HISTÓRICO

- FORTRAN: 'Formula Translation'
- início - Surgiu na segunda metade da década de 50.
- uso - A linguagem Fortran foi criada para facilitar o uso pela comunidade técnico/científica com escrita semelhante a matemática.
- Foi a primeira linguagem de alto nível com várias modificações para aumentar a eficiência dos cálculos e oferecer maior número de ferramentas.
- 1966 - Primeira grande modificação feita pela ANSI surgindo o FORTRAN IV
- 1977 - Segunda grande modificação, ou seja, foi padronizada utilizando o conceito de programação estruturada, surgindo assim, FORTRAN77.
- 1990 - Surgiu um FORTRAN com uma nova padronização surgindo o FORTRAN90.

RECURSOS DO FORTRAN 90

- formato livre e nomes de variáveis mais flexíveis
- memória dinâmica (e ponteiros)
- tipos derivados (definidos pelo usuário)
- módulos: classes - definições globais (tipos, objetos, operadores, subrotinas) e locais;
- novas operações envolvendo conjuntos ou partes de conjuntos - paralelismo
- funções intrínsecas novas
- novas construções de controle
- interfaces (comunicação eficiente e mais segura entre subprogramas)
- Novos recursos de entrada e saída
- recursividade

- portabilidade numérica enriquecida
- definição e/ou extensão de operadores pelo usuário

COMANDOS FORTRAN

- PROGRAM - permite atribuir um nome a rotina principal do programa FORTRAN. Este comando deverá aparecer somente no começo do programa.

Exemplo: PROGRAM teste

- PRINT - permite fazer uma interação de saída padrão do computador, isto é, permite que o programa imprima na tela as frases ou dados, digitados ou calculados.

Exemplo: PRINT *, ' Oi '
PRINT *, ' idade=' , 20

- READ - transfere os dados de uma unidade de entrada, como por exemplo a tela do computador, para a memoria do computador.

Exemplo: READ *, idade

- STOP - provoca uma parada na execução do programa e pode ser utilizado da seguinte forma:

  STOP
  STOP *n* em que n é um número que identifica o STOP acionado.
  STOP *mensagem* faz o mesmo que n , porém escreve uma mensagem.

- END - indica ao compitador o encerramento da unidade de um programa fonte. Portanto só pode existir um comando END e no final do programa.

EXEMPLO1.f

```
 PROGRAM Prog1
!Data: 23 de abril de 2001!
PRINT *, 'Meu primeiro programa de Fortran'
END
```

PARA COMPILAR

Para compilar um programa Fortran no ambiente com compilador da IBM.

- Fortran 77

xlf -o <nome do executavel> <nome do programa.f>

- Fortran 90

xlf90 -o <nome do executavel> <nome do programa.f90>

OPERADORES

Existem 2 tipos de operadores e 1 tipo de expressões: os operadores aritméticos, os lógicos e as expressões lógicas.

OPERADORES ARITMÉTICOS	*SÍMBOLO*
Adição	+
Subtração	-
Multiplicação	*
Divisão	/
Exponenciação	**

Exemplo: a = b + c

HIERARQUIA DOS OPERADORES ARITMÉTICOS

Todas as subexpressões entre parênteses são efetuadas em primeiro lugar. Em seguida as operações de divisão e multiplicação.

OPERADORES LÓGICOS	*SÍMBOLO*
Maior	.GT.
Menor	.LT.
Maior ou igual	.GE.
Menor ou igual	.LE.
Igual	.EQ.
Não igual	.NE.

Exemplo: 'resposta' .EQ. 'sim'
a .GT. b

Em FORTRAN 90, estes mesmos OPERADORES LÓGICOS podem ser representados por símbolos como indicado na tabela a seguir:

OPERADORES LÓGICOS(em FORTRAN)	*SÍMBOLO*
Maior	$>$
Menor	$<$
Maior ou igual	$>=$
Menor ou igual	$<=$
Igual	$==$
Não igual	$/=$

Exemplo:'resposta' == 'sim'
a $>$ b

EXPRESSÕES LÓGICOS	*SÍMBOLO*
Negação Lógica	.NOT.
Conjunção Lógica	.AND.
Disjunção Inclusivo Lógica	.OR.
Equivalência Lógica	.EQV.
Não Equivalência Lógica	.NEQV.

Exemplo:(TEMPERATURA .GT. 90) .AND. (HUMIDADE .GT. 0.90)
(MINIMO .LE. X) .OR. (X .LE. MAXIMO)

EXEMPLO2.f

```
 PROGRAM Prog2
 PRINT *, 'data: 23 de abril de 2001'
 !programa para somar!
 a=2**0+2**1+2**2+2**3+2**4+
& 2**5
 PRINT *,'soma = ',a
 STOP ' o programa Prog2 acabou'
 END
```

VARIÁVEIS

Regras para definição de Variáveis

As regras para definir as variáveis são as seguintes:
1-Os nomes das variáveis devem começar com uma letra.
2-As combinações de letras e números são permitidas.

Tipos de Variáveis

- *Inteiras* - são os números inteiros positivos e negativos
- *Reais* - são os números decimais positivos e negativos
- *Dupla Precisão* - são os números reais com mais casa decimais
- *Complexas* - são os números complexos
- *Lógicas* - são os valores verdadeiro .TRUE. e falso .FALSE.
- *Caracteres* - são as palavras ou expressões

DECLARAÇÃO DE TIPOS DE VARIÁVEIS

Pré-definição do FORTRAN 77

- As variáveis que iniciam com as letras de A-H e de O-Z serão do tipo *REAL*
- As variáveis iniciadas com as letras I,J,K,L,M, e N serão do tipo *INTEIRO*

IMPLICIT

O comando IMPLICIT colocado antes dos outros comandos de um programa indica quais as letras foram atribuídas para cada tipo de variável.

```
EXEMPLO:IMPLICIT INTEGER(G, I-K)
        IMPLICIT REAL(H, O-Y)
        IMPLICIT COMPLEX(Z)
        IMPLICIT LOGICAL(L)
```

Explicitamente através de comandos

- INTEGER - variáveis inteiras
- REAL - variáveis reais
- DOUBLE PRECISION - variáveis de dupla precisão
- COMPLEX - variáveis complexas
- LOGICAL - variáveis lógicas
- CHARACTER - variáveis caracteres
- CHARACTER*w - variáveis a serem declaradas como caracter de tamanho w.

Mesmo tendo definido implicitamente as variáveis é possível defini-las explicitamente. Para melhor compreensão do programa recomenda-se definir todas as variáveis.

PARAMETER

Para atribuir um nome a uma constante utiliza-se a declaração *PARAMETER*. Desta declaração deve preceder os comandos executáveis do FORTRAN.

EXEMPLO:REAL pi
PARAMETER(pi=3.1415927)

EXEMPLO3.f

```
 PROGRAM Prog3
!programa de divisão de números inteiros!
IMPLICIT NONE
REAL a,b,c
PRINT *, 'entre com um número'
READ *,a
PRINT *,'entre com outro número'
READ *,b
IF (a .EQ. b) THEN
PRINT *,'o resultado da divisão é 1'
ELSE
c=a/b
PRINT *,'o resultado da divisão é', c
END IF
END
```

Tabela 1: FUNÇÕES INTRÍNSECAS

F.INTRÍN.	*OPERAÇÕES*	*EXEMPLO*
INT(x)	converte a variável x p/ tipo inteiro	$INT(14.8) = 14$
REAL(x)	converte a variável x p/ tipo real	$REAL(4) = 4.0$
DBLE(x)	converte a variável x p/ tipo dupla precisão	$DBLE(1) = 1.0D0$
NINT(x)	converte a variável x p/ o inteiro mais próximo	$NINT(3.8) = 4$
CHAR(x)	converte a variável x inteira p/ caracter	$CHAR(65) = A$
ICHAR(x)	converte a variável x caracter p/ inteira	$ICHAR('A') = 65$
IABS(x)	retorna o valor absoluto de uma variável inteira	$IABS(-3) = +3$
$ALOG(x)$	calcula o logarítmo natural	$ALOG(2.0) = 4.605170186$
ABS(x)	retorna o valor absoluto de uma variável real	$ABS(-14.8) = +14.8$
MOD(a,b)	mostra o resto da divisão entre a e b	$MOD(3.3, 2.0) = 1.3$
MAX(args)	retorna o maior valor de um conj. de variáveis inteiras	$MAX(1, 2, -5, 4) = 4$

continua na próxima página

Tabela 1: FUNÇÕES INTRÍNSECAS (continuação)

F.INTRÍN.	*OPERAÇÕES*	*EXEMPLO*
MIN(args)	retorna o menor valor de um conj. de variaveis inteiras	$MIN(1, 2, -5, 4) = -5$
AMAX1(args)	retorna o maior valor de um conj. de variáveis reais	$AMAX1(1.0, 2.1, -5.2) = 2.1$
AMIN1(args)	retorna o menor valor de um conj. de variáveis reais	$AMIN(1.0, 2.1, -5.2) = -5.2$
AIMAG(imcomp)	retorna a parte imaginária de um complexo	$AIMAG((1.3, 3.4)) = 3.4$
CONJG(imcom)	retorna o complexo conjugado	$CONJG((1.3, 3.4)) = (1.3, -3.4)$
$SQRT(x)$	calcula a raiz quadrada da variável x	$SQRT(9.0) = 3.0$
$EXP(x)$	calcula o exponencial de e^x	$EXP(1.0) = 2.718281746$
$LOG(x)$	calcula o logarítmo neperiano de x,$ln(x)$	$LOG(2.718281746) = 1.0$
$LOG10(x)$	calcula o logarítmo na base 10 de x	$LOG10(100.0) = 2.0$
$ALOG10(x)$	calcula o logarítmo na base 10 de x	$ALOG10(100.0) = 2.0$
LEN(x)	retorna o tamanho da variável do tipo caracter	LEN('ABCD')=4
INDEX(x,n)	retorna a posição de uma subcadeia	$INDEX('ABCD', 'C') = 3$

continua na próxima página

Tabela 1: FUNÇÕES INTRÍNSECAS (continuação)

F.INTRÍN.	*OPERAÇÕES*	*EXEMPLO*
$SIN(x)$	retorna o valor de seno para um ângulo x em radianos	$SIN(\pi/2) = 1.0$
$COS(x)$	retorna o valor de coseno para um angulo x em radianos	$COS(\pi/2) = 0.0$
$TAN(x)$	retorna o valor da tangente de um ângulo x	$TAN(\pi/4) = 1.0$
$ASIN(x)$	esta função é a inversa da função seno	$ASIN(0.0) = 0.0$
$ACOS(x)$	esta função é a inversa da função coseno	$ACOS(0.0) = \pi/2$
$ATAN(x)$	esta função é a inversa da função tangente	$ATAN(1.0) = \pi/4$
$SINH(x)$	retorna o seno hiperbólico do ângulo x	–
$COSH(x)$	retorna o coseno hiperbólico do ângulo x	–
$TANH(x)$	retorna tangente hiperbólica do ângulo x	–

HIERARQUIA DAS FUNÇÕES, EXPRESSÕES E OPERADORES

Como nas expressões matemáticas, em FORTRAN temos uma prioridade a ser seguida quanto a ordem de resolução de uma expressão. A ordem é:
1. Funções.
2. Exponenciação.
3. Multiplicação e Divisão.
4. Da esquerda para a direita.

Existem algumas exceções como no caso de:
Exemplo: a**b**c $\Rightarrow$ a**(b **c)

Para que não haja dúvidas, é aconselhável o uso de *parênteses* tomando-se o cuidado de utilizar o mesmo número destes abertos e fechados.

NÃO DE DEVE MISTURAR OS TIPOS DE VARIÁVEIS DENTRO DE UMA EXPRESSÃO MATEMÁTICA. Caso isso ocorra, a precedência nas operações de multiplicação, divisão, soma e subtração será:

Inteiro $<$ Real $<$ Dupla Precisão $<$ Complexos

Assim sendo, a mistura dos tipos de variáveis numa expressão provoca o aparecimento de resultados não desejados como mostram os exemplos a seguir

Exemplo: $1/3 + 1/3 + 1/3 = 0$
$16 * *(1/2) = 1$

Exemplo: (TEMPERATURA .GT. 90) .AND. (HUMIDADE .GT. 0.90)
(MINIMO .LE. X) .OR. (X .LE. MAXIMO)

EXEMPLO4.f - Funções intrínsecas

```
 PROGRAM FUNCOES
IMPLICIT NONE
REAL x, y
PRINT *,'Entre com um número real'
READ *, x
y=SQRT(x)
PRINT *,'a raiz quadrada do numero dado é ', y
STOP
END
```

EXEMPLO5.f - Expressões e operadores lógicos

```
 PROGRAM IMPOSTO
IMPLICIT NONE
REAL salario, dependentes
PRINT *, 'Quanto é o seu salário?, R$'
READ *,salario
PRINT *,'Quantos dependentes voce tem?'
READ *, dependentes
IF ((salario .LE. 600.00) .OR. (dependentes .GT. 5)) THEN
PRINT *,'voce é isento do imposto'
ELSE
PRINT *,'voce nao e isento'
END IF
END
```

PROGRAMAÇÃO ESTRUTURADA E ESTRUTURAS DE CONTROLE

Programação estruturada é uma forma de programar utilizado para facilitar a leitura e compreensão dos programas além de apresentar menor tendência a erros.

ESTRUTURAS DE CONTROLE

Estrutura de Controle é um grupo de blocos,conjunto de linhas de comando, que é relacionada com um ponto de entrada e outro de saída. Há 3 categorias de estruturas de controle: a sequencial, a seletiva e a interativa.

- *SEQUENCIAL* - As linhas de comando são organizados em blocos que são executados como listado. Por exemplo, se o programa FORTRAN consistir de um grupo de blocos

Bloco1
Bloco2
Bloco3
⋮
Bloco n

Então, o *Bloco1* será executado antes do *Bloco2*, e assim por diante até o *Bloco n*.

- *SELETIVA* - Quando um algorítmo deve selecionar uma de várias alternativas dependendo dos dados do input. Como mostrado no EXEMPLO3, possui a estrutura *IF-THEN-ELSE*

IF(condição)THEN
Bloco1
ELSE
Bloco2
END IF

- *INTERATIVA* - Esta estrutura provoca a execução de um programa ou uma parte dela mais de uma vez.

nIF(condição)THEN
Bloco1
GO TO n
END IF
ou então:
nBloco1
IF(condição) GO TO n
END IF
O n é o número da linha que deve ser colocado entre as colunas 2 e 5.

GO TO

GO TO *n*

Este comando transfere o fluxo de um programa para o comando que estiver referenciado pela linha n. Este comando provoca uma transferência incondicional para a linha n e portanto não é recomendado para o uso pois desestrutura o programa.
EXEMPLO: GO TO 100

GO TO $(n_1, n_2, \ldots, n_m)$**, i**

Este comando de transferência permite controlar a execução do programa dependendo do valor de i.
Exemplo: GO TO(2,5,7,13),k
se k=1, transfere a execução para a linha 2.
se k=2, transfere a execução para a linha 5.
se k=3, transfere a execução para a linha 7.
se k=4, transfere a execução para a linha 13.

ESTRUTURA DO *IF*

O *IF* é utilizado quando queremos optar por várias alternativas dependendo do resultado ou do dado fornecido. Há 2 formas de sintaxe mais comuns.

IF com uma alternativa

IF (condição) THEN
Bloco do programa A
END IF

Neste caso, se a condição for verdadeira, o programa A é executado. Caso a condição seja falsa, ele pula e prossegue com o programa.

IF com duas alternativas

nIF(condição)THEN
Bloco1
ELSE
Bloco2
END IF

Aqui, se a condição for verdadeira o Bloco1 é executado, se não, o Bloco2 é que será executado.

OUTRAS FORMAS DO COMANDO IF

IF(condição)n_1, n_2, n_3

Neste caso, o comando IF faz um tipo de transferência condicional pois, dependendo do valor da condição este comando direciona o programa para n_1, n_2, n_3 que indicam os números das linhas de outras declarações.
se a condição < 0, o fluxo será transferido para n_1
se a condição $= 0$, o fluxo será transferido para n_2
se a condição > 0, o fluxo será transferido para n_3

IF expandida

*n*IF(condição1)THEN
Bloco1
ELSE IF (condição2) THEN
Bloco2
⋮
ELSE IF (condição *n*) THEN
Bloco*n*
ELSE
Bloco (*n+1*)
END IF

EXEMPLO6.f

```
PROGRAM RESULTADONOTA
IMPLICIT NONE
REAL nota
PRINT *, 'Qual foi a nota da sua prova?'
READ *, nota
IF (nota .GE. 9.00) THEN
PRINT *,'muito bem!'
ELSE IF (nota .GE. 7.00) THEN
PRINT *,'voce foi bem!'
ELSE IF (nota .GE. 5.00) THEN
PRINT *,'voce tem que melhorar um pouco!'
ELSE
PRINT *,'voce tem que estudar bastante se quiser passar!'
END IF
END
```

Neste EXEMPLO 6, podemos ver que a variável nota começou com o maior valor que é maior ou igual a 9.00 e não com o inverso utilizando o menor valor, maior ou igual a 5.00. Pois se começasse com 5.00 e a variável nota fornecida fosse 9.00, por exemplo, este satisfazeria todos os IFs pois é maior que 5.00, maior que 7.00 e igual a 9.00). Portanto, ao utilizar o comando do IF na

forma expandida, deve tomar muito cuidado com o valor e as condições que serão impostas em primeiro lugar para que nao ocorra erros desta natureza.

IF denominado

O IF denominado é muito parecido com o IF expandido. A diferença é que em uma das opções do comando ELSE IF podemos colocar um nome que será utilizado para ser invocado posteriormente por um comando ELSE. O comando IF denominado segue a seguinte sintaxe.

*n*IF(condição1)THEN
Bloco1
ELSE IF (condição2) THEN *nome*
Bloco2
ELSE *nome*
Bloco3
END IF

EXEMPLO7.f

```
PROGRAM RESULTADONOTA
IMPLICIT NONE
REAL nota
PRINT *, 'Qual foi a nota da sua prova?'
READ *, nota
IF (nota .GE. 9.00) THEN
PRINT *,'muito bem!'
ELSE IF (nota .GE. 7.00) THEN
PRINT *,'voce foi bem!'
ELSE IF (nota .GE. 5.00) THEN ESTUDAR
PRINT *,'voce tem que melhorar um pouco!'
ELSE ESTUDAR
PRINT *,'voce tem que estudar bastante se quiser passar!'
END IF
END
```

SELECT CASE

Além do comando IF, temos uma outra forma de especificar a seleção das alternativas que o fluxo de execução do programa deve seguir. O comando SELECT CASE possui a seguinte sintaxe:

SELECT CASE (variável seletora)
CASE (condição1)
Bloco1
CASE (condição2)
Bloco2
CASE DEFAULT
Bloco3
END SELECT

EXEMPLO8-1.f

```
 PROGRAM INGRESSOCONCERTO
!Num concerto, os preços dependendo da numeração das cadeiras são!
!cadeiras                preços!
!50                       50.00!
!100 - 140 e 200 - 240    25.00!
!300 - 340                20.00!
!400 - 440                15.00!
!este programa fornece o preço dos ingressos!
IMPLICIT NONE
REAL preco
INTEGER cadeira
PRINT *, 'Digite o numero da cadeira escolhida'
READ *, cadeira
SELECT CASE (cadeira)
CASE(50)
PRINT *,'preco=R$50.00'
CASE(100:140,200:240)
PRINT *,'preco=R$25.00'
CASE (300:340)
PRINT *,'preco=R$20.00'
CASE (400:440)
PRINT *,'preco=R$15.00'
CASE DEFAULT
PRINT *,'Número da cadeira inválida!!'
END SELECT
END
```

Este mesmo programa escrito utilizando o comando IF ficaria como mostrado no EXEMPLO8-2.f.

EXEMPLO8-2.f

```
 PROGRAM PRECCOINGRESSO
!Num concerto, os preços dependendo das cadeiras são!
!cadeiras                  preços!
!50                         50.00!
!100 - 140 e 200 - 240     25.00!
!300 - 340                  20.00!
!400 - 440                  15.00!
!este programa fornece o preço dos ingressos!
 IMPLICIT NONE
 INTEGER cadeira
 PRINT *, 'Digite o numero da cadeira escolhida'
 READ *, cadeira
 IF (cadeira .EQ. 50) THEN
 PRINT *,'preco=R$50.00'
 ELSE IF(((100 .LE. cadeira) .AND. (cadeira .LE. 140)) .OR.
&((200 .LE. cadeira) .AND. (cadeira .LE. 240))) THEN
 PRINT *,'preco=R$25.00'
 ELSE IF ((300.LE. cadeira) .AND. (cadeira .LE. 340)) THEN
 PRINT *,'preco=R$20.00'
 ELSE IF ((400 .LE. cadeira) .AND. (cadeira .LE. 440)) THEN
 PRINT *,'preco=R$15.00'
 ELSE
 PRINT *,'Número da cadeira inválida!!'
 END IF
 END
```

LAÇOS (EXECUÇÃO REPETIDA)

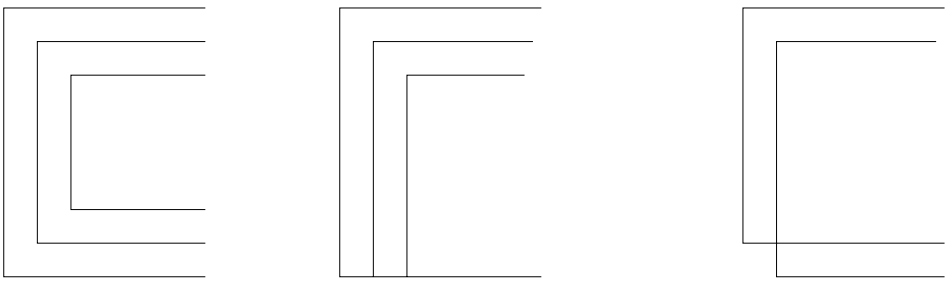
COMANDO *DO*

A sintaxe deste comando é:

DO n i=m_1, m_2, m_3
m_1 é o valor inicial de i
m_2 é o valor final de i
m_3 é o passo pelo qual a variável i será indexada (default m_3=1)
END DO

Este comando permite fazer loops variando de m_1 até m_2 com passo de m_3 n é o número da linha que delimitará o DO. No FORTRAN 77 também é possível encerrar com END DO.Também é possível fazer DO dentro de um outro DO:

PERMITIDAS **NAO PERMITIDA**



Para sair do laço de DO antes deste terminar,pode-se adicionar um GO TO dentro de um IF. Os comandos IF (de transferência), GO TO, STOP e RETURN só poderão ser utilizadas dentro de um *loop*.

CONTINUE

Este comando é utilizada para encerrar um laço de DO antes de um END DO. A sintaxe utilizada é:

```
      DO 11 i=1,10
      Comandos do bloco
 11 CONTINUE
```

ou entao, sem o CONTINUE:

```
   DO i=1,10
   Comandos do bloco
   END DO
```

EXEMPLO9.f

```
   PROGRAM CONTAGEM
      !faz contagem regressiva a partir de um numero fornecido!
      IMPLICIT NONE
      INTEGER interv, tempo, inicio
      PARAMETER (interv=-1)
      PRINT *, 'Entre com um numero inteiro'
      READ *, inicio
      PRINT *, 'Começando a contagem regressiva'
      DO 10 tempo=inicio, 1, interv
      PRINT *, tempo
 10 CONTINUE
      PRINT *, 'BUMMMM!!'
      STOP
      END
```

LAÇOS DENOMINADOS

A situação em que temos um DO dentro de outro DO é chamado de laços denominados da seguinte forma:

```
DO i=1,10 !(externo)!
    DO j=1,5 !(interno)!
       PRINT *, i,j
    END DO !(interno)!
END DO !(externo)!
```

EXEMPLO10.f

```
 PROGRAM LOOP
   !este programa permite observar quando está sendo!
   !executado o primeiro e o segundo DO!
   IMPLICIT NONE
   INTEGER M, N
   PARAMETER (M=3, N=2)
   INTEGER I, J
   PRINT *,' I J '
   DO 10 I=1,M
      PRINT *, 'primeiro DO ', I
      DO 20 J=1,N
      PRINT *, 'segundo DO ',I,J
20 CONTINUE
10 CONTINUE
   STOP
   END
```

DO WHILE

Este comando é utilizado para executar um bloco repetidas vezes enquanto a condição for verdadeira. A partir do momento em que a condição passa a ser falsa, este sai do loop continuando a execução do restante do programa.

```
DO WHILE (condição)
Comandos do bloco
END DO
```

EXEMPLO11.f

```
PROGRAM RADIAÇÃO
  !este programa calcula o nível de radiação e o grau de segurança!
  IMPLICIT NONE
  REAL RADSEG, FATSEG, NIVRAD, RADMIN
  PARAMETER (RADSEG=0.466, FATSEG=10.0)
  INTEGER DIA
  DIA=0
  PRINT *,'Entre com o nível de radiação do dia'
  READ *, NIVRAD
  PRINT *, 'n.dias radiação condição'
  RADMIN=RADSEG/FATSEG
  DO WHILE (NIVRAD .GT. RADMIN)
     IF (NIVRAD .GT. RADSEG) THEN
       PRINT *,DIA,NIVRAD,' INSEGURO'
     ELSE
       PRINT *, DIA, NIVRAD, ' SEGURO'
     END IF
  DIA=DIA+3
  NIVRAD=NIVRAD/2.0
  END DO
  END
```

O FORTRAN 90 possui os comandos de loop EXIT (para sair de um loop) e CYCLE (para repetir um loop). Permite também que os parâmetros e as variáveis do controle do DO loop sejam omitidas formando um loop infinito. A saída do loop ocorre quando o valor zero é digitado. Os valores negativos são ignorados.

*C*YCLE

Produz um desvio na execução de um *DO* a sintaxe deste comando pode ser vista no EXEMPLO12.f

EXIT

Provoca uma interrupção na execução do comando DO cuja sintaxe também é observada no EXEMPLO12.f

EXEMPLO12.f

```
 PROGRAM QUANTIDADE
IMPLICIT NONE
INTEGER produto, item
produto=1
DO
   PRINT *,'Digite a quantidade de produtos desejados'
   READ *, item
   IF (item==0) EXIT
   IF (item < 0) CYCLE
   produto = produto * item
   PRINT *, 'Total de itens pedidos é igual a', produtos
END DO
END
```

COMANDOS DE ENTRADA E SAÍDA

OUTPUT FORMATADO

Até agora, utilizamos os comandos PRINT $*$ para mostrar os resultados de saída. O $*$, neste caso, indica que o tipo de dado de cada lista do output determina a forma em que o valor seja mostrado na tela.
No comando PRINT formatado, substitue-se o $*$ da seguinte forma:
Para o valor de N=5347 temos,

```
      PRINT 15,'o valor de N é',N
15 FORMAT (1X, A, I5)
```

e na tela aparecerá:

O valor de N é 5347

O comando FORMAT possui 3 descritores de edição que ficam entre parênteses.

- 1X - controla a linha de espaço entre esta linha de output e a outra linha seguinte. 1X está especificando o espaço de linha simples.
- A - indica o número de caracteres impressos, neste caso, é exatamente o tamanho do output do item lista.
- I5 - diz que o valor de N é uma variável inteira, tem 5 colunas e que o ultimo digito deve aparecer a direita. No caso de N ter menos que 5 dígitos, aparecerá os espaços em brancos a esquerda do número.

No caso de ter mais de uma variável, por exemplo, as variáveis nome, tamanho, depende; a formatação pode ser feita da seguinte maneira:

```
    PRINT 35, nome, tamanho, depende
35 FORMAT (1X, A10, 3X, F7.2, 2X, I2)
```

- A10, F7.2 e I2 - descrevem a forma que as variáveis serão impressas.
- 3X e 2X - especificam os espaços em brancos a serem deixados entre cada uma das variáveis.

A tabela a seguir mostra o que significam cada um dos descritores do comando FORMAT.

DESCRITOR	*EFEITO*
Aw	Serão impressos exatamente w caracteres. Os caracteres excedentes serão cortados a direita.
A	O n^o de caracteres impressos é do tamanho do output do item lista.
Fw.d	imprime-se um número real, justificado a direita em w colunas. A parte fracionária terá 'd' decimais.
Iw	É impresso uma variável inteira, justificada a direita em w colunas.
nx	Haverá n espaços brancos.

Os caracteres da linha de controle, 1X, podem ser representados por ' '. Assim a linha do 35 FORMAT anterior pode ser escrita como:

35 FORMAT (' ', A10, 3X, F7.2, 2X, I2)

Uma falha na especificação faz com que o Fortran utilize o primeiro caracter do output como linha de controle e isto, pode acarretar resultados indesejáveis. Há outros descritores para controle de linha.

DESCRITOR	*EFEITO*
' 'ou 1X	Imprime a linha especificada na linha seguinte (espaço simples).
'0'	Pula 1 linha entre as linhas (espaço duplo).
'1'	Imprime esta linha no topo da página seguinte.
'+'	Imprime esta linha acima da linha anterior.

- Caso o seu terminal não processe estes descritores, estes devem ser omitidos.
- O Fortran permite que as especificações de formato sejam feitos diretamente logo após o comando PRINT.

EXEMPLO: PRINT '(1X, A, I2)', 'O valor da contagem é', cont

na tela aparecerá:

Valor da contagem é 37

INPUT FORMATADO

É semelhante ao output. Neste caso, ao invés do comando PRINT utiliza-se a formatação após o comando READ.

EXEMPLO: READ 62, mes, dia, ano
62 FORMAT (I2, 1X, I2, 1X, I2)

OUTROS DESCRITORES

- *T* - é análogo a colocar um tab na posição.

EXEMPLO: PRINT 45, nome, tamanho, dep
45 FORMAT(1X,A10,T15,F6.2,T25,I5)

- O T25 provoca um avanço para a posição 25 da linha do output.

- / - é utilizado para formatar mais de uma linha do output.

EXEMPLO: PRINT 18, x, y
18 FORMAT (1X, F8.2, 5(/), 1X, F8.2)

Quando o / aparece de forma consecutiva, isto significa que, vai aparecer linhas brancas na página. No exemplo acima, o 5(/) faz aparecer 4 linhas brancas entre o x e o y

- *E* - outra forma de ler ou mostrar os números reais. O número a ser lido, neste caso, deve estar na notação exponencial utilizando a letra E na forma *Ew.d*. A especificação do tamanho *w* deve incluir a letra *E*, o sinal, e o valor do expoente inteiro. Portanto, o *w*, deve ser normalmente maior que *d*+5

EXEMPLO: o número -6.245E-6 deve ser representado por E9.3

REPETIÇÃO DE DESCRITORES

- Para repetir o mesmo descritor pode se colocar o número de vezes que será repetida na frente do descritor.

EXEMPLO: READ 15, X, Y, Z
15 FORMAT (F3.2, F3.2, F3.2)
ou
15 FORMAT (3F3.2)

EXEMPLO13.f

```
 PROGRAM RADIACAOFORM
!Este programa calcula o grau de contaminação de uma certa!
!area e o tempo necessario para que esta fique com um grau!
   !de segurança!
   REAL RADSEG, FATSEG
   PARAMETER (RADSEG=0.466, FATSEG=10.0)
   INTEGER DIA
   REAL NIVRAD, RADMIN
   DIA=0
   PRINT *,'entre com o nivel de radiação do dia'
   READ *, NIVRAD
   PRINT 15, 'n. dias', 'radiacao', 'condicao'
15 FORMAT (1X,A7,2X,A11,2X,A8)
   RADMIN=RADSEG/FATSEG
   DO WHILE (NIVRAD .GT. RADMIN)
     IF (NIVRAD .GT. RADSEG) THEN
      PRINT 25, DIA, NIVRAD, 'INSEGURO'
    25 FORMAT (1X, I7, 2X, F11.7, 2X, A8)
     ELSE
         PRINT 35, DIA, NIVRAD, 'SEGURO'
     35 FORMAT (1X, I7, 2X, F11.7, 2X, A8)
     END IF
   DIA = DIA + 3
   NIVRAD = NIVRAD/2.0
   END DO
   END
```

COMANDOS

OPEN

- Antes de ler e escrever um arquivo, o sistema computacional necessita de algumas informações básicas sobre o arquivo, como por exemplo, o nome do arquivo, se é um arquivo já existente ou um novo a ser criado pela execução do programa.

Sintaxe : OPEN (UNIT= número da unidade, FILE= nome do arquivo, STATUS= tipo de arquivo)
EXEMPLO: OPEN (UNIT=1, FILE= 'dados', STATUS = 'OLD')
OPEN (UNIT=2, FILE='saida', STATUS='NEW')

- O número da unidade deve ser um número inteiro não associado a um outro arquivo.

- O tipo de arquivo deve ser 'NEW' para um arquivo de saída ou 'OLD' para um arquivo existente.

READ

Para ler as informações de um arquivo, deve-se usar a forma modificada do comando READ da seguinte maneira:

*Sintaxe:*READ(número da unidade, format) input
Exemplos: READ (3,25)PNOME
READ(4,*) x,y

Os dados são colocados em cada variável especificado da lista de input. Os dados são lidos da próxima linha do arquivo associado com o número de unidade indicada. Esta associação é feita pelo comando OPEN que deve ser executada antes do comando READ. O format, indica onde deve ser direcionado a operação do READ. O '*' indica que o formato é livre enquanto que o formatado é dado por um número inteiro.

WRITE

É um comando semelhante ao PRINT.

*Sintaxe:*WRITE (número da unidade, formato) lista de output
Exemplos: WRITE (3,*) 'A hora é', HORA
WRITE (4,15) X,Y,'X é maior que Y'

O valor de cada variável ou constante da lista de output é escrita num arquivo associado ao número da unidade indicada. O formato indica se (*) o output é listado diretamete, ou (número inteiro) se é formatado.

WRITE e PRINT

As diferenças entre os dois comandos são de que:

- PRINT - este comando expõe o output na tela do computador e utiliza a vírgula depois do comando PRINT.

- WRITE - não utiliza a vírgula depois da palavra de comando WRITE, pode expor o output na tela do computador ou armazená-lo em um arquivo e o número da unidade especificada após o WRITE determina onde o output será mandado.

END FILE

Este comando é colocado no final do programa para marcar o fim do arquivo de output. Como este especifica um fim no arquivo de output gravado, nenhum dado será acrescentado depois deste.

*Sintaxe:*END FILE(UNIT=número da unidade)
Exemplo: END FILE (UNIT=2)

CLOSE

Este comando fecha ou desconecta os arquivos com os números das unidades especificados. Todo arquivo aberto deve ser fechado antes de parar com a execução do programa.
Sintaxe: CLOSE(UNIT=número da unidade)
Exemplo: CLOSE(UNIT=1)

O número 6 quando indicado como número de unidade está associando com o monitor enquanto que o número 5 com o teclado. Assim,

- WRITE(6,*)'Horas trabalhadas' ou WRITE (*,*) indica que a mensagem escrita a seguir será mostrado na tela (UNIT=6 ou *).
- READ(5,*) HORA ou READ(*,*) lê o próximo dado digitado (UNIT=5 ou *) como sendo a variável HORA.
- *OBS.* Em alguns sistemas o número 6 significa impressora.

FUNÇÕES DEFINIDAS PELO PROGRAMADOR

DEFINIÇÃO DE FUNÇÃO

Sintaxe: tipo_da_função FUNCTION nome_da_ função (lista de argumentos)

interface da função
declaração dos argumentos
declarações locais
corpo da função

- A primeira linha, especifica o nome da função e o tipo de resultado que retornará. Este resultado pode ser do tipo INTEGER, REAL, LOGICAL, CHARACTER*n). Quando a função não tiver nenhum argumento, um parênteses vazio, '( )', deve aparecer no início.
- A linha seguinte, da interface é uma série de comentários que devem ser colocados com informações úteis para os usuários e outros programadores que queiram utilizar a função. Geralmente, o comentário inicial descreve o que faz a função.

Exemplo: !Precondition: R é definido!

Esta linha de comentário descreve a condição que deve ser verdadeira antes da função ser chamada, esta condição é conhecida como sendo *'precondition'*. Logo após, na linha seguinte, aparece o *'postcondition'*, que descreve a condição que deve ser verdadeira depois que a execução da função for completada.

- A linha do corpo da função descreve a manipulação de dados da função cujo resultado é armazenado no nome da função que deve retornar como um valor da função.

EXEMPLO15.f

```
   tipo      nome (argumentos)!
    REAL FUNCTION AREAC(R)
   ! Esta é a interface da função!
   ! Calcula a area de um circulo com raio R.!
! Precondition: R é definido!
   ! Postcondition: O resultado desta funcao é a area do circulo!


   ! Declaracao de argumentos!
    REAL R
   !
   ! Declaracao local!


    REAL PI
    PARAMETER (PI=3.14159)
   !
   ! Define o resultado da funcao!
    AREAC = PI * R ** 2
   ! Sair da funcao!
    RETURN
    END
```

EXEMPLO15b.f

```
REAL FUNCTION CIRCC(R)
! Calcula a circunferencia de um circulo com raio R!
! Precondition: R eh definido!
! Postcondition: O resultado da funcao eh a circunferencia do circulo!

!Declaracao dos argumentos!
REAL R

!Declaracao Local!
REAL PI
PARAMETER (PI=3.14159)

!Definicao do resultado da funcao!
CIRCC=2.0 * PI * R

!Sair da funcao!
RETURN
END
```

Uma função é um subprograma e como tal não pode ser executada sozinha. Deve portanto, ser chamada para ser executada por um programa principal ou um outro subprograma. Como mostra o EXEMPLO16.f que chama as duas funções do EXEMPLO15.f e EXEMPLO15b.f.

EXEMPLO16.f

```
! Este programa principal chama e executa o subprograma function!
PROGRAM CIRCULO
! Declaracoes das variaveis!
REAL RAIO, AREA, CIRCUM
CHARACTER *12 UNIDADES
REAL AREAC, CIRCC
! Leitura dos valores de RAIO e UNIDADE!
PRINT *, 'Digite com as unidades'
READ *, UNIDADES
PRINT *, 'Entre com o raio (em ',UNIDADE,')'
READ *, RAIO
! Calcula a area!
AREA=AREAC(RAIO)
! Calcula a circunferencia!
CIRCUM=CIRCC(RAIO)
! Imprime os valoresde AREA e CIRCUM!
PRINT *, ' A area é', AREA,UNIDADES,'ao quadrado'
PRINT *, ' A circunferencia mede', CIRCUM, ' ', UNIDADES
STOP
END
```

FUNÇÕES LINHAS DE COMANDO

Algumas definições de funções podem ser escritas em uma linha de comando e possuem a seguinte forma:
Sintaxe: nome_ da_ função (argumentos) = expressão
Exemplo: sind(ang) = sin (3.14159 x ang/180.0)
Quando a função nome_ da_ função é chamada, cada valor do argumento é substituído pelo seu correspondente argumento da expressão e a expressão é executada. As funções linha de comando, devem ser escritas no corpo da rotina que executará a função.

EXEMPLO17.f

```
 PROGRAM NEWTON
!Este programa calcula a raiz da equação utilizando o método de Newton.!
!Declarações das variáveis!
REAL EPSLON
INTEGER MAXGES
PARAMETER (EPSLON=0.00001, MAXGES=100)
REAL XLAST, XNEXT
INTEGER NUMGES
!Funções!
REAL F, FPRIME, X
F(X) = 5 * X * X * X - 2 * X * X + 3
FPRIME(X) = 15 * X * X - 4 * X
!Ler o valor inicial de XNEXT!
PRINT *, ' Entre com o valor inicial para a raiz!'
READ *, XNEXT
! Calcula os valores provaveis!
DO 20 NUMGES = 1, MAXGES
XLAST=XNEXT
XNEXT=XLAST - F(XLAST)/FPRIME(XLAST)
    IF (ABS(XNEXT - XLAST) .LT. EPSLON) THEN
    PRINT 5, 'A raiz aproximada é ', XNEXT
    PRINT 5, 'O valor da função é ', F(XNEXT)
5   FORMAT (1X, A, E17.11)
    PRINT 15, NUMGES, 'outras possibilidades'
15 FORMAT (1X, I3, A)
    !Para sair do loop!
          GOTO 99
        END IF
20 CONTINUE    !Saida!
    PRINT 25, 'Não foram encontradas as raízes antes ',MAXGES
25 FORMAT (1X, A, I3, A)
!Continuação depois do loop!
99 CONTINUE
    STOP
```

END

Comando GOTO

Sintaxe: GOTO número
Exemplo: GOTO 99

Neste comando, o controle do programa é transferida imediatamente para a linha especificada pelo número. Este número deve aparecer entre as colunas 1 e 5 como nos outros comandos do Fortran.

DEFININDO UMA SUBROTINA

Uma *subrotina* é um programa separado cuja execução pode retornar qualquer número de resultados ao programa que o utiliza.
Sintaxe: SUBROUTINE nome_da_subrotina(lista de argumentos)
Seção de interface
Seção de declaração local
Corpo da subrotina
RETURN
END

- O comando SUBROUTINE especifica o nome da subrotina.
- A lista de argumentos pode ser a lista de identificadores, argumentos do input do programa chamado, ou ainda servir para retornar resultados do programa chamado.
- A seção de interface contém as descrições e as declarações de todos os argumentos.
- O corpo da subrotina descreve a manipulação de dados que são executados na subrotina.
- O comando RETURN transfere o controle ao comando de chamada.
- O comando END coloca fim na definição da subrotina.

EXEMPLO 18.f

```
    !nome da subrotina!
SUBROUTINE SUM(A,B,C)
!interface da subrotina!
!Esta subrotina calcula a soma de 3 numeros inteiros!
!Precondition:Há 2 variáveis pré-definidas!
!Postcondition: Há um resultado final!
!Declaração local!
INTEGER A, B, C
!Corpo da subrotina!
C = A + B
RETURN
END
```

CHAMANDO UMA SUBROUTINA

Para chamar uma subrotina é se necessário utilizar o comando CALL

*Sintaxe:*CALL nome_da_subrotina (lista de argumentos atuais)
*EXEMPLO:*CALL ORDER (A,B)

REGRAS PARA USAR A SUBROTINA

- O comando CALL deve ser utilizada para chamar cada subrotina.
- Deve existir o mesmo número de argumentos atuais e argumentos definidos na subrotina.
- Cada um dos argumentos atuais devem ser dos mesmos tipos e na mesma ordem dos argumentos correspondentes.

EXEMPLO 19.f

```
!Este programa utiliza o subprograma do ex18.f e faz a soma!
      PROGRAM PRINCIPAL
      INTEGER X, Y, Z
      X = 5
      Y = 3
      PRINT *, ' X Y Z '
      CALL SUM (X, Y, Z)
      PRINT 5, X, Y, Z
5     FORMAT (1X, I4, I4, I4)
      CALL SUM (Y, X, Z)
      PRINT 5, X, Y, Z
      CALL SUM (Z, Y, X)
      PRINT 5, X, Y, Z
      CALL SUM (Z, Z, X)
      PRINT 5, X, Y, Z
      CALL SUM (Y, Y, Y)
      PRINT 5, X, Y, Z
      STOP
      END
```

SUBPROGRAMAS INTERNOS

Os subprogramas que são compilados separadamente do programa principal são considerados subprogramas externos. Em Fortran90, pode-se declarar um ou mais subprogramas internos antes do comando END do programa principal. O primeiro subprograma interno deve ser precedido do comando CONTAINS. Neste caso, como o subprograma é compilado como parte do programa principal, pode-se acessar qualquer variável ou parâmetro do programa principal sem declará-lo novamente no subprograma.

EXEMPLO 20.f

```
   PROGRAM SUBINT
!Este programa coloca dois itens na ordem e o mostra na tela!
!Declaracao!
REAL x, y
!Entrada de dados!
PRINT *, 'Entre com dois numeros:'
READ *, x, y
!Ordena os dados!
CALL ORDEM
!Mostra os resultados!
PRINT *, 'O menor valor é' , x
PRINT *, 'O maior valor é' , y

STOP

CONTAINS

SUBROUTINE ORDEM
!Ordena os valores em x e y!

!Declarações locais!
REAL temp

!Troca os valores se necessario!
IF ( x > y) THEN
temp = x
x = y
y = temp
END IF

RETURN
END SUBROUTINE ORDEM
```

ARRAYS(Conjuntos)

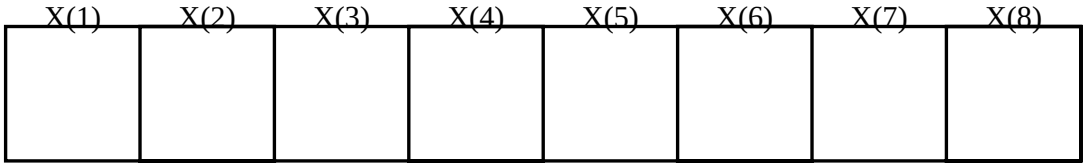
- Um array, ou conjunto, é a estrutura de dados utilizados para estocar todos os itens de dados do mesmo tipo.
- Cada item deste conjunto é armazenado separado de cada item na memória principal. Devido ao armazenamento separado de cada item podemos processar um item mais de uma vez na ordem desejada.

Declaração e referência

Um array é um conjunto de 2 ou mais células de memórias adjacentes chamadas de elementos de arrays que são associados a um nome simbólico. Para declarar um array, deve-se declarar o nome do array e o número de células associadas.

EXEMPLO: REAL X(8)

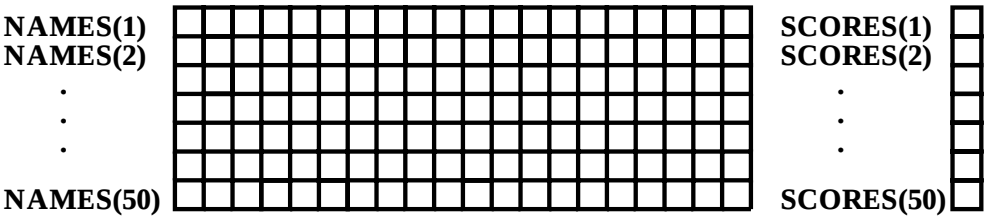
O número 8 entre parênteses é o subscrito do conjunto e subscreve uma variável X com 8 elementos. A variável subscrita X(1) pode ser usada como referência ao primeiro elemento do conjunto X, X(2) o segundo elemento, X(3) o terceiro, e assim por diante.



EXEMPLO: CHARACTER *20 NAMES(50)
INTEGER SCORES (50)

Neste exemplo, as variáveis NAMES e SCORES possuem 50 elementos cada. Sendo que, cada elemento do conjunto NAMES pode ter no máximo 20 caracteres.

Mais de um array deve ser declarado em um tipo simples de declaração.

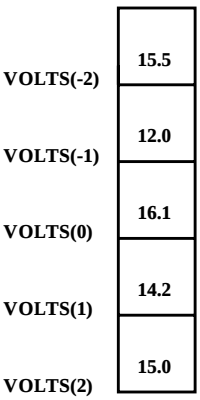


EXEMPLO: REAL CACTUS(5), NO, PINO(6)
INTEGER FATOR(12), N, INDICE

Neste caso, as células de memória individuais serão utilizadas para guardar as variáveis simples NO, N e INDICE. Frequentemente, torna-se necessário especificar o número de elementos do conjunto ficando mais fácil a mudança do tamanho do conjunto.

EXEMPLO: INTEGER BAIXA, ALTA
PARAMETER (BAIXA=-2, ALTA=2)
REAL VOLTS (BAIXA:ALTA)

Neste exemplo, o array possue os elementos: VOLTS(-2), VOLTS(-1), VOLTS(0), VOLTS(1), VOLTS(2)



DECLARAÇÃO DE ARRAY

Sintaxe: tipo_dos_elementos nome (tamanho)
tipo_dos_elememento nome (valor_mínimo:valor_máximo)

EXEMPLO: REAL A(5), B(-2 : 2)

- No primeiro exemplo de declaração de conjunto, o conjunto A aloca tamanho 5 de celas de memória. Cada cela pode armazenar um item do dado do tipo REAL eespecificado no início da declaração. Os elementos individuais do conjunto são referenciados como variáveis A(1), A(2), A(3), A(4) e A(5).

- No segundo array (B(-2 : 2)), o espaço de armazenamento alocado para o conjunto passa a ser -2, -1, 0, 1, 2. Esta forma deve ser utilizada quando o menor valor do subscrito for diferente de 1. Os valores de mínimo e máximo devem ser inteiros constantes ou parâmetros, assim como, o valor mínimo deve ser menor ou igual ao valor máximo.

SUBSCRITO DO CONJUNTO

O subscrito do conjunto serve para diferenciar os elementos individualmente dentro de um array. Isto nos permite manipular apenas os elementos desejados.

EXEMPLO: X(8), significa que tenho um conjunto chamado X com tamanho 8, ou seja:

X(1)	X(2)	X(3)	X(4)	X(5)	X(6)	X(7)	X(8)
16.0	12.0	6.0	8.0	2.5	12.0	14.0	-54.5

Cada qual com o seu valor.

Utilizando LOOP com comando DO para processar um ARRAY

Frequentemente, desejamos utilizar os elementos de um conjunto para fazer cálculos ou um processamento em seqüência. Em Fortran, pode-se fazer isso utilizando o comando DO com uma variável controladora de LOOP. O aumento nesta variável faz com que o próximo elemento do conjunto seja utilizado.

EXEMPLO21.f

```
  PROGRAM CONJLOOP
    INTEGER MAXITM
    PARAMETER (MAXITM = 8)
    INTEGER I
    REAL X(MAXITM), AVERAG, STDEV, SUM, SUMSQR


    PRINT *, 'Entre', MAXITM, 'números, um por linha:'
    DO 10 I=1, MAXITM
    READ *, X(I)
10 CONTINUE


    SUM = 0.0
    SUMSQR = 0.0
    DO 20 I=1,MAXITM
    SUM = SUM + X(I)
    SUMSQR = SUMSQR + X(I) ** 2
20 CONTINUE
    AVERAG=SUM/REAL(MAXITM)
    STDEV=SQRT(SUMSQRT/REAL(MAXITM) - AVERAG ** 2)
    PRINT *
    PRINT 15, 'o valor médio é', AVERAG
    PRINT 15, 'o desvio padrao é', STDEV
15 FORMAT (1X, A, F8.1)


    PRINT *
```

```
      PRINT *, 'tabela de diferença entre X(I) e a media'
      PRINT 25, 'I', 'X(I)', 'diferença'
 25 FORMAT (1X, A4, 3X, A8, 3X, A14)
      DO 30 I=1, MAXITM
            PRINT 35, I, X(I), X(I)-AVERAG
 35         FORMAT (1X, I4, 3X, F8.1, 3X, F14.1)
 30 CONTINUE


      STOP
      END
```

INPUT E OUTPUT DE CONJUNTOS

INPUT DE ARRAY

No *EXEMPLO 21*, vimos que para ler o conjunto X fez-se:

```
      PRINT *, 'Entre com', MAXITM, 'números, um por linha:'
      DO 10 I=1, MAXITM
          READ *, X(I)
 10 CONTINUE
```

Uma outra forma de ler e guardar os elementos do conjunto X consecutivamente seria escreve-lo da seguinte maneira:

```
      PRINT *, 'Entre com', MAXITM,'números:'
      READ *, X
```

Quando não colocamos os subscritos no nome do conjunto, este refere-se ao conjunto todo e não apenas a um dos elementos. Há duas vantagens de se usar este formato.

- A primeira vantagem é a simplicidade e a segunda vantagem é que desta forma, ele permite que você digite vários elementos m uma única linha.

- A desvantagem é que o número de itens dos dados é determinado pelo tamanho declarado do array, então, deve-se trabalhar sempre com aquele número de dados.

OUTPUT DE ARRAY

A mesma técnica do INPUT pode ser utilizada para o OUTPUT. Assim, no *EXEMPLO 21* a linha de comando:

```
      DO 30 I=1, MAXITM
            PRINT 35, I, X(I), X(I)-AVERAG
   35     FORMAT (1X, I4, 3X, F8.1, 3X, F14.1)
   30 CONTINUE
```

Pode ser escrita como:

```
            PRINT 35, (I, X(I), X(I)-AVERAG, I=1,MAXITM)
   35     FORMAT (1X, I4, 3X, F8.1, 3X, F14.1)
```

COMANDO *DATA*

O comando *DATA* atribui valores iniciais a variáveis em um programa. Esta declaração deverá preceder todas as outras declarações executáveis.

Sintaxe: DATA lista1/a1,a2,k3*a3,a4/, lista2/b1,b2,...
em que,
lista: é o nome do conjunto de variáveis que recebe os valores iniciais.
a: são as constantes reais, inteiras ou caracteres.
*k:*é o número de vezes que a constante seguinte após o asterístico será repetida.

*EXEMPLO:*DATA A/3*1/ que representa DATA A/1,1,1/
DATA A,B,C /2.5,6.5,9.0/

que armazenará cada um dos valores iniciais em cada variável.

A	B	C
2.5	6.5	9.0

que pode ser escrita desta outra forma:

DATA A/2.5/, B/6.5/, C/9.0/

Quando temos um conjunto com mais de um elemento, o comando DATA é utilizado como:

DATA letras/'A', 'B', 'C', 'D', 'E'/

Neste caso, como *letras* é um conjunto, letras(1) vai corresponder ao caracter A, letras(2) ao B, e assim por diante.

Isto pode ser também representado por:

DATA(letras(i),i=1,5)/'A','B','C','D','E'/

OPERAÇÕES COM ARRAYS (CONJUNTOS)

Em Fortran77, cada operação deve representar um único valor. Enquanto que, no Fortran90, um operador possui um array e seu respectivo operador. Isto permite que todo array sofra o efeito da operação.

EXEMPLO 22a.f

```
      PROGRAM FORTRAN77
    IMPLICIT NONE
    INTEGER i
    INTEGER A(3), B(3), C(3)
    DATA A/1,2,3/, B/4,5,6/
    DO 10 i=1,3
    C(i)= A(i) + B(i)
    PRINT *, C(i)
10  CONTINUE
    STOP
    END
```

EXEMPLO 22b.f

```
      PROGRAM FORTRAN90
    INTEGER i
    INTEGER A(3), B(3), C(3)
    DATA A/1,2,3/, B/4,5,6/
    C = A + B
    PRINT *, (C(i), i=1,3)
    STOP
    END
```

Assim, as operações de arrays em Fortran90, podem ser:
EXEMPLO: C = A + B, C = A * B, C = SQRT(B)

CONSTRUTORAS DE ARRAY

Pode-se declarar um subarray de um array. Este subarray A é chamado de seção de array e utiliza a seguinte notação:

A(3:6)que representam A(3), A(4), A(5), A(6) de um array A.

Pode-se especificar um array constante utilizando as *construtoras* que são as listas de expressões de constantes que estão entre as barras (/ /)

EXEMPLO: A(3:6) = (/-1, 0, -1, 0)

que representa A(3)=-1, A(4)=0, A(5)=-1, A(6)=0

Caso apareça 3 números entre parênteses para denotar uma seção de array, o 3^o parâmetro é análogo ao loop do DO.
EXEMPLO: A(1:10:2) = (/5*1/)

Neste exemplo o A(1)=1, A(3)=1, A(5)=1, A(7)=1 e A(9)=1

COMANDO *WHERE*

Este comando, permite limitar o array para um determinado elemento deste.

Sintaxe: WHERE (expressão lógica com conjuntos)
bloco de comandos 1
ELSEWHERE
bloco de comandos 2
END WHERE

ou ainda:

WHERE (expressão com conjuntos) comandos

EXEMPLO 23.f

```
    PROGRAM WHERE
INTEGER A(10), B(10), C(10)
DATA A/1,2,3,4,5,6,7,8,9,1/
DATA B/4,2,5,2,3,1,10,1,8,0/
WHERE (A > B)
C = A - B
ELSEWHERE
C = B - A
END WHERE
PRINT *, C
END
```

ALGUMAS FUNÇÕES INTRÍNSECAS PARA ARRAYS (CONJUNTOS)

Em Fortran90 existem várias funções intrínsecas que operam sobre todos os elementos do array fornecendo os resultados.

FUNÇÕES INTRÍNSECAS	*OPERAÇÕES*
MAXVAL(A)	fornece o maior valor do conjunto A
MINVAL(A)	fornece o menor valor do conjunto A
DOT_PRODUCT(A,B)	fornece o produto dos conjuntos A e B
SUM(A)	fornece a soma dos elementos do conjunto A
PRODUCT(A)	fornece o produto dos elementos do conjunto A
ALL(T)	retorna .TRUE. se todo T(i) for .TRUE.
ANY(T)	retorna .TRUE. se qualquer T(i) for .TRUE.
COUNT(T)	fornece a quantidade de elementos de T que forem .TRUE.

CONJUNTOS (ARRAYS) MULTIDIMENSIONAIS

Declaração de conjuntos multidimensionais

Sintaxe: tipo_elemento nome_array (tamanho1, tamanho2, ...)
tipo_elemento nome_array (minval1:maxval1, minval2:maxval2, ...)

EXEMPLO: REAL TABELA (10, 0:6)
REAL TABELA2 (7,5,6)

Um array ou conjunto multidimensional pode ter no máximo 7 dimensões. No primeiro exemplo, temos um conjunto bidimensional em que, o primeiro subscrito vai de 1 até 10 e o segundo de 0 até 6. No segundo exemplo, temos um conjunto tridimensional, em que, o primeiro subscrito vai de 1 até 7, o segundo de 1 até 5 e o terceiro de 1 até 6.

Manipulação de Conjuntos Bidimensionais

Os subscritos linha e coluna devem ser especificados para referenciar um elemento de um conjunto de duas dimensões. Caso a variável I seja inteira como no exemplo a seguir:

EXEMPLO: PRINT *, (TICTAC(1,I), I=1,3)

Esta linha de comando diz para mostrar todos os elementos da primeira linha do conjunto TICTAC. Portanto, para trabalhar com todos os elementos do conjunto TICTAC tem-se que utilizar os loops em uma ordem pré-determinada.

Inserir os valores no conjunto

Para inserir os dados em um conjunto bidimensional, pode-se utilizar o loop através do comando DO como mostra o fragmento a seguir:

```
      INTEGER TABELA(5,3), ROW, COLUMN
      DO 10 COLUMN = 1, 3
          PRINT *, 'Entre com os dados da coluna', COLUMN
          READ *, (TABELA (ROW, COLUMN), ROW = 1,5)
 10   CONTINUE
```

ou então:

```
   PRINT *, 'Entre com 15 valores para o conjunto começando pela 1ª coluna'
   READ *, TABELA
```

Comando DATA

O comando *DATA* pode também ser utilizado para inicializar um conjunto multidimensional.

EXEMPLO:
```
DATA((TABELA(I,J), J=1,3),I=1,5)
  + /1,2,3, 4,5,6, 7,8,9, 10,11,12, 13,14,15/
```

Isto quer dizer que 1,2,3 será colocado na primeira linha do conjunto TABELA; 4,5,6 na segunda linha e assim por diante.

ARGUMENTOS PARA CONJUNTOS MULTIDIMENSIONAIS

Os conjuntos multidimensionais podem ser utilizados como argumentos de subrotinas. Quando utilizamos uma subrotina para definir os elementos de um conjunto, as variáveis locais correm o risco de serem perdidas. Para evitar que isso ocorra, ao retornar ao programa principal, utiliza-se o comando SAVE.

Comando *SAVE*

Sintaxe:
```
SAVE
SAVE lista
```

A *lista* contem todos os nomes das variáveis locais ou conjuntos, cujos valores devem ser salvos na execução do comando RETURN do subprograma. Caso o comando SAVE apareça sem a lista das variáveis ou conjuntos, todos os valores das variáveis locais do subprograma serão salvos. Como o comando SAVE não é um comando executável, deve necessariamente, aparecer na seção de declaração do subprograma antes de qualquer comando executável.

EXEMPLO 24.f

```
    REAL FUNCTION RANDOM
!Este programa gera numeros randomicos utilizando SEED como gerador de
números randomicos!
!Precondition: SEED é predefinido e OLDSED contem os valores anteriores
para o seed.!
!Postconditions: Retorna um valor entre 0.0 e 1.0 e OLDSED terá os valores
do próximo seed.!

!Declaração de argumentos!
INTEGER SEED

!Declaração local!
INTEGER C1, C2
PARAMETER (C1 = 19423, C2 = 811)
INTEGER OLDSED
SAVE OLDSED
DATA OLDSED /0/

IF (OLDSED .EQ. 0) THEN
OLDSED = SEED
END IF

!Gerando novos valores de OLDSED!
OLDSED = MOD(C1 * OLDSED, C2)

!Definindo os resultados!
```

```
RANDOM = REAL(OLDSED) / REAL(C2)

!sair da função!
RETURN
END
```

Comando *COMMON*

Todos os recursos de comunicação entre o programa principal e os subprogramas era feito através das listas de argumentos. O Fortran possui a capacidade de comunicar dados, colocando-os em uma área comum chamada *bloco COMMON*.

```
EXEMPLO: INTEGER MAXSIZ
         PARAMETER (MAXSIZ = 100)
         COMMON SCORES (MAXSIZ)
         INTEGER SCORES
```

No fragmento acima, o bloco COMMON guardará o conjunto SCORES de MAXSIZ variáveis do tipo INTEGER. O tipo de cada variável no bloco COMMON deve ser declarado tanto antes ou depois do comando COMMON. Caso a variável seja um conjunto, seu tamanho deve ser declarado.

RESTRIÇÕES DO COMANDO COMMON

- Um parâmetro não pode ser estocado.
- Uma variável listada no comando COMMON não pode ser um argumento deste subprograma.
- As variáveis do tipo CARACTER não pode ser misturada com outros tipos de variáveis.
- Itens do COMMON não podem aparecer nos comandos SAVE e DATA.

DECLARAÇÕES DO COMANDO COMMON

Sintaxe: COMMON lista
COMMON /nome/lista

EXEMPLO: COMMON A, B(10)
COMMON /HAT/ C, D

No primeiro caso, os conjuntos A e B são alocados no bloco COMMON sem nome. No segundo caso, o bloco COMMON chamado HAT, recebe os conjuntos C e D. Assim, os conjuntos C e D compartilham da mesma área de memória. Este segundo caso é recomendado pois além de melhor organizar os arquivos pode ser chamado pelo programa principal.

OPERAÇÕES COM MATRIZES

DECLARAÇÃO DE CONJUNTO

O Fortran90 permite uma alternativa para declarar as matrizes da seguinte forma:

```
REAL, DIMENSION (5,10) :: M
REAL, DIMENSION (5,6 : 15) :: N
```

Nestes casos, os conjuntos M e N possuem 5 linhas e 10 colunas. O conjunto N possue subscritos linhas 1 até 5 e subscritos colunas de 6 até 15. A notação l:u significa o menor subscrito (l) e o maior subscrito (u). Caso não tenhamos *l*, o menor subscrito do conjunto é assumido e quando não temos o *u* o maior subscrito é assumido. Assim, para o nosso conjunto M e N temos:

- M(: , :5) significa que tenho um conjunto de 2 dimensões com subscrito linha de 1 a 5 e coluna de 1 a 5.
- M(2, :) significa que tenho conjunto de 1 dimensão com todos os elementos da linha 2.
- M(: ,5) significa que tenho um conjunto de 1 dimensão com todos os elementos da coluna 5.

FUNÇÕES INTRÌNSECAS PARA MATRIZ

Com exceção do *DOT_PRODUCT* todas as demais funções podem ser utilizadas para matrizes. E ainda, há algumas outras.

MATMUL(M, P) multiplica matriz M pela matriz P
TRANSPOSE(N) matriz transposta de N

EXEMPLO 25.f

```
 PROGRAM MATRIZ

INTEGER I,J
INTEGER, DIMENSION (3,3) :: M
INTEGER, DIMENSION (3,3) :: N
INTEGER, DIMENSION (3,3) :: P

DATA ((M(I,J),J=1,3),I=1,3)/1,2,3, 4,5,6, 7,8,9/
DATA ((N(I,J),J=1,3),I=1,3)/9,8,7, 6,5,4, 3,2,1/

PRINT *, 'M =',M
PRINT *, 'N =',N

P = MATMUL(M,N)
PRINT *, 'P =',P
END
```

Módulos

O Fortran90 permite que o programador agrupe ou encapsule um conjunto de declarações para os dados de um objeto (variáveis ou arrays) e subprogramas que faça, alguma operação utilizando estes objetos. Por exemplo:

```
MODULE blank_common
    INTEGER max_size
    PARAMETER (max_size = 100
    INTEGER scores (max_size)
END MODULE
```

Neste exemplo, o MODULE blank_common contem as declarações do array *scores* e um parâmetro (max_size). No outro exemplo a seguir, o MODULE stu_data possui as declarações dos dois arrays.

```
MODULE stu_data
    USE blank_common
    CHARACTER *20 names(max_size)
    CHARACTER *1 grades(max_size)
END MODULE stu_data
```

Cada módulo deve ser salvo no próprio arquivo fonte e compilado separadamente. A linha de comando

```
    USE blank_common
```

Indica que o *stu_data* tem acesso a todas as declarações no módulo *blank_common* isto nos permite declarar o parâmetro *max_size* apenas uma vez. O módulo *stu_data* é considerado *cliente* de zemphblank_common. Portanto, o módulo *blank_common* deve ser compilado antes do modulo *stu_data*.

ORGANIZAÇÃO DAS VARIÁVEIS DE FORTRAN NA MEMÓRIA DO COMPUTADOR

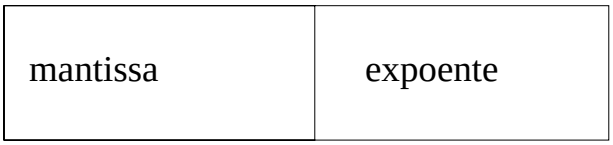
- Na memória do computador *todos* os dados são armazenados em binários de 0 e 1. Portanto, o armazenamento de variáveis dos tipos INTEGER e REAL são diferentes.
- As variáveis INTEGER podem guardar somente valores inteiros como por exemplo, os números 100, 0, -999, ocupando menor espaço de armazenamento quando comparado com as variáveis REAL e sendo portanto, de acesso mais rápidas.

formato INTEGER

- As variáveis do tipo REAL possuem o ponto decimal e a parte fracional ocupando 2 espaços de armazenamento na memória. Uma para os números e outra para os expoentes. Todos os números reais são transformadas da seguinte forma:

EXEMPLO: $3.14159 \rightarrow 0.314159.10^{1}$
$-15.0 \rightarrow -0.15.10^{2}$
$0.000123 \rightarrow 0.123.10^{-3}$

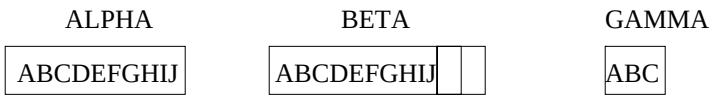
formato REAL



- As variáveis do tipo CHARACTER vão reservar o número de espaços de armzenamento na memória indicado na declaração.

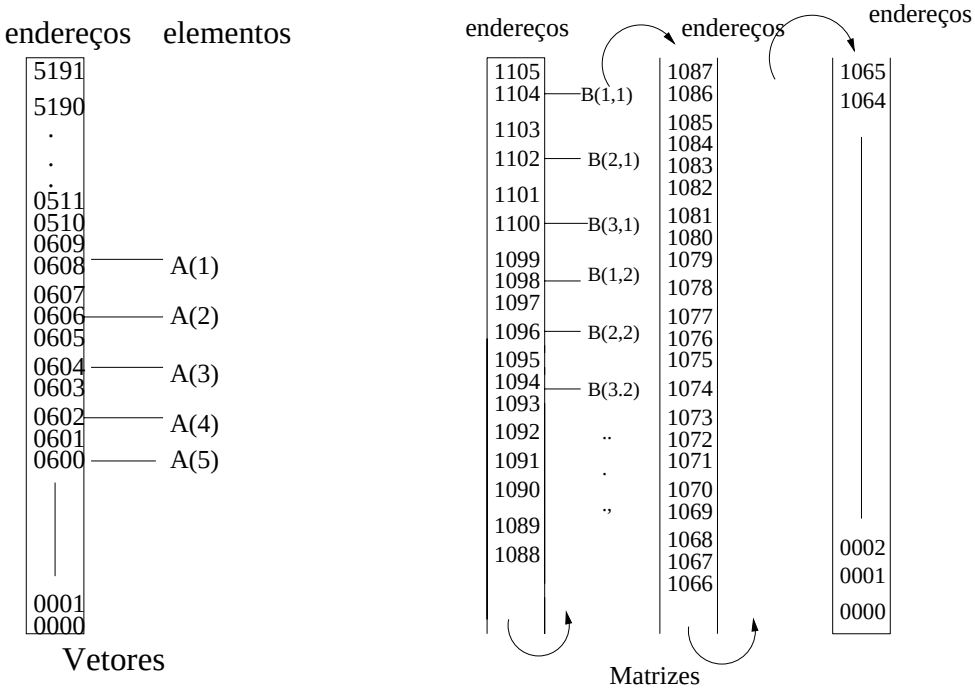
Por exemplo: CHARACTER ALPHA *10, BETA *12, GAMMA *3

ALPHA = 'ABCDEFGHIJ'
BETA = ALPHA
GAMMA = ALPHA



Vetores e Matrizes

Os arrays como os vetores e matrizes são armazenados da seguinte maneira:



COMANDO *EQUIVALENCE*

O comando EQUIVALENCE permite a mais de um nome de variáveis apontar para o mesmo endereço de memória porém, todas as variáveis que compartilharem o mesmo endereço de memória terão o mesmo valor. Desta maneira, economiza-se espaço em memória. O uso indevido deste comando pode provocar erros de lógica no programa por isso, deve tomar cuidado ao utilizar este comando.

EXEMPLO: EQUIVALENCE (a, b, c)

Neste exemplo, através do comando EQUIVALENCE, as variáveis a, b e c serão equivalentes compartilhando o mesmo espaço na memória. Portanto, os valores de a, b e c serão exatamente os mesmos, podendo se dizer que b e c tornaram-se apenas *apelidos* para a variável a.

COMANDO *EXTERNAL*

O comando EXTERNAL, avisa ao compilador que os endereços passados para funções não são de variáveis, mas de funções. Estas funções são definidas no programa que chama a função utilizando-as como argumentos.

EXEMPLO: EXTERNAL sin, cos,xexp

Neste exemplo, o xexp será tratada como sendo uma função intrínsica da mesma maneira que o cos e o sin sendo que, o xexp é uma linha de comando como:

xexp(x) = x * exp(x)

DERIVADOS E PONTEIROS

TIPOS DERIVADOS

Pode-se criar tipos de dados adicionais, conhecidos como tipos derivados a partir e dados do tipo intrínsecos e de outros derivados.

Sintaxe: TYPE nome
declarações
END TYPE nome

em que o *nome* é o nome do tipo derivado que pode ser uma função intrínseca. Entre o nome e o comando TYPE, pode-se especificar o tipo de acesso deste derivado quando se trabalha com módulo. Este acesso pode ser do tipo PRIVATE ou PUBLIC.

EXEMPLO 26

```
    ...
     TYPE COORD_3D
        REAL :: x,y,z
     END TYPE COORD_3D
     TYPE ESFERA
        TYPE (COORD_3D) :: centro
        REAL :: raio
     END TYPE ESFERA
...

...
     TYPE PEOPLE
        INTEGER AGE
        CHARACTER*20 NAME
     END TYPE PEOPLE
     TYPE PEOPLE
     TYPE (PEOPLE) :: SMITH = PEOPLE(25, 'John Smith')
     END
...
```

```
MODULE ABC
    TYPE, PRIVATE :: SYSTEM     !neste caso, somente o tipo de derivado SYSTEM pode ser acessado!
        SEQUENCE    !faz sequencia com o módulo ABC!
        REAL :: PRIMARY
        REAL :: SECONDARY
        CHARACTER(20), DIMENSION(5) :: STAFF
    END TYPE
END MODULE
```

Comando *PRIVATE*

O comando PRIVATE especifica que um módulo não é acessível quando se está fora deste módulo.

Sintaxe: PRIVATE :: lista

- Refere-se a lista, todas as especificações genéricas ou o nome das variáveis, procedures, tipos derivados, constante ou um grupo de lista.
- O comando PRIVATE só pode aparecer no módulo. Se o módulo tiver um argumento ou um resultado de função que possui acesso privado e não deve possuir um identificador genérico como no PUBLIC.
- Caso o comando PRIVATE for especificado sem a definição do tipo derivado, todos os componentes do tipo derivado se tornam privado.

EXEMPLO 26.f

```
MODULE MC
      PUBLIC
      INTERFACE GEN
          MODULE PROCEDURE SUB1, SUB2
      END INTERFACE
      PRIVATE SUB1
      CONTAINS
          SUBROUTINE SUB1(I)
            INTEGER I
  I = I + 1
          END SUBROUTINE SUB1
          SUBROUTINE SUB2(I,J)
              I = I + J
          END SUBROUTINE
  END MODULE MC


  PROGRAM ABC
      USE MC
      K = 5
      CALL GEN(K)


      CALL SUB2(K,4)
      PRINT *, K
  END PROGRAM
```

Comando *INTERFACE*

É o primeiro comando do bloco de interface, o que pode especificar uma interface explícita para um procedure externo ou não.

EXEMPLO:

```
INTERFACE
   FUNCTION VOL(RDS, HGT)
      REAL VOL, RDS, HGT
   END FUNCTION VOL
   FUNCTION AREA(RDS)
      REAL AREA, RDS
   END FUNCTION AREA
END INTERFACE
```

ATRIBUIÇÕES EM TIPOS DERIVADOS

Há duas maneiras de se atribuir vlores aos tipos derivados: por componente (utilizando o símbolo %) ou como objeto.
Por exemplo:

```
   TYPE (esfera) :: e1, e2, e3
   e1 % raio = 1.0
   e1 % centro % x = 0.0
   e1 % centro % z = -2.0
...
...
   e2 % raio = 2.0
   e2 % centro = coord_3D(5.0, -1.5, -0.5)
...
   e3 = esfera (coord_3D(4.0, 1.0, 3.5), 3.0
...
...
   e1 = e2
...
```

PONTEIROS

- Os ponteiros são variáveis especiais que faem referência aos espaços de memória. Em Fortran90, os ponteiros só podem apontar para objetos que tenham os atributos TARGET, POINTER, ALOCATABLE ou INTENT.

- Os ponteiros precisam ter o mesmo tipo, parâmetro e dimensões de seus alvos. E o alvo não deve ser uma seção de array com um vetor subscrito e nem todo um array completo.

- Os objetos do ponteiro não podem aparecer nos comandos DATA, EQUIVALENCE, NAMELIST, integer POINTER (POINTER (P,I)).

EXEMPLO 27.f

```
TYPE T
   INTEGER, POINTER :: COMP_PTR
ENDTYPE T
TYPE(T) T_VAR
INTEGER, POINTER :: P,Q,R
INTEGER, POINTER :: ARR(:)
BYTE, POINTER :: BYTE_PTR
LOGICAL(1), POINTER :: LOG_PTR
INTEGER, TARGET :: MYVAR
INTEGER, TARGET :: DARG(1:5)
P => MYVAR
Q => P
NULLIFY (R)
Q => R
T_VAR = T(P)
ARR => DARG(1:3)
BYTE_PTR => LOG_PTR
END
```

ESTADO DE PONTEIROS

Um alvo associado a um ponteiro pode ser referenciado por uma *associação de ponteiros* que possuem 3 status.

- *associado:* o ponteiro possui sempre um alvo.
- *não associado:* o ponteiro é anulado por um comando NULLIFY mas continua existindo, ou seja, sem um alvo.
- *indefinido:* é o estado inicial de um ponteiro.

Comando *NULLIFY*

Este comando torna os ponteiros em ponteiros não associados.

```
TYPE T
    INTEGER CELL
    TYPE(T), POINTER :: NEXT
ENDTYPE T
TYPE(T) HEAD, TAIL
TARGET :: TAIL
HEAD % NEXT => TAIL
NULLIFY (TAIL%NEXT)
END
```

Comando *BYTE*

O comando BYTE especifica os atributos do objeto e funções do tipo byte. Cada objeto possui tamanho 1. O ponteiro do tipo BYTE não pode ser associado com alvo do tipo CHARACTER, e nem o ponteiro tipo CHARACTER pode ser associado ao alvo do tipo BYTE.

REFERÊNCIAS BIBLIOGRÁFICAS

- XL Fortran for AIX
  Language Reference
  Version 3 Release 2
  Third Edition, North York, Ontario, Canadá, 1994

- Koffman, E.B. e Friedman, F.L. *in* Fortran
  Fifth edition Update, USA, 1997