

Programação em Linguagem FORTRAN

Helder Pereira Cristo

helderpc@deq.ufmg.br

Belo Horizonte - Setembro/1999

ÍNDICE

Primeira Parte: Conceitos da Linguagem 5

1. Introdução 5
2. Formatação 5
3. Conceitos Básicos 6
4. Declaração de Variáveis 7
 - Tipos de Variáveis: 7*
 - Inteiras (INTEGER): 7*
 - Reais (REAL): 7*
 - Complexas (COMPLEX): 8*
 - Alfanuméricas (CHARACTER): 8*
 - Lógicas (LOGICAL): 8*
5. Operadores 9
 - Atribuição 9*
 - Operadores Literais 9*
 - Operadores Aritméticos 9*
 - Operadores Relacionais 10*
 - Operadores Lógicos 10*
 - Prioridade 11*
6. Funções Intrínsecas 11
 - Funções Trigonométricas 11*
 - Outras Funções 12*
7. Funções e Subrotinas 13
 - Funções 13*
 - Subrotinas 14*
8. Leitura e Impressão 15
9. Formatos 16
 - Outros Recursos Para Formatos 18*
10. Arquivos 19
 - Outros Recursos 21*

Segunda Parte: Estruturas de Programação 22

1. Estrutura Seqüencial 22
2. Comando 'GO TO' ou 'GOTO' 23
3. Estrutura Condicional 24

Estrutura Condicional Simples 24
Estrutura Condicional Composta 24
Estrutura Condicional Composta Simplificada 25

4. Estruturas de Repetição 28
Estruturas de Repetição Simples 25
DO WHILE (F90) 25
DO Implícito (WIN) 30

Terceira Parte: Recursos de Programação 32

1. Deslocamento 32
GOTO Implícito 32
IF Com Deslocamento 32
2. Declarações e Atribuições Avançadas 33
DIMENSION 33
PARAMETER 34
TYPE (F90) 35
DATA 36
3. Designação de Memória 38
COMMON 38
BLOCK DATA 39
4. Modularização 39
INCLUDE 39

Apêndice A: Funções Intrínsecas 41

1. Funções Trigonométricas 41
2. Funções Genéricas 42
3. Exponenciais 43
4. Logaritmos 43
5. Máximos 43
6. Mínimos 43
7. Restos 43
8. Raiz Quadrada de X 44
9. Truncamento de X 44
10. Arredondamento de X 44
11. Diferença Positiva Entre X e Y 44
12. Tipo de Dado 44
13. Transformação do Tipo de X 43
14. Complexos 46

15. Caracteres **46**

16. Valores Absolutos de X **46**

Apêndice B: Opções de Arquivos, Leitura e Escrita 48

1. Abertura de Arquivos (OPEN) **48**

2. Fechamento de Arquivos (CLOSE) **49**

3. Escrita (WRITE) **49**

4. Leitura (READ) **50**

5. Recuo Total (REWIND) **50**

6. Recuo de um Campo (BACKSPACE) **51**

Apêndice C: Tabela de Valores ASCII 52

Primeira Parte: Conceitos da Linguagem

1. Introdução

Em FORTRAN existem basicamente duas formas de se escrever um programa: com formulário fixo ("*fixed form*") ou com formulário livre ("*free form*"). Sendo este segundo disponível apenas para compiladores baseados em WINDOWS. Outro ponto importante são os comandos válidos somente nos novos compiladores (por exemplo Microsoft Developer Studio) que aceitam programas em FORTRAN 90. Por isso será adotado o seguinte critério para as explicações :

- tópicos precedidos do identificação WIN só são válidos para WINDOWS;
- tópicos precedidos de F90 são válidos apenas para os compiladores que aceitam comandos FORTRAN 90.

Deve ficar claro que compiladores para FORTRAN 90 aceitam também os outros dois tipos, e os baseados em WINDOWS aceitam todos os comandos dos compiladores mais antigos (FORTRAN 77), e que a recíproca não é verdadeira.

Os programas podem ser escritos em qualquer editor de texto, desde que sejam salvos com as extensões .for ou .f90. Esta segunda forma somente para FORTRAN 90. Os compiladores em WINDOWS possuem um editor próprio, que deve ser usado pois possui muitos recursos adicionais, como por exemplo o destaque das palavras chaves.

2. Formatação

Os seguintes critérios devem ser seguidos para se escrever um programa em FORTRAN no modo de formulário fixo:

- colunas 1 a 5: são usadas escrever os rótulos "*label*" ou números de comando. Estes números devem ser inteiros e estar totalmente contido nestas colunas. Não podem se repetir e não precisam estar em ordem crescente. Serão usados para que outros comandos possam identificar aquela linha;
- coluna 6: qualquer caractere diferente de 0 "zero" nesta coluna indica que o que vem a seguir é continuação da linha anterior ou da última linha que não seja um comentário (próximo item), podem existir até 19 linhas de continuação;
- colunas 7 a 72: comandos ou comentários;
- colunas 73 a 80: campos de identificação, são usados pelo computador, portanto não se deve escrever nestas colunas.

F90 – O programa pode ser escrito em qualquer posição, desde que o modo "*free form*" esteja ativado. Alguns pontos devem ser observados para este formato:

- as linhas de continuação são indicadas pelo símbolo '&' no fim da sentença, e o que estiver na linha abaixo será tomado como continuação. É permitido a inserção de comentários após o '&';
- os rótulos devem ser os primeiros caracteres da linha, e podem estar em qualquer coluna.

3. Conceitos Básicos

1. Comentários: não são interpretados pelo computador, um bom programa deve conter muitos para que fique o mais claro possível. Em FORTRAN a letra 'c' ou o caractere '*' na primeira coluna indicam que toda a linha é um comentário. Na linha de comentário é permitido o uso de qualquer caracter, especial ou não.

WIN – o ponto de exclamação '!' indica que o que vem após ele é comentário, ele pode vir em qualquer posição, inclusive após comandos.

2. Variáveis e Nomes de Blocos: devem ter no máximo seis letras, não é permitido o uso de caracteres espaciais e não podem começar com um número.

WIN – podem ter 31 caracteres, inclusive o caracter *uppercase* '_'

3. Constantes:

- o Numéricas: podem conter qualquer valor real, inteiro ou complexo. A parte decimal é separado da inteira por um ponto '.'. Os zeros antes e depois do ponto decimal podem ser omitidos, se não forem significantes. O expoente decimal é indicado pela letra 'e' ou 'E', deve vir entre o número seu expoente sem o uso de espaços entre eles. Números negativos assim como a parte exponencial quando for negativa deve vir precedida do sinal menos '-'. O sinal '+' é opcional em ambas as partes. Os números imaginários devem vir entre parênteses e a parte real deve ser separada por uma vírgula da parte imaginária.
- o Alfanuméricas: (são as '*strings*', seqüências de letras e/ou números) podem conter qualquer seqüência de caracteres. Deve vir entre aspas " " ou apóstrofos ' '. As aspas tem preferencia sobre os apóstrofos, portanto um valor literal pode conter apóstrofos, desde que seu valor venha entre aspas. Não é permitido o uso de caracteres especiais e letras acentuadas. Uma outra forma de se declarar valores alfanuméricos é o uso de **wH**valorliteral, onde w é o número de caracteres do valor alfanumérico.

4. Maiúsculas e Minúsculas: o FORTRAN não faz qualquer distinção entre letras maiúsculas e minúsculas. É permitido inclusive o uso do nome da variável escrita de formas diferentes no mesmo programa. EX.: VAR = var = Var.
5. Os programas podem conter no início o seu nome (program nome_do_programa), e devem terminar com a palavra '*end*'.
6. Outra forma de parar o programa, e esta pode ser usada em qualquer parte dele, é usando a palavra '*stop*'. O programa terminará independentemente de haver mais comandos na seqüência.

4. Declaração de Variáveis

As variáveis podem ser inteiras, reais ou literais. A declaração de uma variável deve vir antes que ela seja usada, se isto não ocorrer o compilador assumirá as variáveis que começam com as letras I até N como inteiras (*INTEGER*4*) e todas as outras como reais (*REAL*4*).

Esta forma de declaração implícita pode ser modificada usando o comando '*implicit tipo* (a1-a2,b1-b2,...)' antes do tipo de variável. Sendo a1, a2, b1, b2 quaisquer letras da alfabeto. Quando não se deseja que nenhuma variável seja declarada implicitamente usa-se o comando '*implicit none*'.

Para se declarar variáveis que sejam matrizes e vetores deve-se indicar suas dimensões logo após o nome da variável; entre parênteses, e separadas umas das outras por vírgula. Ex.: a(4,3) indica uma matriz de 4 linhas por 3 colunas. As variáveis podem receber valores iniciais usando '/valor/', logo após sua declaração.

Tipos de Variáveis:

Inteiras (INTEGER):

Podem assumir os seguintes valores:

INTEGER*1 -128 a 127

INTEGER*2 -32,768 a 32,767

INTEGER*4 -2,147,483,648 a 2,147,483,647

INTEGER*4 pode ser representado somente por: INTEGER

Reais (REAL):

Precisão simples até 6 casas decimais:

REAL*4 ou REAL ` 3.402823E+38

Incremento mínimo de ` 1.175494E-38

Precisão dupla até 15 casas decimais:

REAL*8 ou DOBLE PRECISION ` 1.797693134862316D+308

Incremento mínimo de ` 2.225073858507201D-308

A parte exponencial deve ser separada por um 'd' ou 'D' no lugar do 'e' ou 'E'.

WIN – pode ser usado o 'e' ou 'E' para separar a parte exponencial. Não só isso mas também todas as outras funções (item 6 primeira parte) podem ser iguais a de um real comum para compiladores baseados em WINDOWS.

Complexas (COMPLEX):

Precisão simples até 6 casas decimais:

COMPLEX*8 ou COMPLEX

Precisão dupla até 15 casas decimais:

COMPLEX*16

Os valores que um complexo pode assumir são os mesmos que os reais.

Alfanuméricas (CHARACTER):

CHARACTER NOME*w

Onde w representa o número máximo de caracteres que a variável pode conter dentro do programa.

Ou CHARACTER *wvar1,var2

(var1 e var2 possuem o mesmo tamanho w)

Ou CHARACTER (LEN = w) var1, (LEN=w2) var2

(var1 tem tamanho w e var2 tamanho w2)

Lógicas (LOGICAL):

LOGICAL NOME

Podem assumir os valores .TRUE. (VERDADEIRO) ou .FALSE. (FALSO)

Ou somente T e F

```
implicit integer (a-t,x,y,z)
integer num_dias/3/
real temp(3)
character data1*8,*8 data2,data3
data1 = "12\10\98"
data2 = "20\11\98"
data3 = "15\3\99"
```

```
temp (1) = 25.6  
temp (2) = 22.4  
temp (3) = 22.8
```

O sinal '=' atribui a variável o valor que vem á sua direita (item 5).

F90 – estas declarações poderiam estar na mesma linha desde que fossem separadas por ponto e vírgula ';':

```
data3 = "15\3\99";temp (1) = 25.6;num_dias= 3
```

5. Operadores

Atribuição

A variável ou identificador que estiver a esquerda do sinal de atribuição '=' recebem o valor da expressão, constante ou variável que estiver à direita.

Identificador = Expressão

```
nome = 'Engenharia Quimica'
```

```
ano = 1999
```

Operadores Literais

Uma função útil para variáveis literais é a concatenação, ou a junção de duas ou mais palavras. Em FORTRAN a concatenação é feita pelo operador '//'.
Exemplo: a = 'tele', b = 'visao', c = a//b => c = 'televisao'

```
a = 'tele'  
b = 'visao'  
c = a//b => c = 'televisao'
```

Operadores Aritméticos

Executam operações aritméticas comuns.

FORTRAN	Matemática Tradicional	Significado
+	+	soma
-	-	Subtração

*	\square_{+}	Multiplicação
/	\square_{+}	Divisão
**	a^p	Potenciação

Quando uma variável inteira recebe o resultado de uma divisão com resto, este resto é desprezado ou seja o valor é truncado.

$$C = A^{**}2 + B^{**}2$$

$$D = E^{**}(1/2)$$

Operadores Relacionais

Comparam variáveis, constantes ou expressões e retornam '.TRUE.' ou '1' se a comparação for verdadeira, '.FALSE.' ou '0' se a comparação for falsa.

FORTRAN	Matemática Tradicional	Significado
.LT.	<	MENOR QUE
.LE.	\square_{+}	MENOR OU IGUAL QUE
.EQ.	=	IGUAL A
.NE.	\square_{+}	DIFERENTE DE
.GT.	>	MAIOR QUE
.GE.	\square_{+}	MAIOR OU IGUAL QUE

20 .ne. 30	=> verdadeiro
1000 .lt. 500	=> falso

Operadores Lógicos

São usados quando é necessário mais de uma condição relacional ou quando é preciso inverter seu resultado.

FORTRAN	Significado
.AND.	Junção
.OR.	Disjunção
.NOT.	Negação
10.GT.5 .AND. 20.GT.25	=> .FALSE.
10.GT.5 .OR. 20.GT.25	=> .TRUE.

```
.NOT. 20.GT.25
```

```
=> .TRUE.
```

Prioridade

Operador	Prioridade
**	1 ^a
*	2 ^a
/	2 ^a
+	3 ^a
-	3 ^a
.EQ.	4 ^a
.NE.	4 ^a
.GT.	4 ^a
.GE.	4 ^a
.LT.	4 ^a
.LE.	4 ^a
.NOT.	5 ^a
.AND.	6 ^a
.OR.	7 ^a

O uso de parênteses pode ser feito para trocar a ordem de prioridade.

```
(20.GT.10 .AND. 20.GT.25) .OR. (10.LT.20 .AND. 10.LT.(3*10))  
=> .TRUE.
```

6. Funções Intrínsecas

Existem várias funções predefinidas em FORTRAN, que podem ser usadas em qualquer parte do programa.

Funções Trigonômicas

Nome	Definição	Tipo de Argumento	Tipo da Função
SIN (x)	seno (radianos). Se x for complexo, a parte real é assumida como valor em radianos.	Real ou complexo.	REAL*4
ASIN (x)	Arcoseno (radianos).	Real, x .le. 1	REAL*4

	Retorna valores na faixa [$-\pi/2$, $\pi/2$]		
COS (x)	Coseno (radianos) Se x for complexo, a parte real é assumida como valor em radianos.	Real ou complexo	REAL*4
ACOS (x)	Arcocoseno (radianos)) Retorna valores na faixa [0, π]	Real, x .le. 1	REAL*4
TAN (x)	Tangente (radianos)	Real	REAL*4
ATAN (x)	Arcotangente (radianos). Retorna valores na faixa [$-\pi/2$, $\pi/2$]	Real	REAL*4
SINH (x)	Seno Hiperbólico (radianos)	Real	REAL*4
COSH (x)	Coseno Hiperbólico (radianos)	Real	REAL*4
TANH (x)	Tangente Hiperbólica (radianos)	Real	REAL*4

Outras funções trigonométricas podem ser encontradas no Apêndice A.

Outras Funções

Nome	Definição	Tipo de Argumento	Tipo da Função
ALOG10 (x)	logaritmo de x na base 10	real	real
ALOG (x)	logaritmo neperiano de x (x > 0)	real	real
EXP (x)	o número e (base dos logaritmos neperianos) elevado a x	real	real
ABS (x)	valor absoluto de x	real	real
IABS (x)	valor absoluto de x	inteiro	inteiro
IFIX (x)	conversão de real para inteiro, truncando	real	inteiro
FLOAT (x)	conversão de inteiro para real	inteiro	real
DBLE (x)	converte para dupla precisão	real	real (dupla precisão)
CMPLX (x)	converte para o tipo complexo	real	complexo
SIGN (x,y)	fornece valor positivo de x se y \geq 0 e negativo de x se y < 0	real	real
MOD (x,y)	resto da divisão de x por y	inteiro	inteiro

AMOD (x,y)	resto da divisão de x por y	real	real
SQRT (x)	raiz quadrada de x ($x \geq 0$)	real	real

Outras funções intrínsecas podem ser encontradas no Apêndice A.

7. Funções e Subrotinas

Funções e subrotinas podem ser usadas para economizar espaço e tempo de programação já que podem ser usadas várias vezes num mesmo programa. Outro ponto importante é que elas podem dar ao programa maior clareza, pois várias seqüências de cálculos ou execuções podem vir separadas do restante dos comandos.

O que dá grande versatilidade as funções e subrotinas, são os argumentos passados a elas, que quando variam produzem resultados diferentes. As funções e subrotinas podem vir em qualquer parte do programa, mas o mais comum é que apareçam no fim (após o 'end' de termino do programa), por motivo de clareza. As variáveis usadas em funções e subrotinas são locais e por isso devem ser declaradas novamente, podendo ser usado os mesmos nomes de variáveis de outras funções e subrotinas ou mesmo do programa principal. Os parâmetros necessários devem ser passados junto com a chamada da função ou subrotina, devem vir entre parênteses e separados por vírgula. Os nomes das variáveis não precisam ser os mesmos na chamada e definição de uma função ou subrotina, devendo apenas estar na mesma ordem.

Funções

Retornam sempre um valor, e a ela podem ser passados qualquer número de parâmetros. As funções funcionam de forma semelhante as funções intrínsecas, com a diferença de que agora é o programador que define o que a função de gerar como resultado. O tipo da função deve ser declarado no programa principal, como se fosse uma variável comum. Uma função pode utilizar outras funções.

Chamada:

```
nome_da_função(lista_de_parâmetros)
```

Definição:

```
function nome_da_função(lista_de_parâmetros)
definição e declaração das variáveis e constantes locais
seqüência de comandos
return
end
```

A palavra chave *'return'* é opcional. Ela indica que o comando deve retornar ao programa principal ou à função ou subrotina que a chamou.

A função retornará o valor do ultimo comando do tipo:

```
nome_da_função = expressão
```

Exemplo:

```
volume = gas_ideal(T,P,3)

function gas_ideal(temp,press,n_mols)

implicit none

real temp,press,n_mols,gas_ideal

gas_ideal= n_mols*8.314*temp/press

return

end
```

Subrotinas

Não retornam nenhum valor, e também a elas podem ser passados qualquer número de parâmetros. Nas subrotinas podem conter qualquer tipo de comandos como imprimir resultados, abrir arquivos (estes serão vistos à frente) ou executar cálculos. Como ocorre com as funções, as subrotinas podem 'chamar' outras subrotinas.

Chamada:

```
call nome_da_subrotina (lista_de_parâmetros)
```

Definição:

```
subroutine nome_da_subrotina (lista_de_parâmetros)

definição e declaração das variáveis e constantes locais

seqüência de comandos

return

end
```

A palavra chave *'return'* é opcional.

8. Leitura e Impressão

Na maior parte dos programas é preciso haver uma interação entre o programa e o usuário, essa interação pode ser feita em FORTRAN pelos comandos de leitura e escrita de dados. Esses comandos na sua forma mais simplificada possuem a seguinte estrutura:

leitura :

```
read (unidade, formato) lista_de_parâmetros
```

impressão:

```
write (unidade, formato) lista_de_parâmetros  
print formato, lista_de_parâmetros
```

Onde `lista_de_parâmetros` representa os dados que serão impressos, e devem vir separados por vírgula. Podendo conter variáveis ou expressões alfanuméricas, estas últimas devem vir entre apóstrofos `'`. Unidade representa a unidade onde os dados serão impressos ou de onde serão lidos. Formato pode conter uma lista de formatos de impressão (mostrado a seguir), um rótulo que indique um comando *'format'*, que contenha a lista de formatos ou o símbolo `*` que indica impressão ou leitura de forma livre.

As unidades `'6'` e `*` se não forem definidas dentro do programa, serão consideradas como a tela do computador (*'write'* ou *'print'*). Da mesma forma as unidades `'5'` ou `*` são definidas como o teclado (*'read'*). O comando `'print'` imprime sempre os resultados na unidade definida por `*` ou na tela caso não haja nenhuma definição para de uma unidade usando especificada pelo `*`.

Na leitura os dados devem vir separados por espaços ou vir na linha seguinte. Caso se tenha mais dados em uma linha do que os que serão lidos por um comando `'read'` eles serão desprezados, inclusive pelo próximo comando `'read'`. Na escrita os dados virão um após o outro separados por espaços (no caso de `'strings'` verão sem espaço entre elas), ou na linha seguinte quando não houver mais espaço. O próximo comando `'write'` ou `'print'` começará a escrever na linha seguinte.

F90 – para que se leia ou escreva dados na mesma linha mesmo após mudar o comando pode-se usar a opção `'advance='advance'` no comando anterior. Onde *advance* pode ser `'yes'` ou `'no'` e indica se o comando deve ou não avançar automaticamente para a próxima linha. São usados somente nos comandos `'read'` e `'write'` onde a opção formato não seja livre.

```
write (*,2,advance='no') 'Engenharia'  
  
write (*,2) ' Quimica'  
  
2 format(a)  
  
imprime: Engenharia Quimica
```

a significa que será impressa um string de qualquer tamanho (item 9).

Outras opções de impressão e leitura são encontrados no Apêndice B.

9. Formatos

Os formatos servem para que os dados sejam impressos ou lidos de uma forma específica, determinado pelo programador. Os formatos são compostos por uma seqüência de especificações que determinarão como os dados serão processados. Cada uma dessas especificações devem vir separadas por vírgulas. Pode-se ainda imprimir constantes numéricas e alfanuméricas, sendo que esta última deve vir entre apóstrofos ‘ ‘. O uso do ‘*’ no lugar do formato indica que todos os dados serão impressão de forma livre, com o número de casas especificados pelo próprio compilador. É um recurso útil para se evitar erros.

O FORTRAN não considera a primeira coluna da unidade de leitura e impressão quando a saída ou entrada é formatada, por isso deve-se incluir uma casa de impressão a mais para dados. Para formato livre não é necessário pois o FORTRAN os posiciona automaticamente.

Quando mais de um dado usar a mesma especificação, ela pode ser feita da seguinte forma: **n**especificação1, **n2**especificação2 ou **n**(especificação1 , especificação2, ...), onde **n** e **n2** representam o número de vezes que a seqüência deve se repetir.

Caso o número de especificações seja menor que o de variáveis a serem lidos ou impressos, a última especificação ou a última seqüência, caso seja usado o recurso **n**(especificação1, especificação2, ...), será repetida até que se complete o número necessário. Isto não é válido para constantes.

Quando o número de especificações é maior que os de dados a serem impressos as últimas serão desprezadas.

A forma de se declarar os formatos é a seguinte:

```
r format (especificação1, especificação2, ...)
```

Onde **r** é um número inteiro, e representa o rótulo do ‘format’. Um mesmo ‘format’ pode ser usado por vários comandos de escrita e leitura.

WIN – o formato pode vir entre apóstrofos e parênteses ‘(esp.1, esp2,..)’, dentro do próprio comando de impressão ou leitura.

```
print '(esp.)', var1
write (*, '(esp.1,esp.2)') var1,var2
read (*, '(esp.1,esp.2)') var1,var2
```

Strings devem vir entre apóstrofos duplos (“string”) nesse formato.

	Formato	Uso
	Iw[.m]	Valores Inteiros
	Fw.d	Valores Reais
	Ew.d[Ee]	Valores Reais com expoente
	Gw.d[Ee]	Mesmo que Iw[.m], Ew.d[Ee], Lw e A[w]
	Dw.d	Valores Reais de Dupla Precisão
	Lw	Valores Lógicos
	A[w]	Seqüência de Caracteres
WIN	Zw_hexedit	Valores Hexadecimais
F90	Bw[.m]	Valores Binários
F90	Ow[.m]	Valores Octadecimais
F90	ENw.d[Ee]	Valores Reais em Notação de Engenharia
F90	ESw.d[Ee]	Valores Reais em Notação Científica

'w' representa o tamanho do campo a ser impresso ou lido, 'm' representa o número de zeros que virá antes do número, 'd' representa o número de casas decimais e 'e' representa o número de casas para o expoente.

O ponto decimal, o 'e' do expoente e o sinal '-' devem ser contados ao se dar o tamanho do campo ('w'). Ao ler um dado que não possui ponto decimal e no seu formato é esperado um, o compilador lerá este dado como se lá houvesse um ponto.

12345
read (*, '(f5.3)') a => a = 12.345

Quando o ponto existe mas não está na posição especificada, a sua posição original é mantida.

12.345
read (*, '(f5.4)') a => a = 12.345

A notação com zeros 'm' só é válida quando m é maior que o número de casas que o número ocuparia no formato livre.

a = 555
print '(i5.4)', a => 0555

Nas notações com expoente (ENw.d[Ee], ESw.d[Ee], ...) não se pode omitir o 'E' ou 'e'. Sempre que houver este termo, a parte exponencial vai existir independente do seu valor.

```

a = 12.345

b = 12345.789e6

write (*,1) a,b

1 format (' a=',e10.4e3, ' e b=',e10.4e3)

=> a=.1235E+002 e b=.1235E+011

```

Caso as strings sejam maiores que o espaço reservado à elas, serão tomados apenas os w primeiros caracteres. Se forem menores, elas serão alinhadas a direita e os outros espaços deixados em branco.

Nome = 'belo horizonte'	
read (*, '(a10)') nome	=> belo horiz
write (*, '(a15)') nome	=> belo horiz

F90 – os valores dos campos podem ser variáveis desde que venham entre '<var1>'. Não é válido para 'print'.

a = 5	=> FORTR
b = 'FORTRAN'	
write (*, '(a<a>')') b	

Para valores numéricos que precisão de mais casas de impressão que as indicadas no 'format', serão impressos somente '*' naquele campo.

pi = 3.1416	
print '(1x,E7.3e2)', pi	=> *****
print '(1x,E8.3e2)', pi	=> .314E+01

Outras Recursos Para Formatos

Alguns recursos adicionais são permitidos para 'read' ou 'write'.

	Formato	Uso
	string	Transmite uma string para a saída
F90	Q	A variável recebe o número de espaços que o valor ocupa
	nH	Transmite os próximos n caracteres para a saída

	Tc	Move o ponto de leitura ou escrita para a posição c
	TLc	Move o ponto de leitura ou escrita c posições á esquerda
	TRc	Move o ponto de leitura ou escrita c posições á direita
	nX	Deixa n posições em branco
	SP	Escreve o sinal '+' onde ele é opcional
	SS	Omite o sinal '+' onde ele é opcional
	S	Retorna ou padrão 'ss'
	/	Muda de linha
WIN	\	Começa a escrever no ponto onde a ultima escrita parou
WIN	\$	Começa a escrever no ponto onde a ultima escrita parou
	:	Termina a impressão se não houver mais itens na lista
	kP	O número é multiplicado por 10 elevado à -k (se o número já possuir expoente não há efeito)
	BN	Ignora os espaços em branco
	BZ	Transforma os espaços em branco em zeros

Antes ou depois de ', ", /, \, nH, \$ ou : a vírgula é opcional.

```

real a, b

character dia*20

a = 23.99

b = 55.8e-3

dia = 'Segunda-feira'

write (*, '(5x, sp, e10.3e2, 2x, en12.3e3)') a, b

write (*, '(/2x, ''hoje e '', a8)') dia

```

produz o seguinte resultado:

```

+.240E+02 +55.800E-003

hoje e Segunda-

```

10. Arquivos

Quando se deseja trabalhar com grandes quantidades de dados, o melhor é armazená-los numa unidade de memória secundária, ou seja em arquivos. Um programa pode gerar tantos dados que todos eles não caberiam na tela de uma só vez, e ainda seriam perdidos ao finalizar o programa. Os dados salvos em arquivos podem ser usados pelo próprio

programa ou exportados para serem processados de outra forma. Arquivos de leitura economizam um tempo precioso para o usuário do programa pois ele não vai precisar de enviar dados via teclado, e com arquivos milhares de dados podem ser lidos em segundos.

O primeiro passo para se usar arquivos em FORTRAN é indicar ao programa qual é o nome do arquivo e o número de unidade referente a ele.

```
open (unidade, file='nome.ext')
```

Outras opções para abertura e fechamento de arquivos podem ser encontradas no Apêndice B.

Onde unidade deve ser um inteiro maior ou igual a zero, e é uma referencia a este arquivo. O número da unidade deve vir no comando 'read' ou 'write' indicando que os dados serão retirados ou colocados neta unidade. A disposição dos dados em arquivos são as mesmas utilizadas nas unidades de entrada e saída padrão (Item 8 Primeira Parte), com uma única diferença, as strings devem vir entre apóstrofos ' '. A abertura do arquivo pode ser feita em qualquer parte do programa (inclusive dentro de funções e subrotinas), desde que venha antes de um comando que o utilize.

Apesar de se poder usar qualquer extensão de arquivo ou até omiti-la, as extensões .dat para leitura e .out para saída são mais comumente encontradas.

Quando não for utilizado o comando 'open', o programa emitirá uma mensagem na tela pedindo o seu nome, podendo o usuário escolher um nome diferente a cada vez que o programa for executado. Todos os arquivos dever estar ou serão criados no mesmo diretório em que estiver o programa.

Um arquivo pode também ser fechado, isto fará com que o FORTRAN coloque uma marca de fim de arquivo naquele ponto, esta marca pode ser identificado por outro comando ou função (Apêndices A e B).

```
close (unidade, status='estado')  
  
ou  
  
endfile unidade
```

Onde status='estado' é opcional. Estado pode ser 'keep' que mantém o arquivo na memória (esta é a opção assumida quando status='estado' é omitida), ou 'delete' que apaga o arquivo da memória.

Arquivos fechados podem ser reabertos em qualquer parte do programa.

Outros Recursos

rewind unidade	(volta o controle ao primeiro espaço do arquivo)
backspace unidade	(volta o controle ao primeiro espaço do arquivo)

Programa	Arquivo 'arqui.dat'
<pre>character *15a,b,c open(20,file='arqui.out') open(30,file='arqui.dat') read (30,*) a write(20,*) ' este é um ' write(20,*) ' arquivo de ' write(20,*) ' saída ' read (30,*) b rewind 30 read (30,*) c write(20,*) a write(20,*) b,c end</pre>	<pre>' Química' ' Física' ' Engenharia'</pre>
	Arquivo 'arqui.out'
	<pre>este é um arquivo de saída Química Engenharia Química</pre>

Segunda Parte: Estruturas de Programação

1. Estrutura Seqüencial

Os programas em FORTRAN devem conter os comandos escritos na ordem em que serão executados, com exceção das funções, subrotinas e laços de repetição. Portanto um programa em FORTRAN deve seguir o seguinte padrão:

```
declaração 1  
declaração 2  
...  
declaração n  
comando 1  
comando 2  
...  
end
```

Onde as declarações são opcionais (item 4 primeira parte).

O comando end indica o fim do programa. Se o programador preferir pode finalizar o programa prematuramente usando o comando 'stop' ou 'call exit'.

Programa	Resultado
integer*1 a,b a=10 b=20 c=30 write (*,*) a write (*,*) b write (*,*) c end	10 20 30.000000
integer*1 a,b	10 20

<pre> a=10 b=20 c=30 write (*,*) a write (*,*) b stop write (*,*) c end </pre>	
<pre> integer*1 a,b a=10 b=20 c=30 write (*,*) a call exit write (*,*) b write (*,*) c end </pre>	<pre> 10 </pre>

2. Comando 'GO TO' ou 'GOTO'

O quando se deseja que o comando do programa avance ou recue em sua estrutura, usa-se o comando 'goto' ou 'go to'.

```
goto r
```

Onde r é um rótulo de uma linha que possui ou não um comando. Como uma linha rotulada não pode estar em branco, se não for necessária à inclusão de nenhum comando, pode-se usar a palavra chave '*continue*', que não irá interferir em nada no programa.

Programa	Resultado
<pre> goto 20 10 write (*,*) ' linha 10' 20 write (*,*) ' linha 20' 30 write (*,*) ' fim do programa' end </pre>	<pre> linha 20 fim do programa </pre>

3. Estrutura Condicional

Estrutura Condicional Simples

```

if (expressão de teste) comando

ou

if (expressão de teste) then

    seqüência de comandos

end if

```

Quando a expressão de teste for verdadeira os comandos serão executados, quando for falsa o programa segue para o próximo comando logo abaixo da estrutura condicional. A primeira opção só é valida quando for executado um único comando. Este comando pode ser de qualquer tipo, atribuição, escrita, leitura ou 'goto'.

Estrutura Condicional Composta

```

if (expressão de teste) then

    seqüência de comandos 1

else

    seqüência de comandos 2

end if

```

Quando a expressão de teste for falsa a seqüência de comandos 2 será executada. Mesmo quando só há um comando na seqüência de comandos 1, não se pode omitir a palavra chave 'then'.

É permitido o uso de estruturas condicionais umas dentro das outras.

Programa	Resultado
<pre> implicit integer (a-z) a=10 b=20 c=30 if (a.lt.b) then write (*,*) ' a<b' a=b+10 else write (*,*) ' a>b' b=a-5 end if if (c.ge.20) write(*,*)' c=',c end </pre>	<pre> a<b c= 30 </pre>

Estrutura Condicional Composta Simplificada

Uma outra forma de se usar uma estrutura condicional composta (somente WINDOWS e FORTRAN 90) é usando o comando 'case'.

WIN	F90
SELECT CASE (exp. case)	nome_case: SELECT CASE (exp. case)
CASE (lista de seleção 1)	CASE (lista de seleção 1)
comandos 1	comandos 1
CASE (lista de seleção 2)	CASE (lista de seleção 2)
comandos 2	comandos 2
...	...
CASE (lista de seleção n)	CASE (lista de seleção n)
comandos n	comandos n
CASE DEFAULT	CASE DEFAULT
comandos d	comandos d
END SELECT	END SELECT nome_case

Onde 'exp. case' é uma expressão ou constante inteira, lógica ou literal (somente um caractere 'character*1'). Caso o valor de 'exp. case' estiver na 'lista de seleção 1', os 'comandos 1' serão executados. Se o valor não estiver na 'lista de seleção 1' o computador irá avaliar a 'lista de seleção 2', se for verdadeira serão executados os 'comandos 2' e assim até terminar os comandos 'case (lista de seleção n)'. O comando 'case default' é opcional, e faz com que os 'comandos d' sejam executados caso nenhuma das outras avaliações sejam verdadeiras. 'nome_case' é opcional, deve seguir as mesmas regras usadas para dar nomes as variáveis. A sua utilidade é apenas de dar maior clareza ao programa

As listas de seleção podem ser da seguinte forma:

Estrutura	Condição para ser verdadeira
case (valor)	Exp. teste igual à valor
case (:valor)	Exp. teste menor que valor
case (valor:)	Exp. teste maior que valor
case (valor1:valor2)	Exp. teste entre valor1 e valor2
case (valor1,valor2,...,valorn)	Exp. teste igual à valor1 ou igual à valor2 ou ... valorn.

Não é permitida a superposição de valores.

Os valores literais devem vir entre apóstrofes e seus valores serão avaliados de acordo com o padrão ASCII. Uma tabela com os valores ASCII pode ser encontrada no Apêndice C.

Programa	Resultado
<pre> i=20 select case (i) case (10) write (*,*) 'a=10' case (20) write (*,*) 'a=20' case (11,19) write (*,*) 'a>11 a<19' end select end </pre>	<pre> a=20 </pre>
<pre> valor_i: character i*1 i='h' select case (i) case ('a','b','c') write (*,*) 'i=a ou b ou c' case ('d':'m') write (*,*) 'i esta entre d e m' case ('D':'M') write (*,*) 'i esta entre D e M' end select valor_i end </pre>	<pre> i esta entre d e m </pre>

4. Estruturas de Repetição

Quando o mesmo comando precisa ser executado várias vezes até que se atinja uma certa condição ou um número certo de repetições, o melhor é usar as estruturas de repetição. Estas estruturas são bem simples e podem economizar várias linhas de comando.

Estruturas de Repetição Simples

```
r1 if (condição_de_teste) goto r2  
  
    comandos  
  
    goto r1  
  
r2 continue
```

Esta é uma estrutura de repetição muito simples, mas não recomendada para programação. Estruturas mais aceitas, pois possuem equivalentes em outras linguagens, são da seguinte forma:

```
do r1 var=n1,n2,n3  
  
    seqüência de comandos  
  
r1 continue  
  
ou (WIN)  
  
do var=n1,n2,n3  
  
    seqüência de comandos  
  
end do
```

'var' é uma variável inteira que recebe inicialmente o valor 'n1', a seqüência de comandos se repete, e o valor de 'var' aumenta de 'n3' a cada vez que o comando volta para a linha do 'do'. A repetição só para quando o valor de 'var' ultrapassa 'n2'.

A palavra chave 'end do' pode ser escrita como 'enddo'. 'n1', 'n2' e 'n3' podem ser constantes ou variáveis inteiras ou reais, positivas ou negativas. Quando 'n3' for igual a um ele pode ser omitido.

DO WHILE (F90)

Uma forma de se usar uma expressão de teste no comando 'do', é o uso do 'do while'.

```

Do while(exp. teste)

    seqüência de comandos

end do

```

Enquanto 'exp. teste' for verdadeira a seqüência de comandos será executada.

Programa	Resultado
<pre> real*8 a,b a=0 b=0 1 if(a.lt.0) goto 2 a=cos(b) b= b+.125 write (*,*) a goto 1 2 continue end </pre>	<pre> 1.0000000000000000 9.921976672293290E-001 9.689124217106447E-001 9.305076219123143E-001 8.775825618903728E-001 8.109631195052179E-001 7.316888688738209E-001 6.409968581633251E-001 5.403023058681398E-001 4.311765167986662E-001 3.153223623952687E-001 1.945477079889872E-001 7.073720166770291E-002 -5.417713502693632E-002 </pre>
<pre> integer i/1/, j/1/ do while (i) j=j+1 if(j.eq.5) i=0 print *,j,i end do end </pre>	<pre> 2 1 3 1 4 1 5 0 </pre>

real*8 a(4)	+2.97646E+000
i = 1; a(1) = 2.97645984	-----
a(2) = 576.74e5	+57.67400E+006
a(3) =.45; a(4) = sin(.5)	-----
do while(i.le.4)	+449.99999E-003
print'(2x,sp,en16.5e3)', a(i)	-----
write (*,*)' -----'	+479.42554E-003
i = i + 1	-----
end do	
end	
do i=9,12	09
print '(2x,I2.2)', i	10
enddo	11
end	12

DO Implícito (WIN)

Nos comandos read e write é possível usar uma estrutura de repetição semelhante ao 'DO', mas de forma implícita e mais simplificada.

```
read(unidade, formato) (lista_de_parâmetros, var=n1, n2, n3)
```

Onde 'var=n1, n2, n3' tem o mesmo significado que na estrutura 'DO'. Pode ser usadas várias estruturas deste tipo em um mesmo read ou write desde que venham separadas por parênteses. Quando impressos desta forma, os dados ficam na mesma linha. Para leitura eles podem vir na mesma linha ou em linhas diferentes. Os parâmetros em formato serão usados até que se atinja o seu final, quando isto acontecer o comando muda de linha e os formatos são usados novamente desde o começo.

Programa	Resultado
integer a(5,3), b(3)	11 - 21 - 31
do i=1,5	41 - 51 - 101

do j=1,3	12 - 22 - 32
a(i,j)=10*i+j	42 - 52 - 108
b(j)=100+j**3	13 - 23 - 33
enddo	43 - 53 - 127
enddo	
write (*,1) ((a(i,j), i=1,5) &	
& ,b(j), j=1,3)	
1 format (2x,i4,' - ',i5,' - ',i4)	
end	

Terceira Parte: Recursos de Programação

1. Deslocamento

GOTO Implícito

Outra forma de usar o 'goto' é a seguinte:

```
goto (r1, r2, ..., rn) variável
```

Onde 'variável' é uma variável inteira que deve ter valor máximo igual ao número de rótulos que estão dentro dos parênteses. Quando esta variável tiver valor 1 o comando do programa vai para a linha rotulada com 'r1', quando tiver valor 2 vai para a linha com o rótulo r2, e assim por diante. Se o valor de 'variável' for maior que o número de rótulos o comando do programa vai para a linha imediatamente abaixo do 'goto'.

Programa	Resultado
<pre>i=2 goto (10,20,15) i 10 write (*,*) ' o valor de i=1' goto 1 20 write (*,*) ' o valor de i=2' go to 1 15 write (*,*) ' o valor de i=3' 1 continue end</pre>	<pre>o valor de i=2</pre>

IF Com Deslocamento

Uma outra maneira de se deslocar num programa, usando agora uma expressão no lugar de uma variável é a seguinte:

```
if (exp. numérica) r1,r2,r3
```

Exp. numérica não pode ser complexa. r1, r2, r3 são rótulos. Se o valor de 'exp. numérica' for menor que zero o comando vai para a linha com o rótulo r1, quando for igual a zero o comando vai para linha com o rótulo r2 e quando for maior que zero vai para r3.

Programa	Resultado
<pre> i=1 10 if(i=2) 1,2,3 1 write (*,*) ' i<2' i=2 goto 10 2 write (*,*) ' i=2' i=3 goto 10 3 write (*,*) ' i>2' end </pre>	<pre> i<2 i=2 i>2 </pre>

2. Declarações e Atribuições Avançadas

DIMENSION

Uma outra forma de se declarar vetores e matrizes, com a vantagem de se poder especificar os índices mínimos e máximos em cada direção é usando o comando *'dimension'*.

```

tipo var1,var2,...,varn

dimension var1(Li1:Ls1), var2(Li2:Ls2),..., varn(Lin:Lsn)

ou

tipo var1(Li1:Ls1), var2(Li2:Ls2), ...,varn(Lin:Lsn)

```

'tipo' pode ser qualquer tipo de variável inteira, real, lógica, complexa ou literal. Li1, Li2,..., Lin são os índices inferiores, e Ls1, Ls2,..., Ln são os índices superiores da matriz ou vetor. Os valores de Li e Ls podem ser positivos, nulos ou negativos desde que sejam inteiros. Quando o valor de Li é omitido, o valor um é usado como padrão. As variáveis podem ter duas ou mais dimensões, sendo declaradas da mesma forma, com cada dimensão separada das outras por vírgulas.

```

var(Li1:Ls1, Li2:Ls2, ...,Lin:Lsn)

```

Programa	Resultado
integer a	a(-03,+01)=-03
dimension a(-3:2,3)	a(-03,+02)=+09
do i=-3,2	a(-03,+03)=-27
do j=1,3	a(-02,+01)=-02
a(i,j)=i**j	a(-02,+02)=+04
print 1,'a(',i,',',j,')=',a(i,j)	a(-02,+03)=-08
1 format(1x,sp,a,i3.2,a,i3.2 &	a(-01,+01)=-01
& ,a,i3.2)	a(-01,+02)=+01
enddo	a(-01,+03)=-01
enddo	a(+00,+01)=+00
end	a(+00,+02)=+00
	a(+00,+03)=+00
	a(+01,+01)=+01
	a(+01,+02)=+01
	a(+01,+03)=+01
	a(+02,+01)=+02
	a(+02,+02)=+04
	a(+02,+03)=+08

PARAMETER

Uma constante pode ser representada por um símbolo no lugar do seu valor, ou seja o valor de `_` pode ser sempre que preciso referenciado como 'pi', no lugar de se escrever sempre 3.14159, bastando para isso o uso do comando '*parameter*'. A diferença entre um valor com '*parameter*' e uma variável comum é que com '*parameter*' o valor não pode ser modificado em nenhuma parte do programa ou ser lido através de um comando 'read'. O tipo do constante deve ser especificado antes ou serão usadas as atribuições implícitas (item 4 primeira parte).

```
tipo constant1, constante2,...
```

```
parameter ( constant1 = valor, constante2 = valor,...)
```

Os parênteses são opcionais. 'tipo' pode ser 'integer', real ou qualquer outro tipo de variáveis. Em 'parameter' não se pode declarar vetores e matrizes.

F90 – Uma forma simplificada de se atribuir o valor e o tipo da constante é feito da seguinte forma.

```
tipo , parameter :: constant1=valor, constante2=valor,...
```

Programa		resultado
real pi,r(3),a	real pi,r(3)	6.291800
parameter (pi=3.1459)	parameter (pi=3.1459)	25.167200
do i=1,3	do i=1,3	56.626200
r=i**2	r=i**2	
a=area(r(i),pi)	call area(r(i),pi)	
print *,a	enddo	
enddo	end	
end	subroutine area(r,pi)	
function area(r,pi)	real r,a	
real r,area	a=2*pi*r	
area=2*pi*r	print *,a	
return	return	
end	end	

TYPE (F90)

Quando várias variáveis estão relacionadas entre si, o melhor é agrupadas em grupos ('type'), de forma que possam ser acessadas pelo nome do grupo.

Declaração:

```
type nome_type
    declarações
end type nome_type
```

Nome_type é o nome do bloco, e deve seguir as mesmas regras para nomes de variáveis. Declarações são declarações de variáveis (tipo variável). Este bloco pode ser usado várias vezes, associando á cada novo bloco um nome. O bloco com esse novo nome terá as mesmas variáveis feitas na declaração.

Associação:

```
type (nome_type) :: nome
```

Atribuição:

```
nome = nome_type (valor das variáveis)
ou
nome%nome_variável = valor
```

Programa	Resultado
<pre>type anivers character nome*10 character mes*10 integer*1 dia end type anivers type (anivers) :: helder type (anivers) :: luciene type (anivers) :: carlo helder=anivers('Helder','Fevereiro',17) carlo=anivers('Carlo','Fevereiro',12)</pre>	<pre>Fevereiro Helder 17 Carlo 12 Luciene Agosto 22</pre>

```

luciene%nome='Luciene'

luciene%mes='Agosto'

luciene%dia=22

write(*,*) helder%mes

write(*,*) helder%nome,helder%dia

write(*,*) carlo%nome,carlo%dia

write(*,'(1x,a,a,i2)') luciene

end

```

DATA

O modo mais estruturado de se atribuir valores iniciais valores a variáveis é pelo comando 'data'. As diferenças entre o 'data' e 'parameter' é que as variáveis declaradas em 'data' podem alterar seu valor durante o programa, e essas variáveis podem ser vetores ou matrizes.

```

data var1/valor1/,var2/valor2/,...,varn/valorn/

ou

data var1,var2,...,varn/valor1,valor2,...,valorn/

```

Os valores de vetores e matrizes devem ser declarados todos de uma vez, com separação entre eles feita por vírgula.

Programa	Resultado
integer h(3,4),a,b,c	010203040506
logical l1,l2	070809101112
character cor*5	05 03 10
data h/1,2,3,4,5,6,7,8,9,10,11,12/	T F
data a/5/,b/3/,c/10/	preto
data l1,l2,cor/.true.,.false.,'preto'/	
print '(2x,6i2.2,/,2x,6i2.2)',h	
print '(2x,3(i2.2,2x))',a,b,c	

```

print ' (2x,17,2x,17) ',11,12

print *,cor

end

```

3. Designação de Memória

COMMON

Quando muitos dados devem ser passados à funções ou subrotinas, um recurso útil é o '*common*', que pode resumir este serviço. Um mesmo '*common*' pode ser usado várias vezes e 'carregar' muitas variáveis ao mesmo tempo. Este comando deve aparecer de forma igual no início do programa principal (logo após a definição dos tipos de variáveis) e também no início de cada função ou subrotina que o utilize.

```
common /nome/lista de variáveis/nome2/lista de variáveis2,...
```

O nome do bloco é opcional, e se for repetido as variáveis serão agrupadas em um bloco de mesmo nome. A ausência do nome e o nome '/' tem o mesmo significado. As variáveis de um '*common*' sem nome não podem ser declaradas pelo comando '*data*'. Uma mesma variável não pode ocupar mais de um comando '*common*'.

Programa	Resultado
<pre> common /func/a,b </pre>	12.000
<pre> common / /c,d </pre>	2.000
<pre> common e,f </pre>	4.000
<pre> data a,b/1,2/ c=3;d=4;e=5 f=funcao(4) call subrotina(5) end ! funções e subrotinas function funcao(t) common /func/a,b a=a*2 </pre>	

```

funcao=t+a+b

return

end

subroutine subrotina(r)

common /func/a,b

common c,d,e,f

r=r+c+d+e

print '(3(2x,f6.3,/))',r,a,f

return

end

```

BLOCK DATA

As variáveis declaradas em um 'common' com nome podem receber valores iniciais de uma forma mais estruturada. No '*block data*' podem ser usados os seguintes recursos: common(com nome), parameter, data, dimension e variáveis derivadas de um type.

```

block data nome_bloco

declarações e atribuições

end

```

O 'nome_bloco' é opcional e deve seguir as mesmas regras para nomes de variáveis.

4. Modularização

INCLUDE

Declarações, atribuições, common e outros comandos que estejam em um arquivo de texto com o mesmo formato usado no programa podem ser adicionados ao programa principal através do comando '*include*'. Esses arquivos podem ser adicionados a outros arquivos '*include*' ao programa principal e a funções e subrotinas. O mesmo arquivo pode ser usado várias vezes. Os arquivos serão interpretados como parte do programa, e por isso devem seguir as mesmas regras de programa normal escrito em FORTRAN. Funções e subrotinas não podem ser declaradas em um arquivo '*include*', mas podem ser acessadas a partir deles.

```
include 'nome.ext'
```

ou

```
include "nome.ext"
```

Nome e extensão podem ser qualquer um, mas a extensão '.inc' é a mais comum para este tipo de arquivo. É permitido o uso de caminhos completos no lugar do nome.

Apêndice A: Funções Intrínsecas

1. Funções Trigonométricas

Nome	Definição	Parâmetro	Resultado
SIN (X)	seno (radianos). se x for complexo, a parte real é assumida como valor em radianos.	real ou complexo.	real*4
SIND (X)	seno (graus se x for complexo, a parte real é assumida como valor em graus.	real ou complexo	real*4
CSIN (X)	seno (radianos)	complex*4	complex*4
CDSIN (X)	seno (radianos)	complex*8	complex*8
DSIN (X)	seno (radianos)	real*8	real*8
DSIND (X)	seno (graus)	real*8	real*8
ASIN (X)	Arcoseno (radianos). retorna valores na faixa [-_ /2, _ /2]	real, x .le. 1	real*4
ASIND (X)	Arcoseno (graus) retorna valores na faixa [-90, 90]	real x .le. 1	real*4
DASIN (X)	Arcoseno (radianos). retorna valores na faixa [-_ /2, _ /2]	real*8	real*8
DASIND (X)	Arcoseno (graus) retorna valores na faixa [-90, 90]	real*8	real*8
COS (X)	coseno (radianos) se x for complexo, a parte real é assumida como valor em radianos.	real ou complexo	real*4
COSD (X)	coseno (graus) se x for complexo, a parte real é assumida como valor em graus.	real ou complexo	real*4
CCOS (X)	coseno (radianos)	complex*4	complex*4
CDCOS (X)	coseno (radianos)	complex*8	complex*8
DCOS (X)	coseno (radianos)	real*8	real*8
DCOSD (X)	coseno (graus)	real*8	real*8
ACOS (X)	Arcocoseno (radianos)) retorna valores na faixa [0, _]	real, x .le. 1	real*4
ACOSD (X)	Arcocoseno (graus) retorna valores na faixa [0, 180]	real, x .le. 1	real*4
DACOS (X)	Arcocoseno (radianos)) retorna valores na faixa [0, _]	real*8, x .le. 1	real*8
DACOSD (X)	Arcocoseno (graus)) retorna valores na faixa [0, 180]	real*8, x .le. 1	real*8
TAN (X)	tangente (radianos)	real	real*4
TAND (X)	tangente (graus)	real	real*4

DTAN (X)	tangente (radianos)	real*8	real*8
DTAND (X)	tangente (graus)	real*8	real*8
COTAN (X)	cotangente (radianos)	real. x não pode ser 0.	real*4
DCOTAN (X)	cotangente (radianos)	real*8. x não pode ser 0.	real*8
ATAN (X)	Arcotangente (radianos).) retorna valores na faixa [- /2, _ /2]	real	real*4
ATAND (X)	Arcotangente (graus).) retorna valores na faixa [-90, 90]	real	real*4
DATAN (X)	Arcotangente (radianos). retorna valores na faixa [- /2, _ /2]	real*8	real*8
DATAND (X)	Arcotangente (graus). retorna valores na faixa [-90, 90]	real*8	real*8
ATAN2 (Y,X)	Arcotangente (y / x) em radianos. retorna valores na faixa [-_ , _]	real. x e y não podem ambos ser 0.	real*4
ATAN2D (Y,X)	Arcotangente (y / x) em graus. retorna valores na faixa [-180, 180]	real. x e y não podem ambos ser 0.	real*4
DATAN2 (Y,X)	Arcotangente (y / x) em radianos retorna valores na faixa [-_ , _]	real*8 x e y não podem ambos ser 0.	real*8
DATAN2D (Y,X)▫	Arcotangente (y / x) em graus. retorna valores na faixa [-180, 180]	real*8 x e y não podem ambos ser 0.	real*8
SINH (X)	seno hiperbólico (radianos)	real	real*4
DSINH (X)	seno hiperbólico (radianos)	real*8	real*8
COSH (X)	coseno hiperbólico (radianos)	real	real*4
DCOSH (X)	coseno hiperbólico (radianos)	real*8	real*8
TANH (X)	tangente hiperbólica (radianos)	real	real*4
DTANH (X)	tangente hiperbólica (radianos)	real*8	real*8

2. Funções Genéricas

Nome	Definição	Parâmetro	Resultado
DPROD(A,B)	a*b	real*4, real*4	real*8
EOF(UNIT)	verifica o final da unidade unit	integer*2	logical
SIGN(X,Y)	retorna x com o sinal de y	real ou inteiro	real ou inteiro
ISIGN(X,Y)	retorna x com o sinal de y	inteiro	inteiro
DSIGN(X,Y)	retorna x com o sinal de y	real*8	real*8

3. Exponenciais

O número 'e' elevado a X (e^X)

Nome	Parâmetro	Resultado
CDEXP (X)	complex*16	complex*16
CEXP (X)	complex*8	complex*8
DEXP (X)	real*8	real*8
EXP (X)	real, inteiro ou complexo	mesmo que o parâmetro

4. Logaritmos

Nome	Definição	Parâmetro	Resultado
LOG (X)	logaritmo natural	real ou complexo	mesmo que o parâmetro
ALOG (X)	logaritmo natural	real*4	real*4
DLOG (X)	logaritmo natural	real*8	real*8
CLOG (X)	logaritmo natural	complex*8	complex*8
CDLOG (X)	logaritmo natural	complex*16	complex*16
LOG10 (X)	logaritmo na base 10	real	mesmo que o parâmetro
ALOG10 (X)	logaritmo na base 10	real*4	real*4
DLOG10 (X)	logaritmo na base 10	real*8	real*8

5. Máximos

Nome	Definição	Parâmetro	Resultado
MAX(X1,X2,..)	maior valor	qualquer tipo	maio tipo entre os valores
MAX0(X1,X2,..)	maior valor	inteiro	inteiro
AMAX1(X1,X2,..)	maior valor	real	real
AMAX0(X1,X2,..)	maior valor	inteiro	real
MAX1(X1,X2,..)	maior valor	real	inteiro
DMAX1(X1,X2,..)	maior valor	real*8	real*8

6. Mínimos

Semelhante as funções de máximo (MIN, MIN0, AMIN1, AMIN0, MIN1, DMIN1)

7. Restos

Resto da divisão de X por Y

Nome	Parâmetro	Resultado
MOD(X,Y)	real ou inteiro	real ou inteiro
AMOD(X,Y)	real*4	real*4
DMOD(X,Y)	real*8	real*8

8. Raiz Quadrada de X

Nome	Parâmetro	Resultado
SQRT(X)	real ou complexo	real ou complexo
DSQRT(X)	real*8	real*8
CSQRT(X)	complex*8	complex*8
CDSQRT(X)	complex*16	complex*16

9. Truncamento de X

Nome	Parâmetro	Resultado
AINT(X)	real	real
DINT(X)	real*8	real*8

10. Arredondamento de X

Nome	Parâmetro	Resultado
NINT(X)	real	inteiro
DNINT(X)	real*8	real*8
ANINT(X)	real	real
IDNINT	real*8	inteiro

11. Diferença Positiva Entre X e Y

(se $Y > X$ o valor é zero)

Nome	Parâmetro	Resultado
DIM(X,Y)	real ou inteiro	real ou inteiro
DDIM(X,Y)	real*8	real*8
DIM(X,Y)	inteiro	inteiro

12. Tipo de Dado

Nome	Definição	Parâmetro	Resultado
ALLOCATED(X)	.true. se a for vetor ou	todos	lógico

	matriz		
EPSILON(X)	menor valor que pode ser incrementado	real	real
HUGE(X)	maior valor possível	real ou inteiro	real ou inteiro
MAXEXPONENT(X)	maior expoente possível	real	real
MINEXPONENT(X)	menor expoente possível	real	real
NEAREST(X,Y)	se y é positivo retorna o maior real se negativo o menor real	real	real
PRECISION(X)	número de casas decimais	real	real
TINY(X)	menor valor positivo que pode ser armazenado	real	real

13. Transformação do Tipo de X

Nome	Parâmetro	Resultado
INT(X)	real, inteiro ou complexo	inteiro
INT1(X), INT2(X), INT4(X), INTC(X)	real, inteiro ou complexo	integer*1, integer*2, integer*4, c integer
IFIX(X)	real*4	inteiro
HFIX(X)	real, inteiro ou complexo	integer*2
JFIX(X)	real, inteiro ou complexo	integer*4
IDINT(X)	real*8	inteiro
REAL(X)	real, inteiro ou complexo	real*4
DREAL(X)	Complex*16	real*8
FLOAT(X)	Inteiro	real*4
SNGL(X)	real*8	real*4
DBLE(X)	real*8 ou complexo	real*8
DFLOAT(X)	real*8 ou complexo	real*8
CMPLX(X)	inteiro, real, complexo	complexo
DCMPLX(X)	inteiro, real, complexo	complex*16
ICHAR(X)	ASCII	inteiro
CHAR(X)	integer*4	ASCII

14. Complexos

Nome	Definição	Parâmetro	Resultado
IMAG(X)	retorna a parte imaginária	complexo	real
DIMAG(X)	retorna a parte imaginária	complex*16	real*8
AIMAG(X)	retorna a parte imaginária	complex*8	real*4
CONJG(X)	retorna o complexo conjugado	complex*8	complex*8
DCONJG(X)	retorna o complexo conjugado	complex*16	complex*16

15. Caracteres

Nome	Definição	Parâmetro	Resultado
LEN(X)	tamanho de x	character	inteiro
LGE(X,Y)	compara x e y se $x \geq y$.true.	character	logical
LGT(X,Y)	compara x e y se $x > y$.true.	character	logical
LLE(X,Y)	compara x e y se $x \leq y$.true.	character	logical
LLT(X,Y)	compara x e y se $x < y$.true.	character	logical
INDEX(X,Y)	procura y em x e retorna a posição	character	inteiro
LEN_TRIM(X)	tamanho de x menos o número de espaços	character	inteiro
SCAN(X,Y)	procura um dos caracteres de y em x e retorna o número de ocorrências	character	inteiro
VERIFY(X,Y)	procura um dos caracteres de y em x e retorna a posição da primeira não ocorrência	character	inteiro

16. Valores Absolutos de X

Nome	Parâmetro	Resultado
ABS(X)	complexo ou real	complexo ou real
IABS(X)	Inteiro	inteiro
DABS(X)	real*8	real*8
CABS(X)	complex*16	complex*16
CDABS(X)	complex*16	real*8

Os valores absolutos dos dados complexos são calculados pela seguinte fórmula:

$$\sqrt{(real)^2 + (imag)^2}$$

Apêndice B: Opções de Arquivos, Leitura e Escrita

As opções contidas aqui foram baseadas no compilador FORTRAN Visual Workbench v 1.00. Outros compiladores possuem outros nomes para estas opções, e podem possuir outras além destas.

As opções entre colchetes são opcionais. Elas podem vir em qualquer ordem, exceto quando explicitado.

1. Abertura de Arquivos (OPEN)

OPEN ([UNIT=*unit*] [, ACCESS=*access*] [, BLANK=*blanks*]

[, BLOCKSIZE=*blocksize*] [, ERR=*errlabel*] [, FILE=*file*]

[, FORM=*form*] [, IOSTAT=*iocheck*] [, MODE=*mode*]

[, RECL=*recl*] [, SHARE=*share*] [, STATUS=*status*])

UNIT= - Quando omitido, o primeiro valor será o número da unidade (*unit*).

unit – Expressão inteira. Indica o número da unidade.

access – Expressão alfanumérica. ‘APPEND’ , ‘DIRECT’ ou ‘SEQUENTIAL’ (padrão, quando *access* é omitido)

blank – Expressão alfanumérica. ‘NULL’ ignora zeros e espaços, ‘ZERO’ espaços são substituídos por zeros. Os formatos BN e BZ podem anular este efeito.

blocksize – Expressão inteira. Especifica o tamanho da unidade (em bytes).

errlabel – Expressão inteira. Indica o rótulo de uma linha no mesmo arquivo para onde o comando vai se houver erro. Quando omitido o efeito é determinado pela presença ou não de *iocheck*.

file – Expressão alfanumérica. Indica o nome do arquivo. Quando omitido o programa pede ao usuário um nome.

form – Expressão alfanumérica. ‘FORMATED’ (padrão quando *access*=‘SEQUENTIAL’), ‘UNFORMATED’ (padrão quando *access*=‘DIRECT’), ‘BINARY’.

iocheck – Variável inteira. Retorna zero quando não ocorre erros, retorna um número negativo se encontrar a marca de fim de arquivo (EOF), retorna o número do erro quando um ocorre.

mode – Expressão alfanumérica. ‘READ’ o arquivo é só para leitura, ‘WRITE’ o arquivo é só para escrita, ‘READWRITE’ o arquivo pode ser usado para leitura ou escrita.

recl – Expressão inteira. Representa o tamanho de cada dado em bytes. É obrigatório para *access*=‘DIRECT’.

share – Expressão alfanumérica. 'DENYRW' ignora *mode*='READWRITE', 'DENYWR' ignora *mode*='WRITE', 'DENYRD' ignora *mode*='READ', 'DENYNONE' ignora qualquer *mode*,

status – Expressão alfanumérica. 'OLD' indica que o arquivo já existe, 'NEW' indica que o arquivo deve ser criado, 'UNKNOWN' (padrão) verifica a existência do arquivo, se ele não existir será criado um, 'SCRATCH'

2. Fechamento de Arquivos (CLOSE)

CLOSE ([UNIT=*unit*] [,ERR=*errlabel*] [,IOSTAT=*iocheck*]
[,STATUS=*status*])

UNIT= - Quando omitido, o primeiro valor será o número da unidade (*unit*).

unit – Expressão inteira. Indica o número da unidade.

errlabel – Expressão inteira. Indica o rótulo de uma linha no mesmo arquivo para onde o comando vai se houver erro. Quando omitido o efeito é determinado pela presença ou não de *iocheck*.

iocheck – Variável inteira. Retorna zero quando não ocorre erros, retorna um número negativo se encontrar a marca de fim de arquivo (EOF), retorna o número do erro quando um ocorre.

status – Expressão alfanumérica. 'KEEP' indica que o arquivo deve ser mantido, 'DELETE' indica que o arquivo deve ser apagado.

3. Escrita (WRITE)

WRITE ([UNIT=*unit*] [, [{[FMT=*format*] | [NML=*nml*] }]
[, ERR=*errlabel*] [, IOSTAT=*iocheck*] [, REC=*rec*]) *iolist*

UNIT= - Quando omitido, o primeiro valor será o número da unidade (*unit*). Se FMT= ou NML= forem omitidos, *format* ou *nml* devem ser o segundo parâmetro.

unit – Expressão inteira. Indica o número da unidade.

format – Expressão inteira. Indica o rótulo de um comando 'format'. Expressão alfanumérica. Expressão que contenha os formatos de impressão.

nml – Lista de variáveis a serem impressas, se estiver presente *iolist* deve ser omitida.

errlabel – Expressão inteira. Indica o rótulo de uma linha no mesmo arquivo para onde o comando vai se houver erro. Quando omitido o efeito é determinado pela presença ou não de *iocheck*.

iocheck – Variável inteira. Retorna zero quando não ocorre erros, retorna o número do erro quando um ocorre.

rec – Expressão inteira. Indica a posição do arquivo onde o dado será impresso. Somente para arquivos com *access='DIRECT'*.

iolist – Lista de variáveis á serem impressas.

4. Leitura (READ)

```
READ { { format, | nml } | ([UNIT=]unit [ , [ {FMT=] format] |  
[NML=]nmlspec] [ , END=endlabel] [ , ERR=errlabel] [ , IOSTAT=iocheck]  
[ , REC=rec])} iolist
```

UNIT= - Quando omitido, o primeiro valor será o número da unidade (*unit*). Se FMT= ou NML= forem omitidos, *format* ou *nml* devem ser o segundo parâmetro.

unit – Expressão inteira. Indica o número da unidade.

format – Expressão inteira. Indica o rótulo de um comando 'format'. Expressão alfanumérica. Expressão que contenha os formatos de leitura.

nml – Lista de variáveis á serem lidas, se estiver presente *iolist* deve ser omitida.

endlabel – Expressão inteira. Indica o rótulo de uma linha no mesmo arquivo para onde o comando vai se a marca de fim de arquivo (EOF) for encontrada.

errlabel – Expressão inteira. Indica o rótulo de uma linha no mesmo arquivo para onde o comando vai se houver erro. Quando omitido o efeito é determinado pela presença ou não de *iocheck*.

iocheck – Variável inteira. Retorna zero quando não ocorre erros, retorna 1 se encontrar EOF, retorna o número do erro quando um ocorre.

rec – Expressão inteira. Indica a posição do arquivo de onde o dado será lido.

iolist – Lista de variáveis á serem lidas.

5. Recuo Total (REWIND)

```
REWIND { unit | ([UNIT=]unit [ , ERR=errlabel] [ , IOSTAT=iocheck])}
```

UNIT= - Quando omitido, o primeiro valor será o número da unidade (*unit*).

unit – Expressão inteira. Indica o número da unidade.

errlabel – Expressão inteira. Indica o rótulo de uma linha no mesmo arquivo para onde o comando vai se houver erro. Quando omitido o efeito é determinado pela presença ou não de *iocheck*.

iocheck – Variável inteira. Retorna zero quando não ocorre erros, retorna o número do erro quando um ocorre.

6. Recuo de um Campo (BACKSPACE)

BACKSPACE {*unit*([UNIT=] *unit* [,ERR=*errlabel*] [,IOSTAT=*iocheck*]) }

UNIT= - Quando omitido, o primeiro valor será o número da unidade (*unit*).

unit – Expressão inteira. Indica o número da unidade.

errlabel – Expressão inteira. Indica o rótulo de uma linha no mesmo arquivo para onde o comando vai se houver erro. Quando omitido o efeito é determinado pela presença ou não de *iocheck*.

iocheck – Variável inteira. Retorna zero quando não ocorre erros, retorna o número do erro quando um ocorre.

Apêndice C: Tabela de Valores ASCII

Valor ASCII – Caracter Correspondente

0		32		64	@	96	`	128	Ç	160	á	192	L	224	ó
1	☒	33	!	65	A	97	a	129	ü	161	í	193	l	225	õ
2	☒	34	"	66	B	98	b	130	é	162	ó	194	T	226	ô
3	♥	35	#	67	C	99	c	131	â	163	ú	195	†	227	ò
4	♦	36	\$	68	D	100	d	132	ä	164	ñ	196	—	228	õ
5	♣	37	%	69	E	101	e	133	à	165	ñ	197	†	229	õ
6	♠	38	&	70	F	102	f	134	ã	166	±	198	ã	230	µ
7	•	39	'	71	G	103	g	135	ç	167	±	199	ã	231	þ
8	◼	40	(72	H	104	h	136	ê	168	¿	200	u	232	þ
9	◊	41)	73	I	105	i	137	ë	169	Ⓞ	201	ü	233	ú
10	◼	42	*	74	J	106	j	138	è	170	¬	202	±	234	û
11	♂	43	+	75	K	107	k	139	ï	171	½	203	ü	235	ù
12	♀	44	,	76	L	108	l	140	î	172	¼	204	ü	236	ú
13	♯	45	_	77	M	109	m	141	ì	173	¡	205	=	237	ý
14	♪	46	.	78	N	110	n	142	ñ	174	«	206	ü	238	˘
15	✱	47	/	79	O	111	o	143	ñ	175	»	207	⊗	239	˙
16	▶	48	0	80	P	112	p	144	é	176	▒	208	ø	240	-
17	◀	49	1	81	Q	113	q	145	æ	177	▒	209	ø	241	±
18	‡	50	2	82	R	114	r	146	ff	178	▒	210	ê	242	=
19	!!	51	3	83	S	115	s	147	ô	179		211	ë	243	¼
20	¶	52	4	84	T	116	t	148	ö	180	†	212	è	244	¶
21	§	53	5	85	U	117	u	149	ò	181	Á	213	í	245	§
22	—	54	6	86	V	118	v	150	û	182	Â	214	î	246	÷
23	‡	55	7	87	W	119	w	151	ù	183	À	215	ï	247	˘
24	↑	56	8	88	X	120	x	152	ÿ	184	Ⓞ	216	ÿ	248	°
25	↓	57	9	89	Y	121	y	153	ö	185	¶	217	J	249	˙˙
26	→	58	:	90	Z	122	z	154	ü	186	¶	218	Γ	250	˙
27	←	59	;	91	[123	{	155	ø	187	¶	219	▀	251	¡
28	↵	60	<	92	\	124	!	156	£	188	¶	220	▀	252	³
29	↗	61	=	93]	125	}	157	Ø	189	Ç	221	!	253	²
30	▲	62	>	94	^	126	~	158	×	190	¥	222	!	254	■
31	▼	63	?	95	_	127	△	159	f	191	†	223	▀	255	