
- nX **pula** n espaços

78-Outros comandos de I/O

- CLOSE Fecha um arquivo
- REWIND Re-posiciona a leitura no primeiro registro
- BACKSPACE Volta a leitura em um registro
- ENDFILE Força a gravação de uma marca de fim de arquivo

79-Comando DATA

Comando que permite iniciar uma lista de variáveis.

DATA <lista1> /<dados1>/, <lista2> /<dados2>/,...<listan> /<dadosn>/

Exemplo:

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./

80-EXERCICIOS 16 – I/O

1 - Caminhe para o diretório ~/curso/Fortran/ex16. 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.

81-EXERCICIOS 17 – 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.)

```
bbb5.34bbbNOb.TRUE.
5.34bbbbbbYbbFbbbb
b6bbbbbb3211bbbbbbT
bbbbbbbbbbbbbbbbbbF
```

2 - Caminhe para o diretório ~/curso/Fortran/ex17. 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).

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

82-REFERÊNCIAS

- 1 – IBM XL Fortran 8.1 User's Guide
- 2 – IBM XL Fortran 8.1 Language Reference
- 3 – INTEL Fortran 9 Language Reference
- 4 – The University of Liverpool – Fortran 90 Programming
Dr. A.C. Marshall