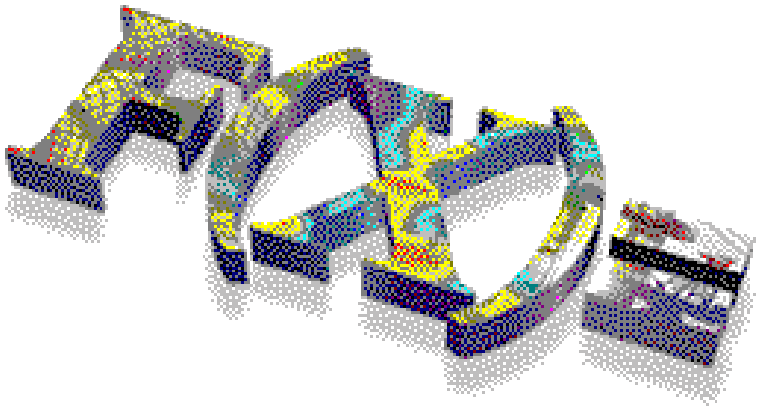
UNIVERSIDADE DE SÃO PAULO
ESCOLA DE ENGENHARIA DE SÃO CARLOS
DEPARTAMENTO DE ENGENHARIA DE ESTRUTURAS



FORTRAN PowerSTATION versão 4.0

Autor: EDUARDO WALTER VIEIRA CHAVES

São Carlos - 1997

AGRADECIMENTO

A Prof[a.] Helena por ter promovido o curso de FORTRAN na pós-graduação da EESC-USP.

A Eliana A. Bertin por ajudar na confecção desta apostila e pelas cabíveis correções.

A todos que contribuíram na formação final deste texto.

Resumo

CHAVES, E.W.V. (1997). ***FORTRAN PowerStation versão 4.0***. São Carlos. Departamento de Engenharia de Estruturas - Escola de Engenharia de São Carlos, Universidade de São Paulo. 75p.

Esta apostila é destinada aos alunos da disciplina **SET-808 - Análise Matricial de Estruturas**, a qual é oferecida pelo Departamento de Engenharia de Estruturas da Escola de Engenharia de São Carlos da Universidade de São Paulo, ministrada pela Prof.ª Dra. Helena Maria C. Carmo Antunes. O objetivo é fornecer aos alunos da disciplina uma noção básica da linguagem de programação FORTRAN, que tem sido bastante utilizada no âmbito de Engenharia de Estruturas na resolução de problemas que envolvem grande manipulação matemática e requerem espaço de memória relativamente grande.

Palavras-chave: FORTRAN, PowerStation versão 4.0

SUMÁRIO

1 NOÇÕES BÁSICAS

Os computadores eletrônicos digitais estão desempenhando um papel cada vez mais significativo nos diversos setores de aplicação da vida atual, quer nas atividades de pesquisa básica e tecnológica, quer nas atividades acadêmicas-universitárias.

A codificação dos algoritmos elaborados de acordo com os princípios da Programação Estruturada pode ser feita, em geral, em qualquer linguagem de programação.

A linguagem de programação FORTRAN (FORmula TRANslation) foi a primeira linguagem de programação de alto nível a ser proposta. Surgiu em 1956 com o objetivo de resolver problemas da área científica, através do uso de computadores. Ainda hoje, esta linguagem é muito difundida no meio técnico científico e tem sido aprimorada ao longo do tempo.

Existem versões como: FORTRAN II, FORTRAN IV, WAFTOR, FORTRAN 77, outras versões como Microsoft FORTRAN PowerStation 1.0. Neste trabalho, será utilizado a versão da Microsoft FORTRAN PowerStation 4.0, para o *Windows* 95.

2 AMBIENTE *FORTRAN*

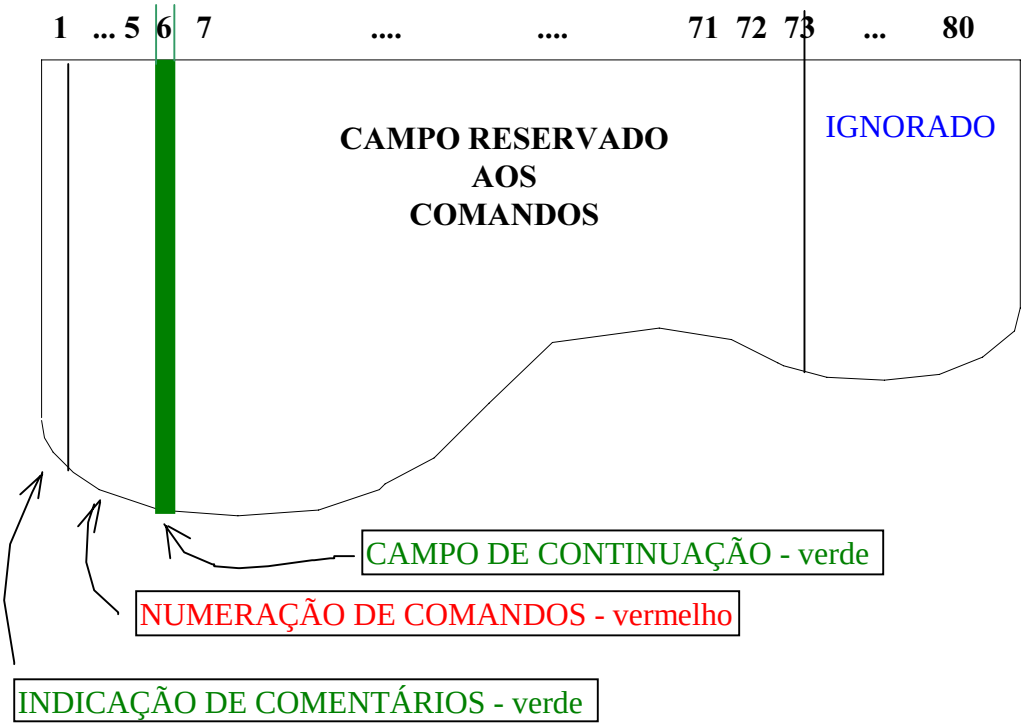


FIGURA 1.1 - Ambiente FORTRAN

Observações:

1. Num programa FORTRAN, os comandos devem ser escritos em campos delimitados e fixos. A figura 1.1 resume a codificação que deve ser usada.
2. Os comentários são ignorados pelo compilador, mas são essenciais a um programa bem documentado. Qualquer caracter não numérico, na coluna 1, indica linha de comentário, que pode ser inserida em qualquer parte do programa. Outra forma de

comentário é o ponto de exclamação, que pode aparecer em qualquer coluna do programa.

3. Para indicar a continuação de um comando, utiliza-se um caractere qualquer (diferente de espaço ou zero) na coluna 6 e pode haver, no máximo, 19 linhas de continuação. A coluna 6 é de cor verde.
4. Os números dos comandos são inteiros e positivos, consistem de 1 a 5 dígitos e devem ser delimitados pelas colunas 1 e 5, as quais ficam escritas em vermelho e os dígitos não precisam estar em ordem. Nem todos os comandos precisam ser numerados.
5. As colunas 73 a 80 são ignoradas pelo compilador e podem ser utilizadas, por exemplo, para informações sobre os comandos ou programa. Estas colunas ficam escritas em azul. A partir da coluna 80, passam para o vermelho, indicando que o limite foi ultrapassado.
6. Espaços em branco, em geral, são ignorados.

É bom salientar que o formato de cores apresentado anteriormente é o formato padrão, podendo então ser alterado, o que não aconselhamos.

3 REPRESENTAÇÃO DE ALGORITMOS

Um algoritmo é uma seqüência ordenada de passos executáveis e precisamente definidos, que manipulam um volume de informações, a fim de obter um resultado. Existem inúmeras formas de representar um algoritmo, como por exemplo, informações oferecidas a terceiros, como: receita de bolo, ensinar a pegar um ônibus, etc. No entanto, quanto mais detalhadas forem estas informações, mais fácil será o caminho a seguir. O mesmo acontece com os algoritmos destinados a resolver problemas com auxílio de um computador, o qual exige que todos os passos sejam bem definidos e que tenham boa consistência durante a execução. Desta forma pode-se representar estes algoritmos de várias formas, como por exemplo, Diagramas de Fluxo, Diagramas de Nassi-Shneiderman e outras.

Aqui será visto apenas o Diagrama de Fluxo ou simplesmente chamado de Fluxograma.

3.1 Fluxograma

Os fluxogramas fornecem uma representação gráfica de um procedimento passo-a-passo, necessário para resolver um problema particular, isto é, para um algoritmo. Tal representação torna uma seqüência complexa de eventos fácil para se ver e compreender.

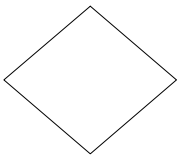
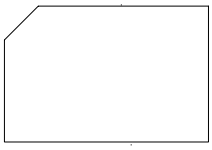
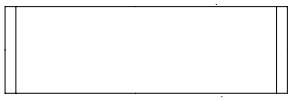
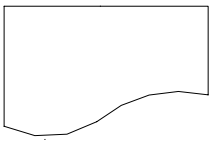
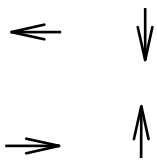
A comunidade de programadores tem aceito um conjunto padrão de símbolos para representar diagramas de fluxo.

As três finalidades mais importantes para o diagrama de fluxo são:

- Como auxiliar no desenvolvimento do projeto de programa.
- Como guia para a codificação do programa.
- Como parte da documentação do programa.

É recomendável que se faça um diagrama de fluxo para todos os programas, por mais simples que eles sejam, até que se torne um hábito pensar em diagrama de fluxo antes de se pensar em programa.

TABELA 3.1 - Alguns símbolos usados em diagramas de fluxo

SÍMBOLO	DESCRIÇÃO
	PROCESSAMENTO - Uma instrução ou um conjunto de instruções que indicam computação.
	DECISÃO - Indicação da possibilidade de desvios para outros pontos do programa de acordo com certas condições
	TERMINAL - Usado no início e no final de um diagrama de fluxo.
	CARTÃO PERFURADO - Principalmente usado para entrada de dados.
	SUBPROGRAMA EXTERNO - Referência a um subprograma externo ao diagrama de fluxo.
	DOCUMENTO/RELATÓRIO - Principalmente usado para saída de resultados.
	DIREÇÃO DO FLUXO - Indicação da ordem de execução das instruções.

4 CONSTANTES E VARIÁVEIS

4.1 Constantes

São valores fixos, tais como números. Estes valores não podem ser alterados durante a execução do programa, pois não existe comando que altere estes valores. O FORTRAN distingue três classes de constantes: as que tratam com números, chamadas de constantes numéricas; as que tratam com valores lógicos, chamadas constantes lógicas, e as que tratam com cadeias de caracteres.

As constantes numéricas podem ser:

- Inteiras - para números inteiros decimais, escritos sem o ponto decimal.
- Reais - para números decimais, isto é, um número inteiro decimal escrito com ponto decimal ou uma fração decimal ou uma combinação de um número inteiro decimal com uma fração decimal.
- Dupla Precisão - para números decimais com mais dígitos significativos do que o máximo permitido para as constantes reais.
- Complexas - para números complexos.
- Hexadecimais - para números inteiros hexadecimais, isto é, do sistema de numeração de base 16.

4.2 Variáveis

Uma variável algébrica é, geralmente, representada por símbolo(usualmente, uma única letra, tal como x ou y) e pode assumir diferentes valores. Em FORTRAN, uma variável é uma entidade que tem um nome e um tipo, podendo assumir diferentes valores.

As variáveis podem ser classificadas em seis tipos válidos:

- Inteiro
- Real
- Dupla Precisão
- Complexo
- Lógico
- Caractere

Pode-se declarar o tipo de uma variável por uma das três maneiras:

1. Através de comandos de especificação explícita de tipo de variável.
2. Através do comando IMPLICIT.
3. Através da convenção predefinida da linguagem FORTRAN, escolhendo nomes de variáveis que comecem com as letras (I,J,K,L,M e N) para variáveis do tipo inteiro e qualquer outra letra para variáveis do tipo real. Esta convenção é o método tradicional FORTRAN de especificar o tipo de uma variável como sendo inteiro ou real.

5 FUNÇÕES INTRÍNSECAS

As funções intrínsecas são certos procedimentos de utilidade geral, supridos pelo compilador **FORTRAN**, que, consequentemente, não precisam ser providos no programa. Essas funções são usualmente identificadas por: um nome genérico e por nomes específicos, como por exemplo, a função intrínseca de nome genérico **LOG**, que calcula o logaritmo natural de um argumento. Este argumento pode ser do tipo real, dupla precisão ou complexo e o resultado é do mesmo tipo do argumento. As funções intrínsecas de nome específico **ALOG**, **DLOG** e **CLOG** também calculam o logaritmo natural de um argumento. No entanto, a função **ALOG** calcula o logaritmo de um argumento real e retorna um resultado real. Do mesmo modo, a função **DLOG** é para argumento e resultado de dupla precisão, e a função **CLOG** é para argumento e resultado em complexo.

- Função Valor Absoluto - **ABS** (1 argumento)

  Exemplos: ABS(1)=1 ; ABS(1.0) = 1.0 ; ABS(-1)=1

- Função Módulo Aritmético - **MOD**(2 argumentos)

  Resto da divisão do primeiro pelo segundo argumento.

  Exemplo: MOD(17,5) = 2 ; MOD(3.124,1.0) = 0.124

- Função comprimento de uma cadeia de caracteres - **LEN** (1 argumento)

  Exemplo: LEN(‘A’) = 1 ; LEN(‘abcd’) = 4

- Índice de um Caracter - **INDEX**(2 argumentos)

  Localização da subcadeia do segundo argumento interna á cadeia do primeiro argumento.

  Exemplo: INDEX(‘abcd’, ‘b’) = 2 ; INDEX(‘ABCABC’,’BC) = 2

- Função raiz quadrada - **SQRT**(1 argumento)

  Exemplo: SQRT(16.0) = 4.0 ; SQRT(2.0) = 1.4142135

  DSQRT(2.0) = (dupla precisão)

- Função exponencial - **EXP**(1 argumento)

  $\mathbf{e^x}$, onde **e=2.71828**...

  Exemplo: EXP(1.0) = 2.71828 ; EXP(2.0) = 7.38905

- Função Logaritmo Natural(ou Neperiano) - **LOG** (1 argumento)

  $\mathbf{\log_e x}$ ou **ln x**

  Exemplo: LOG(2.71828)=1.0 ; LOG(7.38905) = 2.0

- Função Logaritmo comum (ou decimal) - **LOG10**(1 argumento)

  $\mathbf{\log_{10} x}$

  Exemplo: LOG10(100.0) = 2.0 ; LOG10(1000.0) = 3.0

Funções Trigonométricas, Trigonométricas Inversas e Hiperbólicas

- Seno Trigonométrico, com argumento em radianos - **SIN**(1 argumento)

  Exemplo: SIN(3.14159)= 0.0

- Seno Trigonométrico, com argumento em graus - **SIND**(1 argumento)

  Exemplo: SIND(180) = 0.0

- Co-seno Trigonométrico, com argumento em radianos - **COS**(1 argumento)

  Exemplo: COS(3.14159) = 1.0

- Co-seno Trigonométrico, com argumento em graus - **COSD**(1 argumento)

  Exemplo: COSD(180) = 1.0

- Tangente trigonométrico, com argumento em radianos - **TAN**(1 argumento)

  Exemplo: TAN(3.14159/4.0) = 1.0

- Tangente trigonométrico, com argumento em graus - **TAND**(1 argumento)

- Arco-seno trigonométrico(ou seno trigonométrico inverso), em radianos

  ASIN(1 argumento)

  Exemplo: ASIN(0.0) = 0.0

  Obs.: O intervalo do resultado é : $-\pi / 2 \leq$ resultado $\leq \pi / 2$.

- Arco-co-seno trigonométrico(ou co-seno trigonométrico inverso), em radianos

  ACOS(1 argumento)

  Exemplo: ACOS (0.0) = 1.5708

  Obs.: O intervalo do resultado é : $0 \leq$ resultado $\leq \pi$.

- Arco-tangente trigonométrico(ou tangente trigonométrico inverso), em radianos

  ATAN(1 argumento)

  Exemplo: ATAN(1.0) = 0.7854

  Obs.: O intervalo do resultado é : $-\pi / 2 \leq$ resultado $\leq \pi / 2$.

- Seno hiperbólico - **SINH**(1 argumento)

  Exemplo SINH(1.0)=1.1752

- Co-seno hiperbólico - **COSH**(1 argumento)

  Exemplo: COSH(1.0)=1.5431

- Tangente hiperbólico - **TANH**(1 argumento)

  Exemplo: TANH(1.0)= 0.7616

- Arco-Tangente hiperbólico - **TANH**(1 argumento)

6 OPERADORES DE EXPRESSÕES

Uma **expressão** FORTRAN é definida como uma combinação de itens sintáticos, unidos com um ou mais **operadores**, que agrupados com parênteses, representam um único valor.

Exemplo: X+Y, consiste em duas variáveis unidas pelo operador ”+”.

As expressões utilizadas no FORTRAN, são classificadas em:

- aritméticas;
- caracteres;
- relacionais;
- lógicas.

6.1 Expressões Aritméticas

As expressões aritméticas são formadas com a utilização de operadores numéricos combinados com operadores aritméticos.

O cálculo de uma expressão aritmética produz um valor numérico inteiro, real, dupla precisão ou complexo.

Os operadores aritméticos e seus significados são apresentados na Tabela 6.1.

TABELA 6.1 - Operadores Aritméticos

Operador	*Definição*	*Uso do Operador*	*Significado*
**	Potenciação	**a******b**	**a** elevado à potência **b**
*	Multiplicação	**a*****b**	**a** multiplicado por **b**
/	Divisão	**a/b**	**a** dividido por **b**
+	Adição	**a+b**	**a** mais **b**
+	Mais unário	**+a**	o mesmo que **a**
-	Subtração	**a-b**	**a** menos **b**
-	Menos unário	**-a**	**a** com sinal trocado

Exemplos

X = -Y , significa tomar o valor negativo de Y e armazená-lo em X.

O FORTRAN não permite que dois ou mais operadores aritméticos apareçam consecutivamente na expressão, como por exemplo:

A+/B , A*/B e A*-B

A expressão A*-B pode ser escrita corretamente como: A*(-B).

Regras com dois ou mais operadores.

5x+3 significa ” multiplicar 5 por x e então adicionar 3”. Isto é, 5x+3 tem o significado de

(5x)+3

que é diferente de

5(x+3)

Parênteses podem ser usados para agrupar operandos numa expressão. Qualquer expressão entre parênteses é completamente calculada antes do cálculo da expressão da qual ela é uma parte. (parênteses mais internos são calculados antes).

Com exceção do uso do parênteses, as operações são realizadas na seguinte ordem de precedência:

**	Mais alta
* e /	Intermediária
+ e -	Mais baixa

Para operadores de mesma precedência, o cálculo se desenvolve da esquerda para a direita. Então:

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

têm o mesmo efeito que

A**(B**C)

Exemplos:

Na expressão

-X**2

o operador potenciação tem precedência sobre o operador menos unário. Então, a expressão é tratada como

-(X**2)

Seja a seguinte expressão aritmética:

```
4 + 3 * 2 - 6 / 2 = 7
    ↑   ↑   ↑   ↑
    2   1   4   3   Ordem de operação
```

```
((4 + 3) * 2 - 6) / 2 = 7
  ↑     ↑   ↑   ↑
  1     2   3   4   Ordem de operação
```

Pelo fato das informações sobre operações em expressões aritméticas serem um ponto importantíssimo em programação de computadores, vamos detalhar cada uma das operações, classificadas pelos tipos de dados envolvidos:

- *Operações Inteiras*: são operações realizadas somente sobre operadores inteiros. Na aritmética inteira, qualquer fração que resulte da divisão não é arredondada, mas sim truncada, como na expressão: 1/3 + 1/3 + 1/3, cujo valor final é zero e não 1. Na extração da raiz quadrada de um número, na forma 16**(1/2), resulta em um expoente zero e não meio.

- *Operações Reais*: são operações realizadas somente sobre operandos reais ou combinações de operandos reais e inteiros.

- *Operações de Dupla Precisão*: são operações, onde qualquer operando real ou inteiro é convertido para dupla precisão, fazendo com que o mesmo seja a parte mais significativa do elemento de dupla precisão. Para a parte menos significativa é dado o valor zero. A expressão é, então, calculada em aritmética de dupla precisão. Note que a conversão de um elemento real para um elemento de dupla precisão não aumenta sua precisão. Por exemplo, o número real:

  0.3333333

  é convertido para

  0.33333330000...0D0

  e não para

  0.3333333...3D0

6.2 Expressões Caracteres

Os dados do tipo caractere são armazenados na memória do computador de modo diferente dos dados numéricos. Portanto, não é possível realizar operações aritméticas sobre dados caracteres como se faz com dados numéricos. A constante caractere

'12345'

é diferente da constante numérica

12345

As duas constantes são semelhantes na escrita, porém, são inteiramente diferentes na memória do computador.

O cálculo de uma expressão caractere produz um resultado do tipo caractere.

Tabela 1.2 - Operador Caractere

Operador	*Definição*	*Uso do Operador*	*Significado*
//	Concatenação	**a//b**	**a** encadeado com **b**

Concatenar duas cadeias significa encadeá-las ou juntá-las como em uma corrente. O resultado de uma operação de concatenação é uma cadeia de caracteres, encadeaNdo o primeiro operando imediatamente com o segundo operando. Por exemplo:

'AB'//'CDEF'

é a cadeia

'ABCDEF'

6.3 Expressões Relacionais

Uma expressão relacional é usada para comparar os valores de duas expressões aritméticas, ou os valores de duas expressões caracteres.

A comparação entre duas expressões aritméticas ou as duas expressões caracteres é feita com um dos seis operadores relacionais, apresentados na tabela 6.2, os quais têm o mesmo grau de precedência entre si.

TABELA 6.2 - Operadores Aritméticos

Operador	*Definição*	*Uso do Operador*	*Significado*
.EQ.	Igual a	**a.EQ.b**	**a = b**
.NE.	Diferente de	**a.NE.b**	**a ≠ b**
.LT.	menor que	**a.LT.b**	**a < b**
.LE.	Menor ou igual a	**a.LE.b**	**a ≤ b**
.GT.	Maior que	**a.GT.b**	**b > a**
.GE.	Maior ou igual a	**a.GE.b**	**a ≥ b**
***Nota*:** os pontos delimitadores são parte de cada operador.			

O cálculo de uma expressão relacional produz um resultado lógico com um valor “verdade” ou “falso” .

6.4 Expressões Lógicas

Uma expressão lógica é usada para expressar uma computação lógica. O cálculo de uma expressão lógica produz um resultado do tipo lógico, com um valor correspondendo a “verdade” ou “falso”.

Os operadores lógicos e seus significados são apresentados na Tabela 6.3.

TABELA 6.3 - Operadores Lógicos

Operador	*Definição*	*Uso do Operador*	*Significado*
.NOT.	Negação lógica	**.NOT.a**	Complemento de a: se a é verdade, então .NOT.a é falso; se a é falso, então .NOT.a é verdade
.AND.	Conjunção lógica	**a.AND.b**	Produto Booleano de a por b: se a e b são verdade, então a.AND.b é verdade; se a ou b ou ambos são falso, então a.AND.b é falso
.OR.	Disjunção inclusivo lógica	**a.OR.b**	Soma Booleana de a com b: se a ou b ou ambos são verdade, então a.OR.b é verdade; se a e b são falso, então a.OR.b é falso
.EQV.	Equivalência lógica	**a.EQV.b**	Equivalência lógica de a com b: se a e b são ambos verdade ou ambos falso, então a.EQV.b é verdade; caso contrário, é falso.
.NEQV.	Não-equivalência lógica	**a.NEQV.b**	Não-equivalência lógica de a com b: se a e b são ambos verdade ou ambos falso, então a.NEQV.b é falso; caso contrário, é verdade.
Nota**:** os pontos delimitadores são parte de cada operador.			

Exemplo:

Em FORTRAN, a condição algébrica

1 ≤ A ≤ 5

escreve-se da seguinte forma:

1**.LE.**A**.AND.**A**.LE.**5

A Tabela 6.4 é chamada tabela da verdade. Ela mostra todas as possibilidades de resultados da aplicação de qualquer operador lógico a duas expressões lógicas A e B.

TABELA 6.4 - Tabela da Verdade para operadores Lógicos.

A	B	.NOT.A	A.OR.B	A.AND.B	A.EQV.B	A.NEQV.B
.TRUE.	.TRUE.	.FALSE.	.TRUE.	.TRUE.	.TRUE.	.FALSE.
.TRUE.	.FALSE.	.FALSE.	.TRUE.	.FALSE.	.FALSE.	.TRUE.
.FALSE.	.TRUE.	.TRUE.	.TRUE.	.FALSE.	.FALSE.	.TRUE.
.FALSE.	.FALSE.	.TRUE.	.FALSE.	.FALSE.	.TRUE.	.FALSE.

6.5 Comando de Atribuição

O comando de atribuição é usado para atribuir (ou definir) um valor a uma variável ou a um elemento de conjunto, isto é, os comandos de atribuição calculam uma expressão e atribuem o valor resultante a uma variável ou a um elemento de conjunto. Por exemplo, o comando de atribuição

BETA=1.5

diz para armazenar o valor da constante 1.5 na variável de nome BETA.

Os comandos de atribuição são classificados como:

- aritméticos;
- caractere;
- lógico.

7 DECLARAÇÃO DE TIPOS DE VARIÁVEIS

Muitas linguagens de alto nível exigem que as variáveis sejam declaradas antes de serem usadas. Sob este ponto de vista, declaração significa especificar o tipo de cada variável. Em FORTRAN, todas as variáveis têm automaticamente, especificadas um tipo de acordo com a letra inicial de seu nome, a menos que o tipo seja alterado por um comando de declaração de tipo.

Os comandos de especificação que declaram o tipo de variáveis e de conjuntos são os seguintes:

- IMPLICIT - declara implicitamente, o tipo do nome simbólico como: inteiro, real, dupla precisão, complexo, lógico ou caractere.
- INTEGER - declara explicitamente, o tipo do nome simbólico como inteiro.
- REAL - declara explicitamente, o tipo do nome simbólico como real.
- DOUBLE PRECISION - declara explicitamente, o tipo do nome simbólico como de dupla precisão.
- COMPLEX - declara explicitamente, o tipo do nome como complexo.
- LOGICAL - declara explicitamente, o tipo do nome simbólico como lógico.
- CHARACTER - declara explicitamente, o tipo do nome simbólico como caractere.

7.1 Comando IMPLICIT

O comando de especificação IMPLICIT permite que você altere ou confirme a especificação por convenção predefinida que ocorre de acordo com a primeira letra dos nomes simbólicos. Pela convenção, todos os nomes que começam com as letras I

até N, são interpretados para ser do tipo inteiro, e todos os nomes que começam com qualquer outra letra são interpretados para ser do tipo real.

Por exemplo, a especificação predefinida do FORTRAN poderia ser expressa pelo comando:

IMPLICIT REAL (A-H), INTEIRO (I-N), REAL (O-Z)

ou ainda:

IMPLICIT REAL (A-H,O-Z), INTEGER (I-N)

Isto significa que todas as variáveis que se iniciam com a letra A até H são do tipo real, todas as variáveis que se iniciam com a letra I até N são do tipo inteiro, e as que começam com a letra O até Z são do tipo real.

Obs.: As variáveis que forem definidas explicitamente podem começar com qualquer letra, mesmo que a primeira letra da variável tenha sido definida implicitamente como de outro tipo.

7.2 Comandos de Especificação Explícita

O comando de especificação explícita segue as seguintes regras:

1. Os comandos de especificação explícita devem preceder todos os comandos executáveis.
2. Um nome simbólico individual só pode ser declarado como sendo de um único tipo.
3. Um conjunto de especificação explícita pode ser usado para suprir informação de dimensão para um conjunto, isto é, apondo-se ao nome do conjunto um declarador de conjunto.

7.3 Comando INTEGER

O comando INTEGER é usado para declarar explicitamente, um nome de variável, de conjunto, de constante simbólica ou de função, como sendo de tipo inteiro.

Forma Geral

INTEGER nome [,nome]...

Exemplos:

INTEGER I,N,B29

INTEGER CONT, MATRIZ(5,5)

7.4 Comando REAL

O comando REAL é usado para declarar explicitamente, um nome de variável, de conjunto, de constante simbólica ou de função, como sendo do tipo real.

Forma Geral

REAL nome[,nome]...

Exemplos:

REAL MANO, VB

REAL X(20), L, SYST, a

7.5 Comando DOUBLE PRECISION

O comando DOUBLE PRECISION é usado para declarar, explicitamente, um nome de variável, de conjunto, de constante simbólica, ou de função, como sendo do tipo de dupla precisão.

Forma Geral

DOUBLE PRECISION nome [,nome]...

Exemplo:

```
DOUBLE PRECISION DPRE,DP
DOUBLE PRECISION cvf, VD(150)
```

Pode-se declarar variáveis de dupla precisão da seguinte forma:

```
REAL*8 DPRE,DP
REAL*8 cvf, VD(150)
```

OBS.: Para atribuir valor nulo a uma variável de dupla precisão aconselha-se seguir o formato: 0.0d0. Considere dois exemplos processados no *PowerStation* v. 4.0:

	REAL*8 et 100 et= 0.0000000045365 et= et*10000000	REAL*8 et et= 0.0000000045365d0 et= et*10000000
Resultados obtidos:	et=4,536500064489246E-03	et=4,536500000000000E-03

O que aconteceu neste exemplo, é que no primeiro caso o programa armazenou em **et** na linha 100 o valor de et= 4,536500064489246E-10, isto é, veio com lixo. Já no segundo caso o valor armazenado para **et** foi et=4,536500000000000E-10.

7.6 Comando CHARACTER

O comando CHARACTER é usado para declarar, explicitamente, um nome de variável, de um conjunto, de constante simbólica, ou de função, como sendo do tipo caractere.

Forma Geral

```
CHARACTER [*comp[,]]nome[,nome]...
```

comp - especifica o comprimento

Exemplo:

```
CHARACTER*4  C, A(5)
```

8 ESTRUTURA DE CONTROLE

As estruturas de controle permitem aos programadores a especificação do fluxo de controle de um programa em outras palavras, permitem alterar (desviar) o fluxo de execução dos componentes de um programa.

De acordo com a Programação Estruturada, as estruturas de controle compreendem sequência, seleção e iteração (repetição).

As quatro estruturas básicas de controle são:

1. Estrutura seqüencial.
2. Estrutura se-então-senão (“if-then-else”).
3. Estrutura de laço faça-enquanto (“while-loop”).
4. Estrutura de laço repetição (“repeat loop”).

8.1 Programação Estruturada

Programação estruturada é uma técnica de construir um programa de modo organizado, combinando-se comandos de acordo com as estruturas básicas de controle apresentadas na seção anterior.

8.2 Comando IF

Existem três tipos de comando IF, que são

- IF aritmético;
- IF lógico;
- IF bloco;

8.3 Comando IF Aritmético

O comando IF aritmético, também conhecido como IF de três desvios, transfere o controle de fluxo para um dos três comandos numerados, dependendo do valor de uma expressão aritmética contida no IF.

Forma Geral

IF (e) n1, n2, n3

e - é uma expressão aritmética do tipo inteiro, real ou dupla precisão.

n1,n2,n3 - são números de comandos executáveis na mesma unidade de programa.

Se e<0, desvia o controle para n1

Se e=0, desvia o controle para n2

Se e>0, desvia o controle para n3

Exemplo:

```
      ....
      ....
      IF (A-B) 100,100,200
      ....
      ....
100   ..... ! Comando executável qualquer
      ....
      ....
200    .....   !  Comando executável qualquer
```

Este comando transfere o controle para o comando 100 se a variável A for menor ou igual que a variável B ou transfere o controle para o comando 200 se a variável A for maior que B.

8.4 Comando IF lógico

Este comando executa uma função se for satisfeita uma condição.

Forma Geral

IF (<condição>) Comando(de atribuição, ou um desvio incondicional, etc.)

O valor da (<condição>) é sempre .TRUE. ou .FALSE..

O comando só será executado se a (<condição>) for .TRUE. caso contrário executará a próxima linha de comando.

```
IF Bloco
IF (<condição) THEN
        Bloco 1
ENDIF
```

Exemplo:

```
      ....
      ....
      IF (A.AND.B)   RESTO=0.0
70    AUX=X*Y
      ....
```

Neste exemplo, se A e B são ambos .TRUE., o valor da variável RESTO é substituído por 0.0; em qualquer outra situação, o valor de RESTO é inalterado. O comando 70 é executado em qualquer situação.

8.5 Comando IF Bloco

Existem quatro comandos IF Bloco:

- IF THEN
- ELSE
- ELSE IF THEN
- ENDIF

O comando FORTRAN que implementa diretamente a estrutura de seleção SE-ENTÃO-SENÃO é o IF-Bloco.

Forma Geral

```
IF  (<condição>) THEN
|       Bloco 1
ELSE
|       Bloco 2
ENDIF
```

onde:<condição> é sempre expressa através de uma expressão lógica.

Observações:

- Os blocos 1 e 2 podem ser compostos por qualquer combinação de comandos da linguagem (inclusive novos IF-Bloco).
- O comando ENDIF é obrigatório para cada IF utilizado.
- O comando ELSE pode não existir. Neste caso, a estrutura será:

```
IF  (<condição>) THEN
|       Bloco 1
ENDIF
```

Exemplo:

```
SE (A>B) ENTÃO
        A ← B
        B ←  √3
        C ←  B/2
SENÃO
        B←  A
        A←  √A
        C←  A/2
FIM DA CONDIÇÃO
```

EM FORTRAN =>

```
IF  (A.GT.B) THEN
        A=B
        B=SQRT(3)
        C=B/2
ELSE
        B=A
        A=SQRT(A)
        C=A/2
ENDIF
```

Se tivesse mais duas outras condições dentro do ELSE poderia ser feita de duas maneiras:

```
IF (<condição1>) THEN
        Bloco 1
ELSE
        IF (<condição 2>) THEN
            Bloco 2
          IF (<condição 3>) THEN
            Bloco 3
           ENDIF
        ENDIF
ENDIF
```

```
IF (<condição1>) THEN
        Bloco 1
ELSEIF (<condição 2>) THEN
        Bloco 2
ELSE
        Bloco 3
ENDIF
```

8.6 Comando DO WHILE

O comando DO WHILE executa um bloco de comandos repetidas vezes enquanto a condição lógica permanece (.TRUE.).

Formato:

[nome] DO [rótulo[,]] WHILE (expressão lógica)

- nome (opcional): Nome da construção do DO. Se o bloco DO está com um nome, o correspondente END DO deve especificar o mesmo nome.
- Rótulo (opcional): rótulo de um comando END DO ou CONTINUE. Se omitido o rótulo, o *loop* deve finalizar com END DO e não com CONTINUE.
- Expressão lógica: Testa a expressão e retorna valor (.TRUE.) ou (.FALSE.).

Seqüência do comando DO WHILE:

1. A expressão lógica é obtida. Se a expressão é .FALSE., nenhum dos comandos dentro da região do *loop* será executado. Se a expressão for .TRUE., os comandos dentro do *loop* são executados.
2. Quando o último comando dentro da região do DO WHILE é executrado, retorna-se ao comando DO WHILE. A expressão é mais uma vez obtida e o ciclo se repete.

Pode-se interromper o loop do comando DO WHILE com o comando CYCLE ou EXIT.

Exemplo:

```
CHARACTER (1) input
input = ‘ ‘
DO WHILE ((input .NE. ‘n’) .AND. (input .NE. ‘y’))
    WRITE (*, ‘(A)’) ‘ Digite y ou n : ‘
    READ (*, ‘(A)’) input
END DO
```

OBS.: Tem-se que ter cuidado com alguns caracteres, por exemplo o que indicar string. O símbolo correto em uma escala maior para poder diferenciar é este: ‘ e não utilizar este: ´ , pois alguns teclados têm estes dois símbolos.

9 ESTRUTURA DE ITERAÇÃO

9.1 Comando DO

O comando FORTRAN que implementa diretamente uma das formas da estrutura de iteração é o DO.

Forma Geral

```
DO  n   I = m1, m2, m3                    DO   I=m1, m2,m3
        Bloco 1              ou                Bloco 1
n   CONTINUE                              ENDDO
```

onde: n - número do comando que delimita a abrangência do laço (iteração) do DO

i - variável do DO. Do tipo inteiro.

m1 - valor inicial de i;

m2 - valor máximo de i para teste;

m3 - incremento

Observações:

- O incremento m3 pode ser omitido e, neste caso, assume o valor 1, isto é incremento unitário.
- O valor da variável do DO (i) não pode ser alterado por qualquer comando dentro do laço.
- Não é permitido saltar de um ponto do laço do DO para outro dentro dele.
- Pode haver aninhamento de laços de DO (“laços encaixados”), mas os laços internos devem estar totalmente contidos no laço mais externo, isto é, não pode haver cruzamento de laços.

Estrutura VÁLIDA

```
      DO 100 I=1,80        ! ponto de exclamação é válido como comentário
      ....
      ....
        DO  50  J=1,10     ! incremento unitário
        ....
        ....
50      CONTINUE
        DO 100  K=1,20,1   ! incremento unitário
          ....
          ....
100     CONTINUE
```

Estrutura Não VÁLIDA

```
      DO 100 I=1,80        ! ponto de exclamação é válido como comentário
      ....
      ....
        DO  50  J=1,10     ! incremento unitário
        ....
        ....
100     CONTINUE
C         Outra forma de se fazer comentário - C na 1ª. coluna ou outro caractere.
          ....
          ....
50      CONTINUE
```

9.2 Comando GOTO

9.2.1 Desvio Incondicional

A função deste comando é alterar incondicionalmente a ordem seqüencial de execução de procedimentos.

Forma Geral:

GOTO n

onde n é o endereço da instrução para onde será desviado o processamento.

9.2.2 Desvio Condicional

A função deste comando é alterar condicionalmente a ordem de execução de procedimentos.

Forma Geral:

GOTO (n_1,...,n_n) variável

n_1,...,n_n : endereço da instrução para onde será desviado o processamento, dependendo do valor da variável. Se a variável for igual a 1, o processamento será desviado para a instrução de endereço n_1 ; se a variável for igual a n, o processamento será desviado para a instrução de endereço n_n.

Variável : é do tipo inteiro, que define a condição a ser testada. Se o valor da variável estiver fora do intervalo [1,n], o comando GOTO é ignorado.

Segue-se abaixo alguns comandos de controle de fluxo.

9.3 Comando CONTINUE

O comando CONTINUE é um comando executável de controle que não produz nenhum efeito quando executado, porém, transfere o controle para o próximo comando executável.

9.4 Comando PAUSE

O comando executável PAUSE suspende, temporariamente, a execução do programa e exibe uma informação no terminal do operador, para permitir que se tome alguma ação.

9.5 Comando STOP

O comando executável STOP termina a execução do programa, pode exibir uma informação no terminal do operador, e retorna o controle para o sistema operacional.

9.6 Comando END

O comando END indica o final de uma unidade de programa para o compilador.

9.7 Comando CYCLE

O comando CYCLE, deve sempre aparecer dentro de um *loop* DO ou DO WHILE, interrompe o ciclo de interação voltando ao mecanismo de controle do *loop*.

Formato: CYCLE [nome da construção]

Nome da construção: (opcional) corresponde ao rótulo da construção do DO ou DO WHILE. Se o nome não vem expresso indica que o comando CYCLE está referindo ao DO ou DO WHILE mais interno que pertença à construção DO ou DO WHILE.

Exemplo:

```
SAMPLE_LOOP: DO I= 1,5
 PRINT *,I
  IF (I.GT.3) CYCLE SAMPLE_LOOP
PRINT *,I
 END DO SAMPLE_LOOP
PRINT *,'done'
STOP
END
```

Resultado:

```
Sai na tela:
        1
        1
        2
        2
        3
        3
        4
        5
done
```

9.8 Comando EXIT

O comando EXIT, transfere o controle dentro de um *loop*, formado por um do DO ou DO WHILE, para o primeiro comando executável que se segue depois do fim do *loop*.

Exemplo:

```
INTEGER numpontos, pontos
REAL datarray (1000), soma
suma = 0.0
DO pontos =1,1000
    soma = soma + datarray(point)
    IF (datarray(point+1) . EQ. 0.0 ) EXIT
END DO
```

10 ENTRADA E SAÍDA DE DADOS

Os comandos de entrada e saída (E/S) controlam e transferem informações entre unidade de memória principal e um dispositivo externo de entrada/saída (terminais de vídeo, impressoras, unidades de disco, etc.).

10.1 Comando WRITE

O comando básico para saída de dados tem a forma geral:

WRITE (i,n) <lista de variáveis ou expressões aritméticas>

onde: i - assume * ou 0 para a tela do terminal do PC.

n - número de um comando FORMAT, que especifica o formato da impressão e, pode assumir * para que a saída seja sem formato especial, isto é, *formato livre*.

Exemplo:

```
....
      WRITE(*,10)  A, B, I, A+C
10    FORMAT (...)        ! O comando FORMAT será visto mais adiante
....
```

Os valores das variáveis A, B, I e o valor da expressão A+C serão listados na tela, conforme o formato especificado no comando FORMAT de número 10.

10.2 Comando READ

O comando básico para a entrada de dados tem a forma geral:

READ (i,n) (lista de variáveis)

onde: i - pode assumir * ou 0 para o teclado do terminal do PC.

n - número de comando FORMAT, que especifica o formato dos dados de entrada.

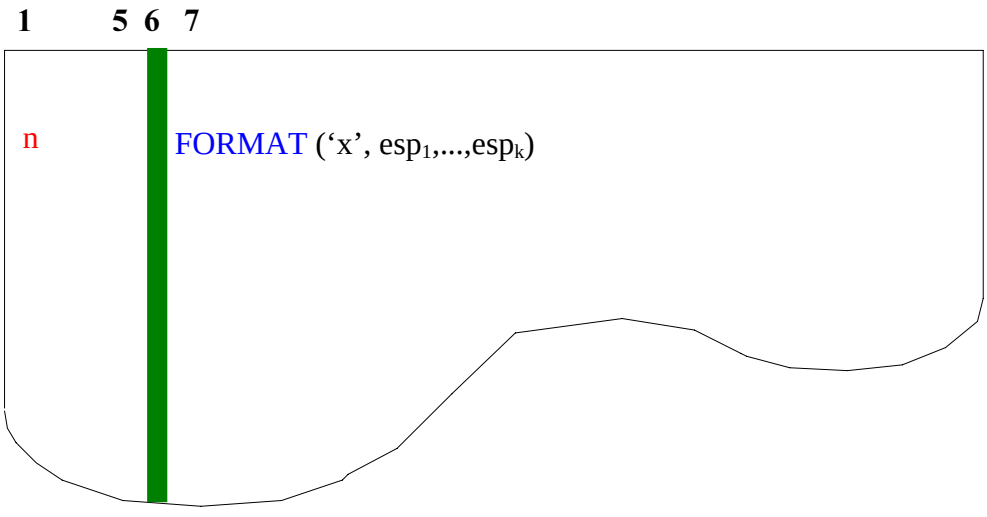
Exemplo:

```
      ....
      READ (*,20)  A,  B,  K
      ....
20    FORMAT (...)      ! O comando FORMAT será visto mais adiante
```

10.3 Comando FORMAT

O comando FORMAT permite especificar a forma em que os dados serão lidos ou impressos pelos comandos de E/S associados; é um comando bastante utilizado, principalmente quando se deseja obter saída de dados mais elaborados.

Forma Geral:



onde: n - número do comando FORMAT;

x - é o caractere de controle do carro da impressora ou do cursor da tela;

esp_i - são as especificações de formato, que serão discutidas a seguir.

Este comando é não-executável e pode aparecer em qualquer lugar dentro do programa fonte; recomenda-se, entretanto, que ele seja inserido logo após o comando de E/S associado.

Com este comando é possível:

- pular linha (/);
- dar espaçamento entre os parâmetros de saída(X);
- escrever textos (‘ ‘);
- especificar a maneira com que uma variável será escrita.

10.4 Especificações de Formato mais usuais

10.4.1 Especificação numérica para inteiro

A especificação numérica para inteiro é utilizada para definir o comprimento do campo associado aos valores inteiros, lidos ou escritos.

Forma Geral

aIw

onde: a - é uma constante inteira e sem sinal que indica o número de vezes que esta especificação I deve ser obedecida, repetitivamente (opcional);

w - é uma constante inteira e sem sinal (não nula) que indica o comprimento do campo do dado (em caracteres).

Exemplos:

Formato mínimo para imprimir os seguintes valores inteiros:

+247 - I4

-33 - I3

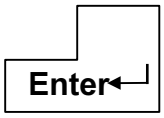
33 - I2

-11561 - I6

Considere o seguinte trecho de um programa:

```
      ...
      INTEGER  VAR1, VAR2, VAR2
      VAR1  = 51
      READ (*,10) VAR2, VAR3
10    FORMAT (2I4)
      WRITE (*,20) VAR1, VAR2, VAR3, VAR1+VAR3
20    FORMAT ('X', I2 , 2I4 ,I4)
      ...
```

Ao executar a linha "READ (*,10)...." , há uma pausa no programa esperando que o usuário forneça dois valores via teclado (*). Suponha que o usuário digite os seguintes números

-128,1024 Ao pressionar a tecla 

O programa vai continuar no fluxo do programa e a próxima linha executável será o comando "WRITE (*,20) ...", o qual escreverá na tela a seguinte informação:

51-12810241075

Percebe-se que não há separação entre os números. Para que os quatro valores saíssem separados, a formatação teria que possuir a característica:

WRITE(*,20) (X , I2, X , 2(X , I4) , X , I4)

e apresentar o seguinte resultado:

51 -128 1024 1075

Observações:

- Se, numa impressão, o campo reservado for insuficiente, aparecerão *** impressos em todo o campo.

  Exemplo: FORMAT: I3
  VALOR INTERNO: -830
  VALOR IMPRESSO: ***

- Na impressão, se o campo reservado for maior que o necessário, o dado fica colocado à direita, com espaços em brancos à esquerda.

Exemplo: FORMAT: I3
VALOR INTERNO: -4
VALOR IMPRESSO: ␣-4

- Na entrada de dados, espaços em brancos são considerados zero. Portanto, se o campo reservado for maior que o número de dígitos dos dados, estes deverão estar à direita no campo.

  Exemplo: FORMAT: I4
  DADO: 23
  ENTRADA : ␣␣23

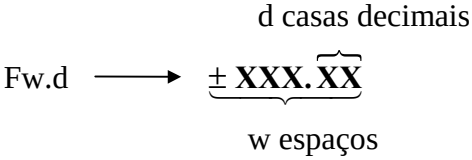
10.4.2 Especificação Numérica para Real Básico (sem expoente)

A especificação numérica para básico é utilizada para definir o comprimento do campo associado aos valores reais básicos, lidos ou escritos. Esta especificação define o número de casas decimais destes valores.

Forma Geral:

aFw.d

onde: a,w - têm o mesmo significado anterior;
d - constante inteira e sem sinal que indica o número de casas decimais.



Exemplos:

Para imprimir completamente os valores reais seguintes, o formato mínimo a ser utilizado é:

126.85 - F6.2

-2.1 - F4.1

-53. - F4.0

10.4.3 Especificação numérica para reais com expoente

A especificação numérica para reais com expoente é utilizada para definir o comprimento do campo e o número de casas decimais associadas aos valores reais com expoente, lidos ou escritos.

Forma Geral:

aEw.d

onde: a,w,d - têm o mesmo significado das especificações anteriores

Exemplo:

Para imprimir completamente os valores reais seguintes, o formato mínimo a ser utilizado é:

1.26E+03 - E8.2

-5.1E-02 - E8.1

Observações:

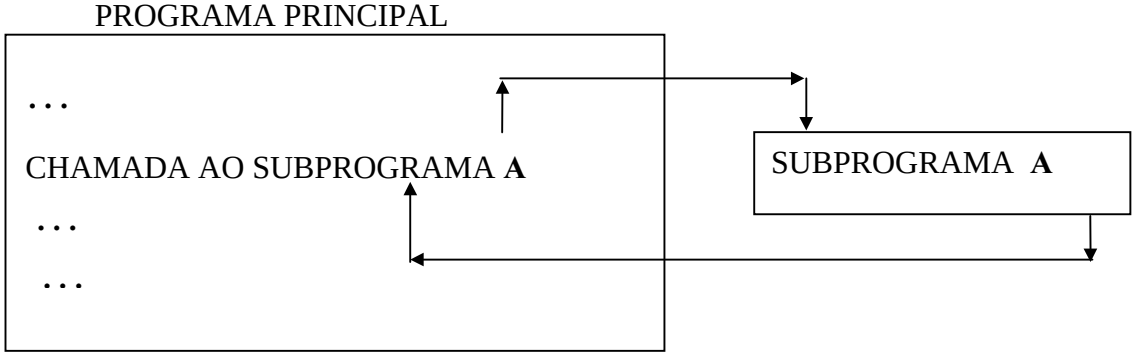
- Na entrada, valem as mesmas convenções do item do formato Fw.d. Na realidade, estas duas especificações têm exatamente o mesmo efeito.
- Na saída, o dígito mais significativo aparece logo à direita do ponto. O número de dígitos significativos depende de d. Quando há espaço no campo, um zero é impresso à esquerda do ponto. No entanto, se estiver mais espaço, descontando o campo do sinal negativo quando for o caso, são impressos brancos. Para garantia deve-se fazer: $w \geq d+7$.

11 SUBPROGRAMAS

Um subprograma é uma unidade de programa independente e simples, que em geral, realiza uma única função e que tem associado um identificador(nome). Sua execução é subordinada ao programa principal ou a uma outra unidade de programa. Uma unidade de programa é uma seqüência de comandos FORTRAN terminada pelo comando END.

Em FORTRAN, há dois tipos básicos de subprogramas: as funções e as sub-rotinas. As características de cada tipo e as diferenças entre eles serão vistas adiante.

Ao chamar uma subrotina ou uma função, o fluxo do programa é alterado como mostra a figura abaixo:



Com a chamada ao subprograma, o controle passa aos comandos do Subprograma A. Após a execução dos comandos, este subprograma retorna ao ponto de chamada. Num programa pode haver várias chamadas a subprogramas diferentes, e um mesmo subprograma pode ser chamado várias vezes.

11.1 Funções

Existem três classes de funções providas pelo FORTRAN:

- Funções Intrínsecas ou de Biblioteca.
- Funções de Comando.
- Subprogramas FUNCTION.

11.1.1 Funções Intrínsecas ou de Biblioteca

São funções providas pela própria linguagem e que podem ser referenciadas em quaisquer expressões em um programa. Deve-se apenas escolher valores adequados para o(s) argumento(s) (ver capítulo 5).

Exemplo: Y=SQRT(var)

11.1.2 Função Instrução

Tem como objetivo criar funções, as quais , através de uma instrução, executam cálculos que se repetem em pontos distintos do programa, fornecendo um único valor como resultado.

Forma Geral

NOME (A1,A2,...,NA) EXPR

onde: NOME é o nome dado a função;

Ai: argumento da função que será utilizado no cálculo da expressão;

EXPR: expressão que define o cálculo executado pela função.

Exemplo:

```
...
REAL*4  GRADI, X
GRADI(X)=X*3.1416/180   ! converte o angulo X de grau para radiano
WRITE(*,*) 'Entre como o valor da ângulo em graus: '
READ(*,*) ANGULO
SN=SN(GRADI(ANGULO))
CS=COS(GRADI(ANGULO))
WRITE(*,*) 'SENO = ', SN
WRITE(*,*) 'CO-SENO = ',CS
END
...
```

11.1.3 Subprograma FUNCTION

Tem como objetivo criar funções, as quais, através de várias instruções, executam cálculos que se repetem em pontos distintos do programa, fornecendo um único valor como resultado.

Forma Geral:

```
TIPO FUNCTION NOME (A1,A2,...,NA)
INSTRUÇÕES
NOME = EXPR
RETURN
END
```

onde:

TIPO: tipo do valor calculado pela função

NOME: nome da função

Ai: parâmetros usados para cálculo da expressão

INSTRUÇÕES: conjunto de instruções que irão realizar a tarefa desejada.

Exemplo:

```
REAL*4  A,B,C,ANG
WRITE (*,*) 'DIGITE O ANGULO'
READ(*,*) A
B = ANG(A)
C = SIN(B)
WRITE (*,*) "SENO = ', C
END
....
....
REAL FUNCTION ANG(X)
REAL*4 X
ANG X*3.1416/180
RETURN
END
```

11.2 Subrotina

CALL: Tem como função acionar uma subrotina definida. Sua forma geral é:

CALL NOMESSUB (ARG1, ARG2,..., ARGN)

onde:

NOMESUB: nome da subrotina;

ARGi: argumento que o programa principal passa para a subrotina, e/ou que a subrotina devolve ao programa principal.

SUBROUTINE: Cria subrotinas que, através de várias instruções, executam procedimentos que se repetem em pontos distintos do programa, fornecendo como resultado um ou vários valores. Sua forma geral é:

```
SUBROUTINE NOMESUB (ARG1, ARG2, ..., ARGN)
[Campo das Declarações]
INSTRUÇÕES [Comandos executáveis]
RETURN
END
```

onde:

NOMESUB: nome da subrotina;

ARGi: argumento que o programa principal passa para a subrotina, e/ou que a subrotina devolve ao programa principal.

Exemplo:

```
      CHARACTER*1 RESP
      INTEGER*2 N,NEL, NNVINC, NMAT
10    WRITE(*,*) 'ENTRE COM NUMERO DE NOS'
      READ (*,*) N
      WRITE(*,*) 'ENTRE COM NUMERO DE ELEMENTOS'
      READ(*,*) NEL
      WRITE(*,*) 'ENTRE COM NUMERO DE NOS VINCULADOS'
      READ(*,*) NNVINC
      WRITE(*,*) 'ENTRE COM NUMERO DE MATERIAIS'
      READ(*,*) NMAT
      CALL CONFDAD (N,NEL,NNVINC,NMAT,RESP)
      IF (RESP.EQ.'N'.OR.RESP.EQ.'n') GOTO 10
      ...
      ...
      SUBROUTINE CONFDAD (N,NEL,NNVINC,NMAT,RESP)
      CHARACTER*1 RESP
      INTEGER*2 N, NEL, NNVINC,NMAT
10     FORMAT (//,1X,'NUMERO DE NOS:',I3,/,1X,'NUMERO DE NOS
        VINC.:',I3,/,1X,'NUMERO DE MATERIAIS DIFERENTES:',
        I3,/,'DADOS CORRETOS? (S/N)')
      READ(*,20) RESP
20    FORMAT (A1)
      RETURN
      END
```

12 ARQUIVOS

Muitas aplicações exigem o armazenamento de grandes quantidades de dados resultados de processamento. A memória principal do computador nem sempre pode suprir estas exigências. Nestas circunstâncias, utilizam-se outros dispositivos de armazenamento (memória secundária), como por exemplo fitas e discos magnéticos.

Manipulação de Arquivos

De maneira geral, na manipulação de arquivos utilizam-se, fundamentalmente, os comandos: READ, WRITE, OPEN, CLOSE, BACKSPACE, REWIND e ENDFILE. Alguns destes comandos serão discutidos a seguir.

12.1 Comando OPEN

O comando OPEN inicializa o arquivo para que as operações de E/S possam ser realizadas sobre ele.

Forma Geral

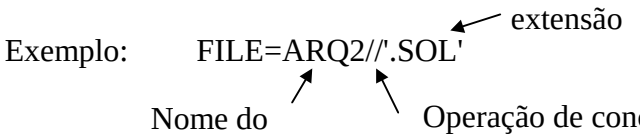
OPEN ([n] [, FILE=<nome>] [, STATUS='<estado>'] [,ACCESS='<acesso>']

[, RECL=<tamanho>] [, FORM='<formato>'])

onde:

- n - número inteiro associado ao arquivo (≥1), Usa-se número de unidade(n) diferente para distinguir um arquivo de outro.
- nome - é o nome do arquivo, expressão do tipo caractere. Geralmente coloca-se extensão para seguir os padrões já existentes dos arquivos. <Nome.ext>. Pode

fazer operação de concatenação neste ponto, acrescentando assim uma extensão a um nome fornecido pelo usuário.



- ‘estado’- o estado de um arquivo pode ser:

  OLD :para arquivos já existentes (default);

  NEW :para arquivos que serão criados;

  UNKOWN : para arquivos com estado desconhecido.

- ‘acesso’ - o tipo de acesso que será feito aos registros do arquivo, podendo ser:

  SEQUENCIAL: para acessar registros do arquivo seqüencialmente (default);

  DIRECT: para acessar os registros do arquivo aleatoriamente, informando o número do registro desejado. Neste caso, é preciso especificar o tamanho do registro.

- tamanho - tamanho em BYTES dos registros (usado somente para arquivos de acesso direto).

- ‘formato’ - formato no qual os dados serão lidos ou gravados, podendo ser:

  FORMATTED : formatado de leitura/gravação (default);

  UNFORMATTED: sem formato para leitura/gravação.

Exemplo:

```
      CHARACTER*12 ARQENT
      WRITE(*,*)’NOME DO ARQUIVO DE ENTRADA:’ ! sai na tela
      READ(*,10) ARQENT   ! variável onde será armazenada o nome do arquivo
10    FORMAT(A12)
      OPEN(1,FILE=NARQ,STATUS=’OLD’)
      ...
      ...
      CLOSE(1) ! fecha o arquivo 1
      END
```

12.2 Comando CLOSE

A função deste comando é informar que o arquivo que havia sido aberto pelo comando OPEN não será mais usado.

Forma Geral

CLOSE (n, STATUS=’estado’)

onde: n - número do arquivo a ser desativado;

‘estado’ - estado no qual o arquivo será desativado

KEEP : manter o arquivo (default)

DELETE : desgravar o arquivo.

12.3 Comando READ - Leitura de Arquivos

READ(n,m)

n - número do arquivo definido no OPEN. Se o nome do arquivo não tiver sido definido o mesmo será solicitado durante a execução.

m - indica o número da instrução onde está especificado o formato de leitura dos dados, ou * para formato padrão. Não deve ser declarado para arquivos com FORM=UNFORMATTED.

Exemplo:

Número do arquivo

```
10    READ(2,15,ERR=16) I, X1, Y1    ! lê no arquivo as variáveis I, X1, Y1
15    FORMAT(I6,2F15.0)
      COR(I,1)=X1    ! armazena X1 em COR(I,1)
      COR(I,2)=Y1
      GOTO 5     ! vai para o comando 5
16    WRITE(*,*)'  Voce entrou com dados errados '
      STOP   !  Para o programa
```

Só entra aqui se houver algum erro nos dados do arquivo

Neste exemplo, a leitura de dados de arquivo(READ(2,15,ERR=16) I, X1, Y1) foi acrescida de “ERR=”, que serve para desviar do controle quando ocorre um erro na operação de entrada e também de saída, como por exemplo quando se tem formatos errados.

12.4 Comando WRITE - Gravação de Arquivos

WRITE (n,m)

n - número do arquivo definido no OPEN. Se o nome do arquivo não tiver sido definido, o mesmo deverá ser solicitado durante a execução.

m - indica o número da instrução onde está especificado o formato de gravação dos dados, ou * para formato padrão. Não deve ser declarado para arquivos com FORM=UNFORMATTED.

Exemplo:

```
      WRITE(3,*)' '   ! escreve uma linha em branco no arquivo 3
      WRITE(3,*)'     NO       X          Y'
          DO 360 I=1,NN     ! NN número de nós
         WRITE(3,361) I, COR(I,1),COR(I,2)   ! grava nó e coordenadas
361       FORMAT('     ',I3,2('    ',F12.3))  ! formato
360   CONTINUE
```

13 CONJUNTOS E VARIÁVEIS INDEXADAS

13.1 Conjuntos na Linguagem FORTRAN - Vetores

Em FORTRAN, um vetor é um conjunto unidimensional de variáveis. Um nome é associado ao vetor como um todo e cada elemento deste vetor é referenciado por um índice. Este vetor poderá ter mais de uma dimensão passando a ser um conjunto multidimensional de variáveis.

13.1.1 Declaração DIMENSION para Vetores

Forma Geral:

DIMENSION a1([e11]:e12), a2([e21]: e22), a3([e11:]e12,[e21:]e22)

onde:

ai - são os nomes dos vetores;

ei1 - são os limites inferiores do índice (opcional);

ei2 - são os limites superiores do índice.

ei1, ei2 podem ser valores inteiros positivos, nulos ou negativos.

Exemplo:

```
REAL*8   VETOR, CORD
CHARACTER*20  NOMES
DIMENSION  VETOR(1:100), NOMES(1:40), CORD(1:100,1:2)
```

Esta declaração informa ao compilador que VETOR e NOMES são vetores e que o VETOR é constituído de 100 elementos do tipo real , NOME de 40 elementos do tipo caractere e CORD é uma matriz de 100 por 2 do tipo real.

A declaração anterior é equivalente a:

```
REAL*8   VETOR, CORD
CHARACTER*20  NOMES
DIMENSION  VETOR(100), NOMES(40), CORD(100,2)
```

ou ainda:

```
REAL*8   VETOR(100), CORD(100,2)
CHARACTER*20  NOMES(40)
```

14 ALOCAÇÃO DINÂMICA

Em muitos casos, inicialmente não se sabe qual será o tamanho dos vetores e matrizes, pois estes dependem dos dados de entrada. Antes era definido um tamanho para os vetores e matrizes, podendo estes serem sub ou super dimensionados.

Com a alocação dinâmica este problema não mais acontece, pois ao fornecer certas variáveis ao computador, este automaticamente cria o tamanho das matrizes e vetores necessária para cada caso particular, utilizando assim, a memória do computador de maneira mais adequada.

Formato

NOMEMAT1 [ALLOCATABLE] (:), NOMEMAT2 [ALLOCATABLE] (:,:)

Como já foi visto, a definição implícita é usada quando se tem uma matriz cujas dimensões são dependentes de dados de entrada. Deste modo, ao invés de se manter dimensões fixas (ocupando mais memória), deixa-se estas dimensões em função de parâmetros que variam a cada exemplo.

Por exemplo, suponha que NOMEMAT1 armazena as coordenadas, x e y dos nós do elemento, sendo NOMEMAT1 = CORD (NN-número de nós),

tem-se que definir CORD como:

```
INTEGER  NN
REAL*8  CORD[ALLOCATABLE](:,:)
```

quando o programa ler o número de nós totais(NN) da estrutura, defini-se o tamanho desta matriz como:

```
ALLOCATE (CORD(NN))
```

Os parâmetros de transferência entre subrotinas podem conter vetores ou matrizes que foram previamente definidas com alocação dinâmica. Vamos supor que se queira transferir a variável do exemplo anterior CORD(NN) para uma subrotina GERACAO. O modelo a seguir ilustra melhor esta possível transferência:

```
INTEGER  NN
REAL*8  CORD[ALLOCATABLE](:,:)
...
ALLOCATE (CORD(NN)) ! após ler a variável NN pode-se fazer a alocação.
...
CALL GERACAO(NN,CORD)   ! chamando a subrotina GERACAO
...
STOP  ! fim do programa principal
END   ! indicação do fim do programa para o compilador FORTRAN
...
SUBROUTINE  GERACAO (NN,CORD)
INTEGER  NN
REAL*8  CORD[ALLOCATABLE](:,:)
...
RETURN
END
```

Pode-se perceber que a definição do tamanho do vetor CORD dentro da subrotina coincide com o tamanho de quando foi feita a alocação dinâmica. Isto deve acontecer sempre.

O tipo da variável também terá que ser o mesmo, isto é, se a variável foi alocada como real de dupla precisão (REAL*8), as variáveis definidas dentro da subrotina, terão que ser também declaradas como dupla precisão.

Quando o tamanho de uma matriz ou vetor foi obtido através da declaração ALLOCATE, esta matriz ou vetor podem ser desalocado com a declaração DEALLOCATE.

Desalocar uma matriz ou vetor que não foi alocado através de declaração ALLOCATE causará erro do tipo “run-time”.

Exemplo:

```
REAL*8    matriz[ALLOCATABLE] (:,:)
INTEGER  ERRO,n
.
.
ALLOCATE (matriz(n,n))
.
.
DEALLOCATE (matriz, STAT=ERRO)
IF (ERRO.NE.0)  WRITE(*,*)’Erro na desalocação’
.
.
ALLOCATE (matriz(2*n,2*n))    ! alocando novamente
```

O comando STAT, opcional, utilizado no exemplo anterior retornará zero se a desalocação foi efetuada com sucesso. Se ERRO for igual a um inteiro qualquer a desalocação não foi efetuada com sucesso.

A desalocação é interessante, pois libera memória durante a execução do programa, ou quando a dimensão de uma determinada matriz muda durante a execução do programa, deve-se primeiro desalocar e em seguida alocar novamente.

15 APLICAÇÕES

15.1 Multiplicação de Matrizes

Sejam $\underset{\sim}{\mathbf{A}} = \left[\mathbf{a}_{uv}\right]_{mxn}$ e $\underset{\sim}{\mathbf{B}} = \left[\mathbf{b}_{rs}\right]_{nxp}$

Define-se $\underset{\sim}{\mathbf{A}}.\underset{\sim}{\mathbf{B}} = \left[\mathbf{c}_{ij}\right]_{mxp}$

onde:

$$\mathbf{c}_{ij} = \sum_{k=1}^{n} \mathbf{a}_{ik}.\mathbf{b}_{jv} = \mathbf{a}_{u1}.\mathbf{b}_{1v} + \ldots + \mathbf{a}_{un}.\mathbf{b}_{nv}$$

Rotina de Cálculo:

```
DO  10  I = 1,m
      DO  10  J = 1, p
           C( I , J ) = 0
           DO   10  K = 1, n
                C(I, J) = C(I, J ) + A(I, K) * B(K , J)
          ENDDO
      ENDDO
ENDDO
```

Segue-se um modelo de um programa, onde a leitura de dados é feita via arquivo e as variáveis são alocadas dinamicamente. Com estes dados armazenados em matrizes, é feita uma multiplicação da duas matrizes e o resultado desta operação é gravado em arquivo de dados.

```
      PROGRAM MULTI    ! declaração opcional
C     definindo as variaveis
      IMPLICIT CHARACTER*30 (A,B)
      INTEGER  n,m,p
      REAL MATA[ALLOCATABLE](:,:),MATB[ALLOCATABLE](:,:),
     +       MATC[ALLOCATABLE](:,:)

      WRITE(*,1)
1     FORMAT(//,'     NOME DO ARQUIVO DE DADOS :',\)
      READ(*,3) ARQ
3     FORMAT (A30)
      WRITE(*,6)
6     FORMAT(//,'     NOME DO ARQUIVO DE SAIDA(sem estensao):',\)
      READ(*,3) ARQ2
      OPEN (1, FILE=ARQ , STATUS='OLD')      ! arquivo de leitura
C     arquivo onde sera gravado
      OPEN (2, FILE = ARQ2//'.SAI',STATUS='UNKNOWN')
5     READ(1,4) m,n,p

4     FORMAT (3I4)

C     alocando as variaveis

      ALLOCATE(MATA(m,n))
      ALLOCATE(MATB(n,p))
      ALLOCATE(MATC(m,p))

C     lendo a matriz A

      DO I=1,m
       DO J=1,n
        IF (J.EQ.n) THEN
          READ (1,11) MATA(I,J)
        ELSE
          READ (1,10) MATA(I,J)
        ENDIF
      ENDDO
      ENDDO
```

```
C       lendo a matriz B
        DO I=1,n
          DO J=1,p
            IF (J.EQ.p) THEN
               READ (1,11) MATB(I,J)
            ELSE
               READ (1,10) MATB(I,J)
           ENDIF
         ENDDO
        ENDDO
10      FORMAT (F10.2,\)
11      FORMAT (F10.2)

C         multiplicando as matrizes A.B e armazenando em C

        DO  I=1,m
          DO   J=1,p
            MATC(I,J)=0.
            DO   K=1,n
                   MATC(I,J)= MATC(I,J)+MATA(I,K)*MATB(K,J)
           ENDDO
          ENDDO
        ENDDO
C         gravando em arquivo

        WRITE(2,*)'MULTIPLICANDO MATRIZES'
        WRITE(2,*)'DIMENSOES DAS MATRIZES:'
        WRITE(2,30) m,n,n,p
30      FORMAT('A (',I4,','I4,')','      B (',I4,','I4,')')

        WRITE(2,*)'MATRIZ A'
        DO I=1,m
          WRITE(2,*) (MATA(I,J),J=1,n)
        ENDDO
        WRITE(2,*)' ' ! pula uma linha

        WRITE(2,*)'MATRIZ B'
        DO I=1,n
          WRITE(2,*) (MATB(I,J),J=1,p)
        ENDDO
        WRITE(2,*)' ' ! pula uma linha

        WRITE(2,*)'MATRIZ C = A * B'
        DO I=1,m
          WRITE(2,*) (MATC(I,J),J=1,p)
        ENDDO
```

```
C       fechando os arquivos
        CLOSE(1)
        CLOSE(2)
        WRITE(*,31)
31      FORMAT (//////)
        WRITE (*,*) '          OPERACAO COM SUCESSO'
        WRITE (*,*) 'pressione qualquer tecla para continuar!'
        WRITE(*,31)
        PAUSE

C       fim do programa
        STOP
        END
```

Exemplo do arquivo de Entrada

```
2,3,2
1., 8., 3.
4.,5.,2 .
1.,4.
7.,3.
5.,1.
```

Arquivo digitado em qualquer editor de texto
(Bloco de Notas, Edit, Ambiente Pascal, etc.)

Exemplo do arquivo gerado pelo programa, cujo nome será [*NOME*.SAI]:

```
MULTIPLICANDO MATRIZES

 DIMENSOES DAS MATRIZES:

A (  2,  3)     B (  3,  2)
 MATRIZ A
      1.000000       8.000000       3.000000
      4.000000       5.000000       2.000000



 MATRIZ B
      1.000000       4.000000
      7.000000       3.000000
      5.000000       1.000000

 MATRIZ C = A * B
     72.000000      31.000000
     49.000000      33.000000
```

15.2 Resolução de Sistema de Equações Lineares

Para uma dada matriz $\underset{\sim}{\mathbf{A}}$ e um vetor $\underset{\sim}{\mathbf{b}}$, de modo que:

$$\underset{\sim}{\mathbf{A}}.\underset{\sim}{\mathbf{x}} = \underset{\sim}{\mathbf{b}}$$

o trabalho consiste em encontrar o vetor $\underset{\sim}{\mathbf{x}}$.

Existem várias formas de resolver um sistema de equações lineares. Uma delas é **Eliminação Gauss** com **retro - substituição**, cujo processo é mais rápido para o computador do que calcular a inversa.

Exemplo:

Seja o sistema:

$$\begin{aligned} 2x_1 + 4x_2 + x_3 &= 9 \\ 5x_1 + x_2 + 6x_3 &= 4 \\ x_1 + 3x_2 + 6x_3 &= 12 \end{aligned} \quad \Rightarrow \quad \underset{\underset{\sim}{\mathbf{A}}}{\begin{bmatrix} 2 & 4 & 1 \\ 5 & 1 & 6 \\ 1 & 3 & 6 \end{bmatrix}}.\underset{\underset{\sim}{\mathbf{x}}}{\begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix}} = \underset{\underset{\sim}{\mathbf{b}}}{\begin{Bmatrix} 9 \\ 4 \\ 12 \end{Bmatrix}}$$

Define-se:

$$R = \frac{A(2,1)}{A(1,1)} = \frac{5}{2}$$

$$b(2) = b(2) - R.b(1) = 4 - \frac{5}{2}.9 = \frac{37}{2}$$

$$A(2,1) = A(2,1) - R.A(1,1) = 5 - \frac{5}{2}.2 = 0$$

$$A(2,2) = A(2,2) - R.A(1,2) = 1 - \frac{5}{2}.4 = -9$$

$$A(2,3) = A(2,3) - R.A(1,3) = 2 - \frac{5}{2}.1 = -\frac{1}{2}$$

Resultando:

$$\begin{bmatrix} 2 & 4 & 1 \\ 0 & -9 & -1/2 \\ 1 & 3 & 6 \end{bmatrix}.\begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} 9 \\ 37/2 \\ 12 \end{Bmatrix}$$

$$\mathbf{R} = \frac{\mathbf{A(3,1)}}{\mathbf{A(1,1)}} = \frac{\mathbf{1}}{\mathbf{2}}$$

$$\mathbf{b(3) = b(3) - R.b(1) = 12 - \frac{1}{2}.9 = \frac{15}{2}}$$

$$\mathbf{A(3,1) = A(3,1) - R.A(1,1) = 1 - \frac{1}{2}.2 = 0}$$

$$\mathbf{A(3,2) = A(3,2) - R.A(1,2) = 3 - \frac{1}{2}.4 = 1}$$

$$\mathbf{A(3,3) = A(3,3) - R.A(1,3) = 6 - \frac{1}{2}.1 = \frac{11}{2}}$$

$$\begin{bmatrix} 2 & 4 & 1 \\ 0 & -9 & -1/2 \\ 0 & 1 & 11/2 \end{bmatrix}.\begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} 9 \\ 37/2 \\ 15/2 \end{Bmatrix}$$

$$\mathbf{R} = \frac{\mathbf{A(3,2)}}{\mathbf{A(2,2)}} = \frac{\mathbf{1}}{\mathbf{-9}} = -\frac{\mathbf{1}}{\mathbf{9}}$$

$$\mathbf{b(3) = b(3) - R.b(2) = \frac{15}{2} - \left(-\frac{1}{9}\right).\left(\frac{-37}{2}\right) = \frac{49}{9}}$$

$$\mathbf{A(3,1) = A(3,1) - R.A(2,1) = 0 - \left(-\frac{1}{9}\right).0 = 0}$$

$$\mathbf{A(3,2) = A(3,2) - R.A(2,2) = 1 - \left(-\frac{1}{9}\right).(-9) = 0}$$

$$\mathbf{A(3,3) = A(3,3) - R.A(2,3) = \frac{11}{2} - \left(\frac{-1}{9}\right).\left(\frac{-1}{2}\right) = \frac{49}{9}}$$

$$\begin{bmatrix} 2 & 4 & 1 \\ 0 & -9 & -1/2 \\ 0 & 0 & 49/9 \end{bmatrix}.\begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} 9 \\ -37/2 \\ 49/9 \end{Bmatrix}$$

Retro-substituição

Forma Geral:

$$x_i = \frac{1}{a'_{ii}} . \left[b'_i - \sum_{j=i+1}^{N} a'_{ij} . x_j \right]$$

logo

$$x_3 = \frac{b(3)}{A(3,3)} = \frac{49/9}{49/9} = 1$$

$$A(2,2).x_2 = b(2) - A(2,3).x_3 = -\frac{37}{2} - \left(-\frac{1}{2}\right).1 = -18$$

$$x_2 = \frac{-18}{A(2,2)} = \frac{-18}{-9} = 2$$

$$A(1,2).x_1 + A(1,2).x_2 + A(1,3).x_3 = b_3$$

$$2.x_1 + 4.2 + 1.1 = 9 \qquad \Rightarrow x_1 = 0$$

```
      PROGRAM SOLUCAO

      IMPLICIT CHARACTER*30 (A,B)
      INTEGER N
      REAL*8 MATA[ALLOCATABLE](:,:),B[ALLOCATABLE](:),
     +    U[ALLOCATABLE](:)

      WRITE(*,1)
1     FORMAT(//,'    NOME DO ARQUIVO DE DADOS :',\)
      READ(*,3) ARQ
3     FORMAT (A30)
      WRITE(*,6)
6     FORMAT(//,'    NOME DO ARQUIVO DE SAIDA(sem estensao):',\)
      READ(*,3) ARQ2
      OPEN (1, FILE=ARQ , STATUS='OLD')     ! arquivo de leitura
C       arquivo onde sera gravado
      OPEN (2, FILE = ARQ2//'.SOL',STATUS='UNKNOWN')
5     READ(1,4) N
4     FORMAT (3I4)

C     alocando as matrizes

      ALLOCATE (MATA(N,N+1))
      ALLOCATE (B(N))
      ALLOCATE (U(N))

C       lendo a matriz MATA

      DO I=1,N
       DO J=1,N
        IF (J.EQ.N) THEN
          READ (1,11) MATA(I,J)
        ELSE
          READ (1,10) MATA(I,J)
        ENDIF
      ENDDO
      ENDDO

C       lendo a matriz B

       READ (1,*) ( B(I), I=1,N)

10     FORMAT (F10.2,\)
11     FORMAT (F10.2)
```

```
C
C          para resolucao do sistema o vetor b deve entrar
C             como sendo a ultima coluna de MATA, isto e N+1
      DO I=1,N
       MATA(I,N+1)=B(I)
      ENDDO
C
C       gravando em arquivo
C
      WRITE(2,*)'PROGRAMA PARA RESOLUCAO DE SISTEMA'
      WRITE(2,*)'DIMENSOES DAS MATRIZES:'
30    FORMAT('A (',I4,',I4,')')
      WRITE(2,*)'MATRIZ A'
      DO I=1,N
       WRITE(2,*) (MATA(I,J),J=1,N)
      ENDDO
      WRITE(2,*)' '    ! pula uma linha
      WRITE(2,*)'VETOR B'
      DO I=1,N
       WRITE(2,11) B(I)
      ENDDO
      WRITE(2,*)' '   ! pula  linha

C
C       chama subrotina para resolucao de sistema
C
       CALL SOLVER (MATA,U,N)
C
C          fim da resolucao do sistema
C
       WRITE(2,*)'RESOLUCAO DO SISTEMA'

       DO J=1,N
        WRITE(2,40) J,U(J)
40      FORMAT('X',I1,' : ',F15.6)
       ENDDO
       WRITE(2,*)' ' ! pula uma linha

C          fechando os arquivos

      CLOSE(1)
      CLOSE(2)
      WRITE(*,31)
31    FORMAT (//////)
```

```
      WRITE (*,*) '          OPERACAO COM SUCESSO'
      WRITE (*,*) ' pressione qualquer tecla para continuar!'
      WRITE(*,31)

C          fim do programa

      STOP
      END

C
C       SUBROTINA - RESOLUCAO DO SISTEMA DE EQUACOES
C
      SUBROUTINE SOLVER (R,U,N)
      INTEGER N
      REAL*8 R(N,N+1), U(N), SOMAT
     DO 360 I=1,N
      IF (I.GT.1) THEN
        DO 330 K=I,N
         SOMAT=0
          DO 325 J=1,(I-1)
                SOMAT=SOMAT+R(K,(I-J))*R((I-J),I)
325        CONTINUE
          R(K,I)=R(K,I)-SOMAT
330     CONTINUE
      ENDIF
      DO 350 K=(I+1),(N+1)
       SOMAT=0
       IF(I.GT.1) THEN
        DO 340 J=1,(I-1)
         SOMAT=SOMAT+R(I,(I-J))*R((I-J),K)
340     CONTINUE
       ENDIF
       R(I,K)=(R(I,K)-SOMAT)/R(I,I)
350   CONTINUE
360   CONTINUE
      U(N)=-R(N,(N+1))
      DO 380, I=(N-1),1,-1
      U(I)=0
      SOMAT=0
      DO 370, J=(I+1),N
        SOMAT = SOMAT+R(I,J)*U(J)
370   CONTINUE
      U(I)=-R(I,(N+1))-SOMAT
```

```
380     CONTINUE
        DO 390, I=N,1,-1
          U(I)=-U(I)
390     CONTINUE
        RETURN
        END
```

Modelo do arquivo de entrada de dados:

```
3
2.,4.,1.
5.,1.,2.
1.,3.,6.
9.,4.,12.
```

Arquivo gerado pelo programa, após obtenção do resultado da resolução do sistema de equações:

```
PROGRAMA PARA RESOLUCAO DE SISTEMA

 MATRIZ A
      2.000000000000000      4.000000000000000      1.000000000000000
      5.000000000000000      1.000000000000000      2.000000000000000
      1.000000000000000      3.000000000000000      6.000000000000000

 VETOR B
     9.00
     4.00
    12.00

 RESOLUCAO DO SISTEMA
X1 :        .000000
X2 :       2.000000
X3 :       1.000000
```

16 MANUSEANDO O FORTRAN PowerStation

O FORTRAN PowerStation 4.0 oferece muitos recursos, se comparado com as versões anteriores.

O *Microsoft Developer Studio* é um ambiente de desenvolvimento integrado para aplicações do FORTRAN que permite acesso direto a um editor de código e oferece ferramentas para a construção de interfaces.

Neste capítulo não será possível abordar todos os recursos oferecido pelo FORTRAN versão 4.0 Portanto para aqueles que tiverem maior interesse em se aprofundarem aconselha-se que consulte o ***User's Guide***.

A melhor maneira de aprender o processo de desenvolvimento de um programa com *FORTRAN PowerStation* é a construção de um aplicativo, como exemplo.

O manuseio será dividido basicamente em quatro partes:

1. Iniciando o FORTRAN
2. Criando um novo projeto
3. Rodando o programa
4. Depurando o programa

16.1 Iniciando o FORTRAN

O acesso ao FORTRAN PowerStation 4.0 se dá com um *click* duplo no ícone  *Microsoft Developer Studio*.

É permitido aumentar ou diminuir a tela, assim como nos programas *for windows* e usar o FORTRAN para ver vários documentos ao mesmo tempo, sendo que cada documento aparece em cada *MDI Multi-Document Interface*.

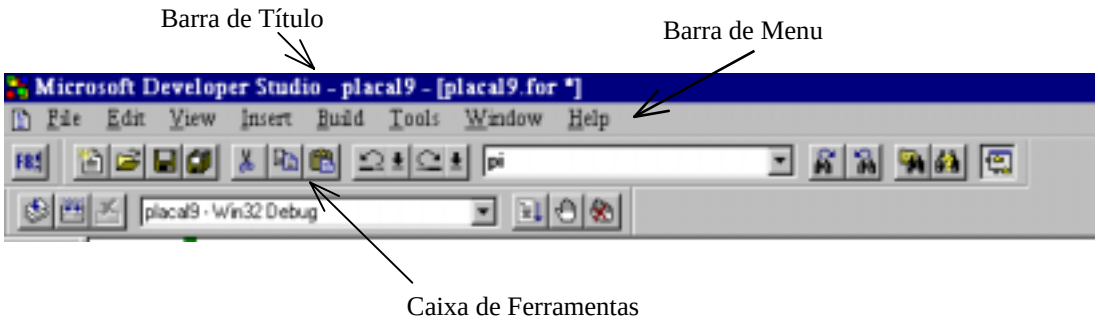
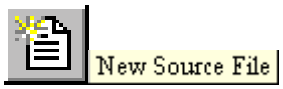


FIGURA 16.1 - Barra de Ferramentas.

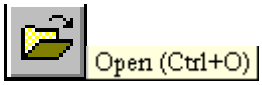
16.1.1 Barra de Ferramentas

A Barra de Ferramentas é composta de Barra de título, barra de Menus e de caixa de ferramentas (FIGURA 16.1), onde:

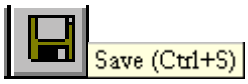
- A barra de título apresenta o título do projeto corrente;
- A barra de Menus fornece acesso aos principais comandos e opções do ambiente FORTRAN PowerStation 4.0, conforme será mostrado mais adiante.
- A caixa de ferramentas é composta de botões, os quais fornecem atalhos para diversas tarefas, como:



Abre um novo arquivo



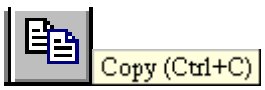
Abre um arquivo existente - tecla de atalho (Ctrl+O)



Permite salvar o arquivo corrente - tecla de atalho (Ctrl+S)

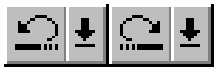
Salva todos os arquivos

Apaga texto selecionado - Tecla de atalho (Ctrl+X)



Copia texto selecionado para área de transferência
tecla de atalho(Ctrl+C)

Cola o texto existente na área de transferência
tecla de atalho (Ctrl+V)

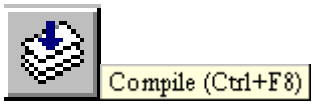


Desfaz ou Refaz ações

Localiza palavras no texto

Este botão localiza **pi** na região do texto acima do cursor

Este botão localiza **pi** na região do texto abaixo do cursor

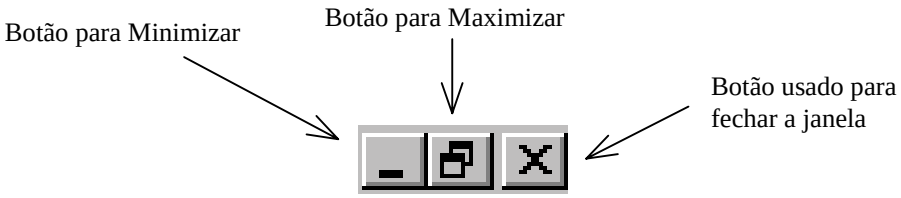


Compila o programa ativo - tecla de atalho (Ctrl+F8)

Constrói o executável do programa ativo - tecla de atalho (Shift+F8)

Para a execução do comando Build

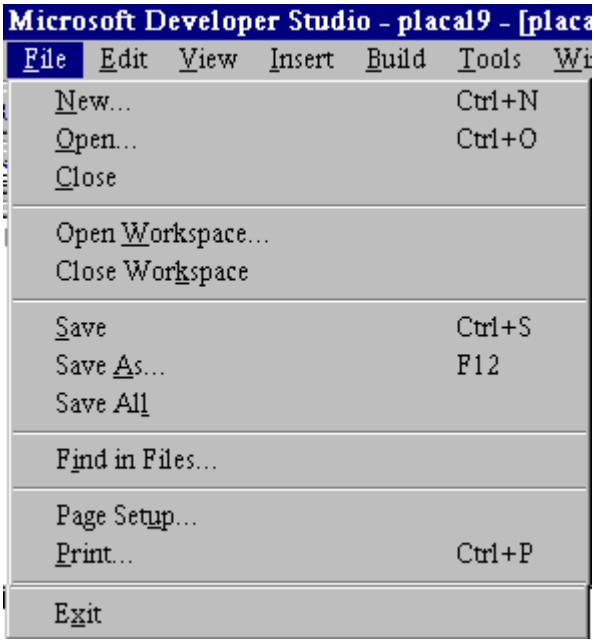
Os três botões abaixo são também padrões do ***Windows95.***



16.1.2 Barra de Menu

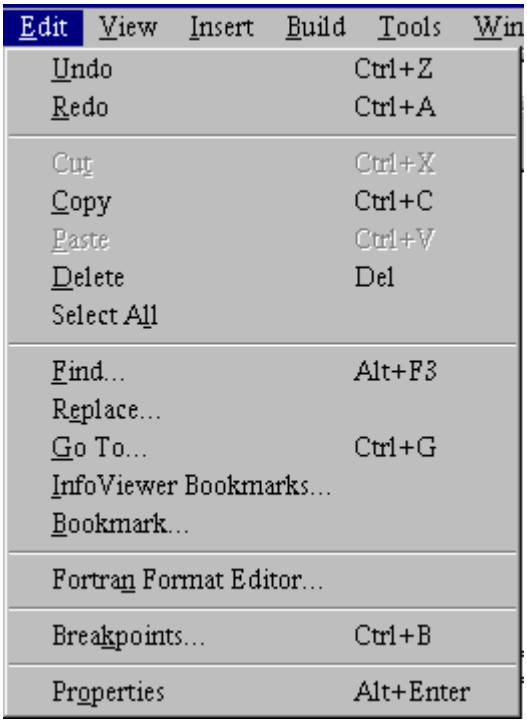
16.1.2.1 Menu File

Você utiliza o menu File principal para trabalhar com os arquivos que formam seu projeto. Esse menu inclui comandos para salvar, carregar e imprimir arquivos.



Novo arquivo
Abre um arquivo existente
Fecha apenas o arquivo corrente
Abre e fecha projetos
Salva apenas o arquivo corrente
Salvar o arquivo
Salva todos os arquivos
Configura página
Imprime arquivo
Sai do ambiente FORTRAN

16.1.2.2 Menu Edit



Desfaz uma ação

Refaz uma ação

Elimina texto selecionado (vai para área de transferência)

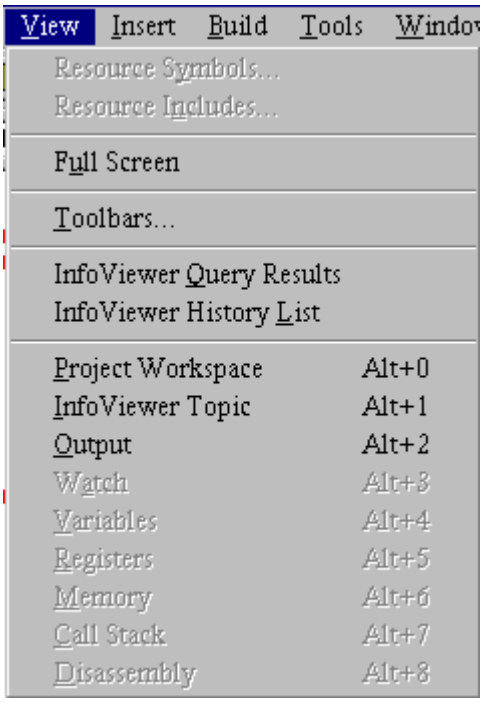
Copia para área de transferencia

Cola texto na área de transferência

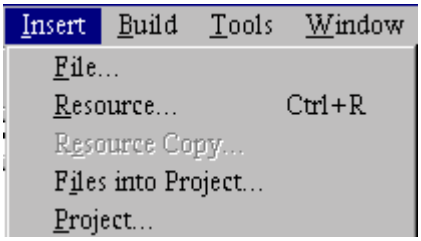
Elimina texto selecionado

Localiza texto no programa

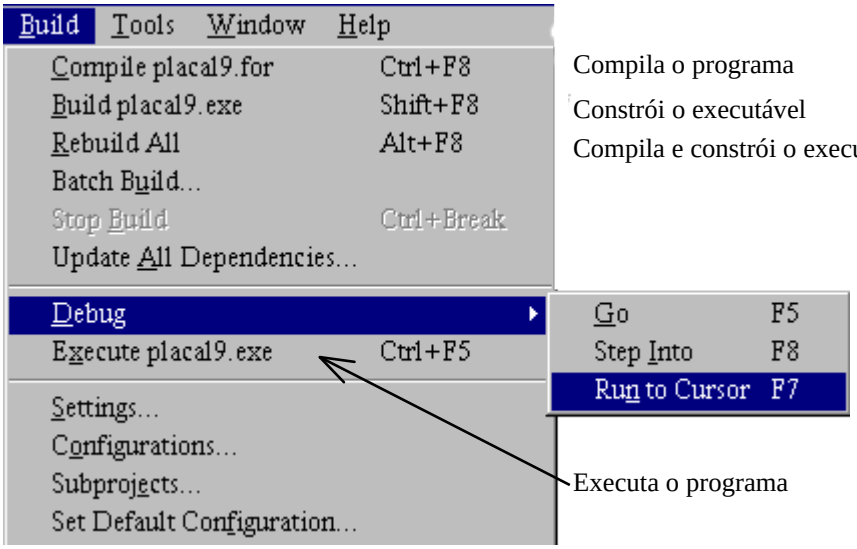
16.1.2.3 Menu View



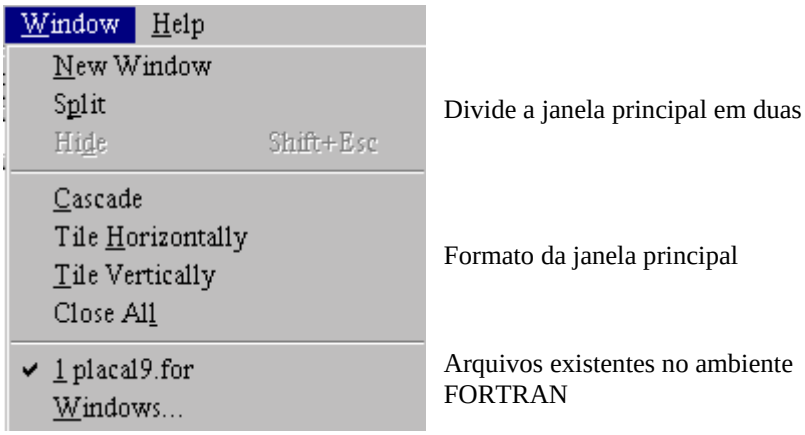
16.1.2.4 Menu Insert



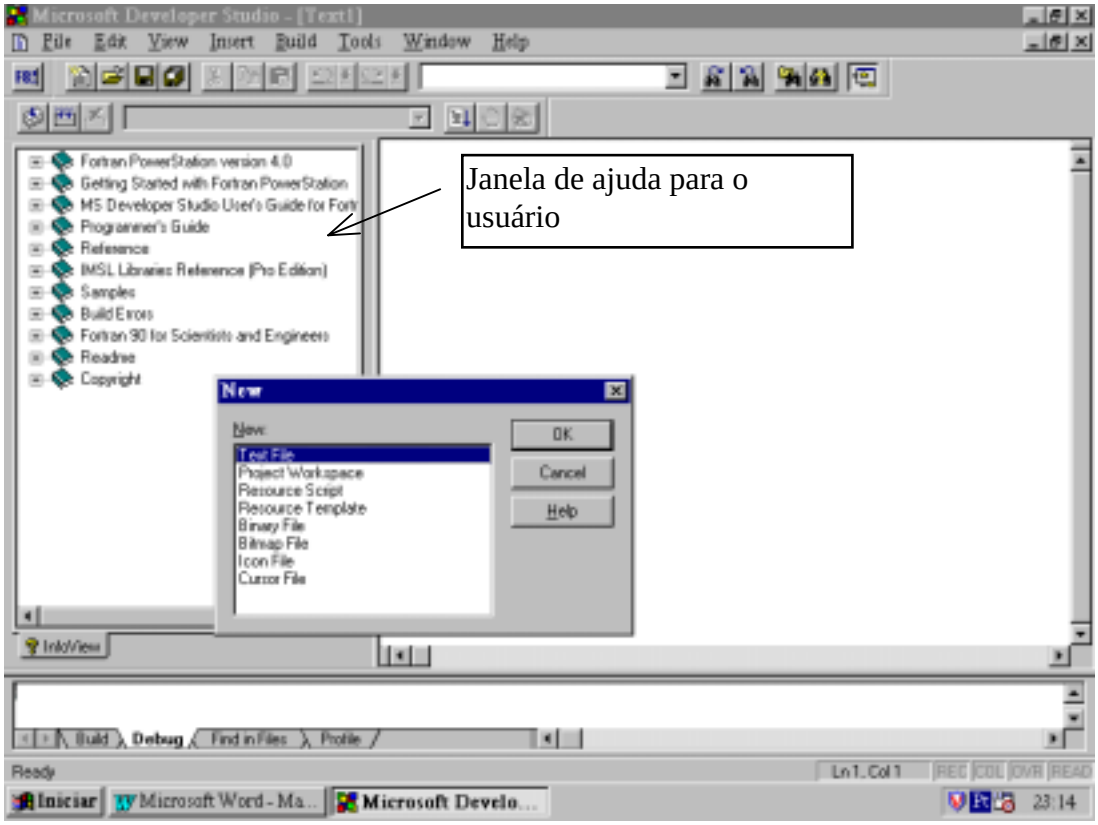
16.1.2.5 Menu Build



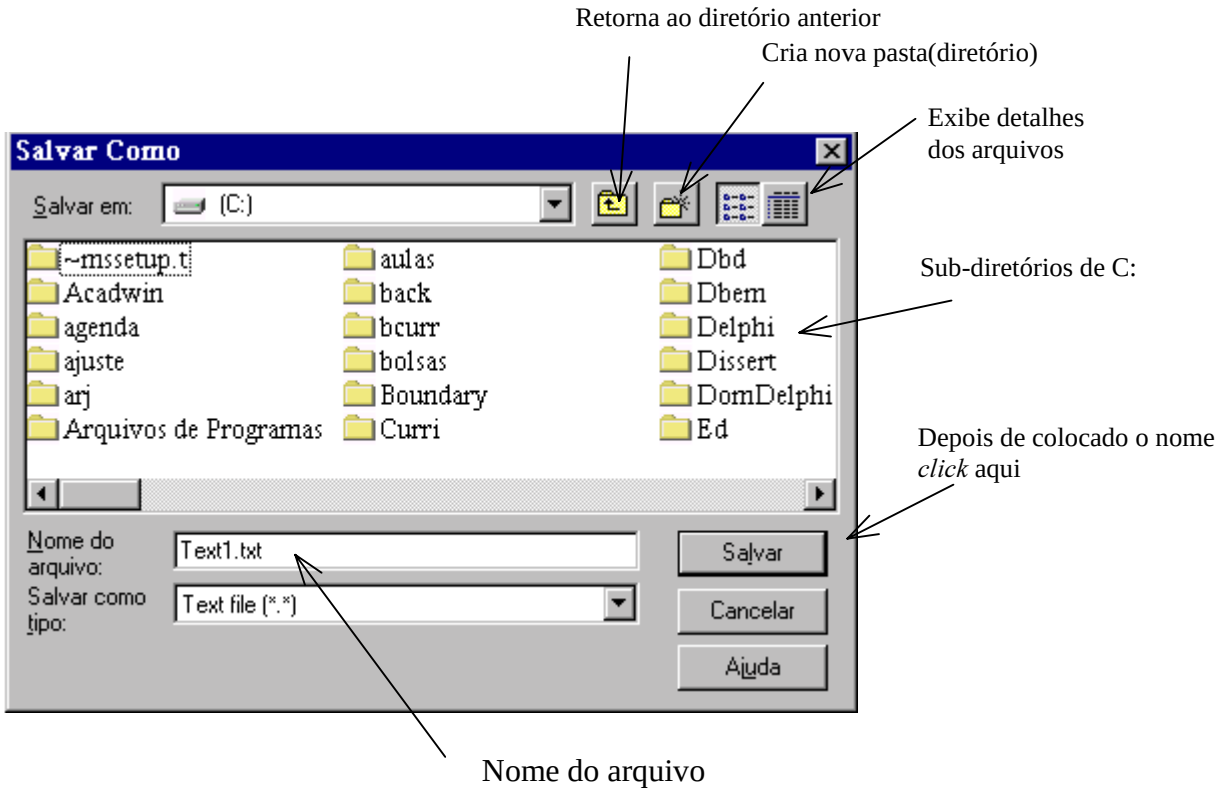
16.1.2.6 Menu Window



16.2 Criando um novo projeto (Editando, Salvando, Abrindo Arquivos)



Para começar a digitar um programa, deve-se ir ao Menu File e ao SubMenu New. Automaticamente aparecerá uma janela para que o usuário selecione o tipo de texto. Aconselha-se que selecione o Text File e em seguida salve o documento. Para isso, com o auxílio do *mouse* dá-se um *click* no Menu File e com o *mouse* arrasta-se até o SubMenu Save. Em seguida, aperta-se o botão esquerdo do *mouse* e aparecerá a janela:



Caso o usuário esteja salvando um arquivo que depois será executado pelo FORTRAN, deve-se salvar com [NOME].**FOR,** que em seguida o texto receberá todas as características de um programa FORTRAN. Por isso, as posições devem ser respeitadas conforme visto no capítulo 1.

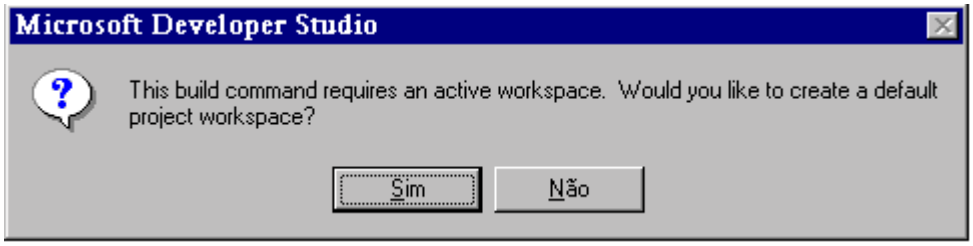
Caso o arquivo tenha sido salvo uma vez, não mais aparecerá esta janela. O arquivo será salvo automaticamente com o nome e diretório corrente. Só aparecerá esta janela novamente, se o usuário escolher o SubMenu Save As do Menu File.

16.3 Rodando o programa

Para execução do programa, o FORTRAN compila o que foi digitado pelo usuário, isto é, transforma em linguagem de máquina o texto digitado. Com isso, poderá ocorrer erros, como: símbolos, formatações, posição dos comandos nas colunas errados, etc.

Após a compilação do programa, deve-se construir o executável para então processá-lo e tornar possível a sua execução sem a utilização do aplicativo FORTRAN.

Para compilar um programa, pode-se utilizar da Caixa de Ferramentas ou do Menu Build e SubMenu Compile. Se foi a primeira vez que tentou compilar o programa e que ainda não tenha sido criado o nome do projeto, aparecerá a seguinte janela:



Informando que ainda não foi criado o nome do Projeto. Para programas simples deve-se escolher a opção do Default, cujo nome será o mesmo do programa principal, isto é, dá-se um *click* em: 

.

Ao terminar o programa, aconselha-se que antes de sair do ambiente FORTRAN, se feche o *Workspace* através do menu File, pois sempre que se inicia o *Microsoft Developer Studio,* ele carrega automaticamente o projeto ativo antes de sair pela última vez. Se você fechou o projeto antes de sair não será carregado nenhum projeto ao iniciar novamente o FORTRAN. Isto evita que algumas pessoas menos informadas carregue outro programa no projeto que foi automaticamente carregado pelo programa ao iniciar. Ao compilar este projeto dará erro de compilação, pois existem dois programas sem nenhuma ligação no mesmo *Workspace*.

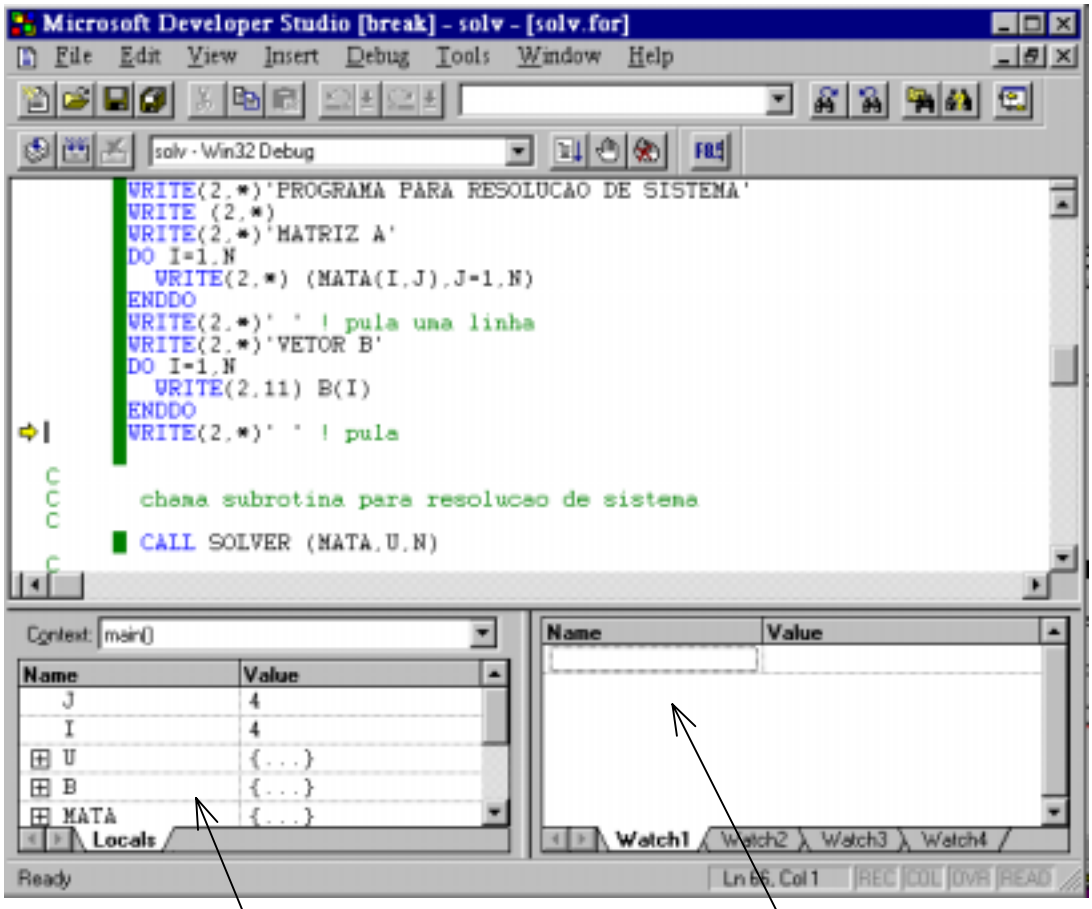
16.4 Depurando o programa

Caso o programa já tenha passado pelo compilador do FORTRAN e não foi acusa nenhum erro, pode haver a necessidade de executar o programa passo a passo,

para identificar algum erro de lógica do programa, pois este tipo de erro o compilador não identifica.

Tome como exemplo o programa de resolução de equações do capítulo 15. Suponha que o programa SOLUCAO já tenha tido digitado e compilado com sucesso e que o resultado da resolução do sistema não tenha sido fornecido os valores corretos, e que seja preciso verificar os valores das matrizes e de algumas constantes do programa antes de fornecer os valores finais. Para isso, coloca-se o cursor na posição onde se deseja que o programa execute até determinado posição e segue os seguintes passos: Menu **Build** , Submenu **Debug** e **Run to Cursor**,ou simplesmente coloca-se o cursor na posição onde se deseja que o programa execute e pressione a tecla F7.

Tem-se então as seguinte características:



Valores automaticamente fornecidos pelo FORTRAN

Pode-se solicitar valores das variáveis

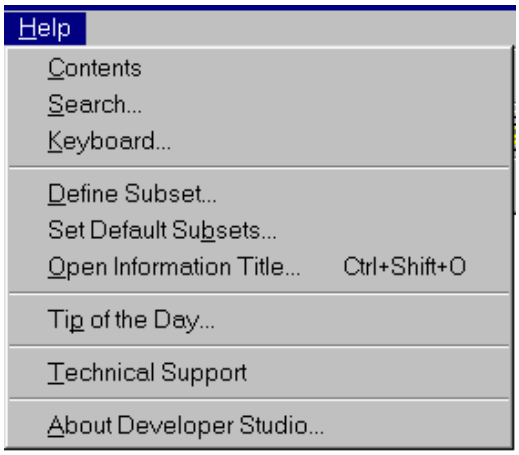
A seta amarela indica a posição do programa, isto é, indica a próxima linha que será executada. A partir daí, pode-se pedir valores das variáveis através da janela abaixo da janela principal do ambiente FORTRAN.

As variáveis que aparecem com a cruz ⊞ são vetores ou matrizes. Ao dar um *click* na cruz, os valores dos coeficientes do vetor correspondente aparecem automaticamente.

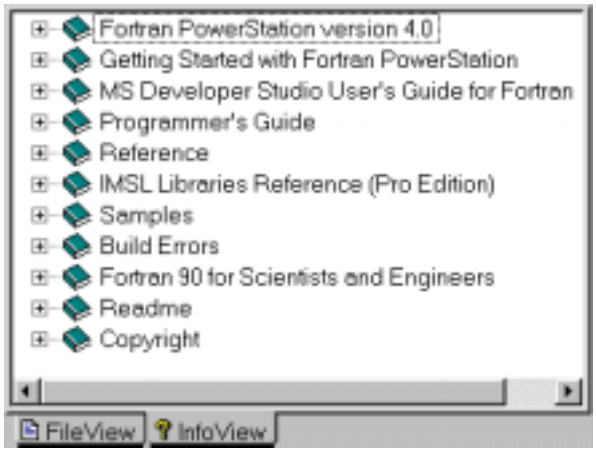
Através da tecla **F8**, o programa é executado passo a passo, alterando automaticamente os valores das variáveis. A tecla **F7** executa um passo maior, isto é, da posição da seta amarela até a posição do cursor. Este procedimento é útil quando necessita-se passar rapidamente por um "*loop"* grande.

16.5 Ajuda

O ambiente *Microsoft Developer Studio* possui um Help muito bom para qualquer dúvida que possa ter sobre a programação, tipos de erros de compilação e quanto a erros de *Link*. O menu Help está mostrado na figura abaixo:

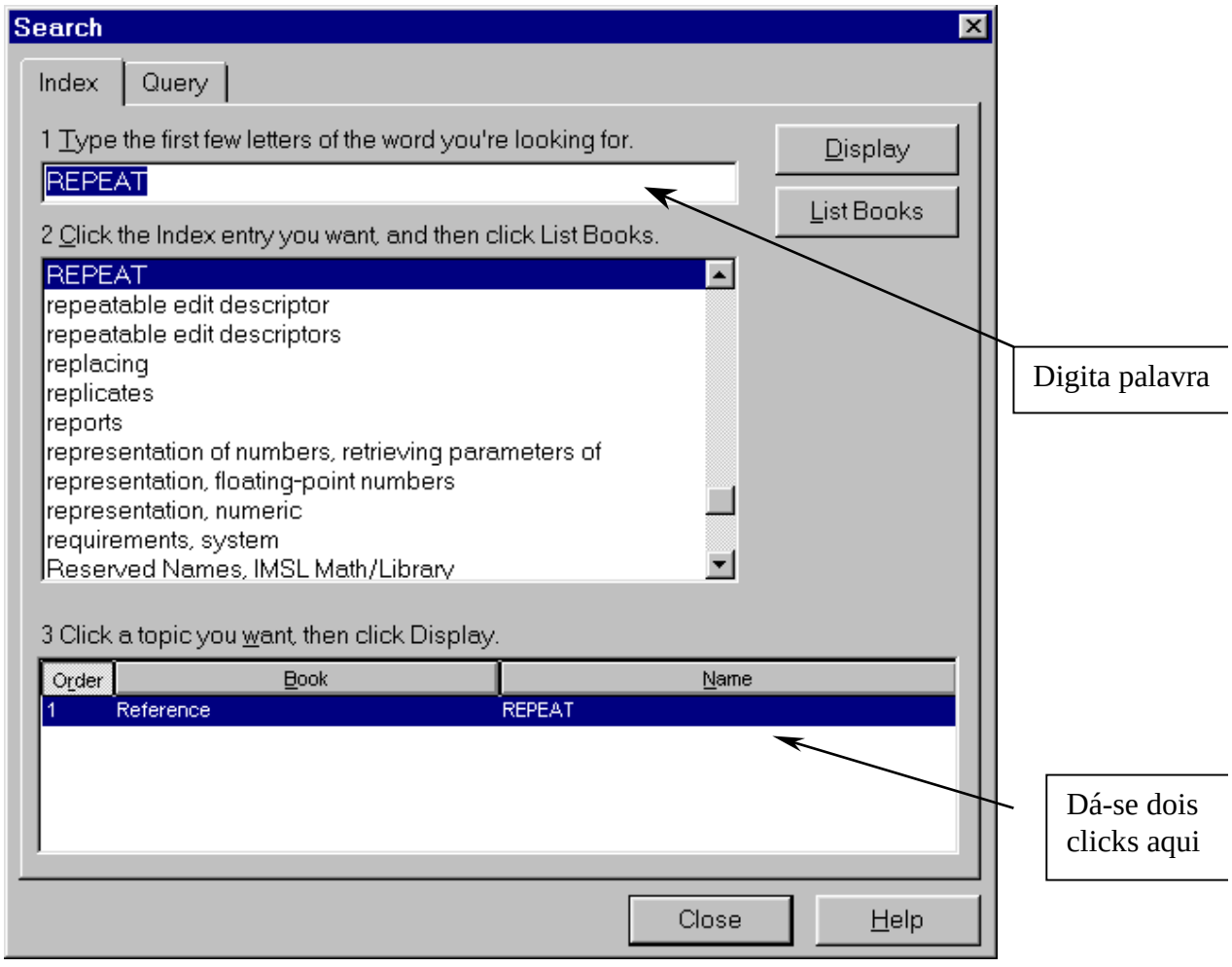


No submenu *Contents*, contém todo Help fornecido pelo *Microsoft Developer Studio*, que tem como temas principais:



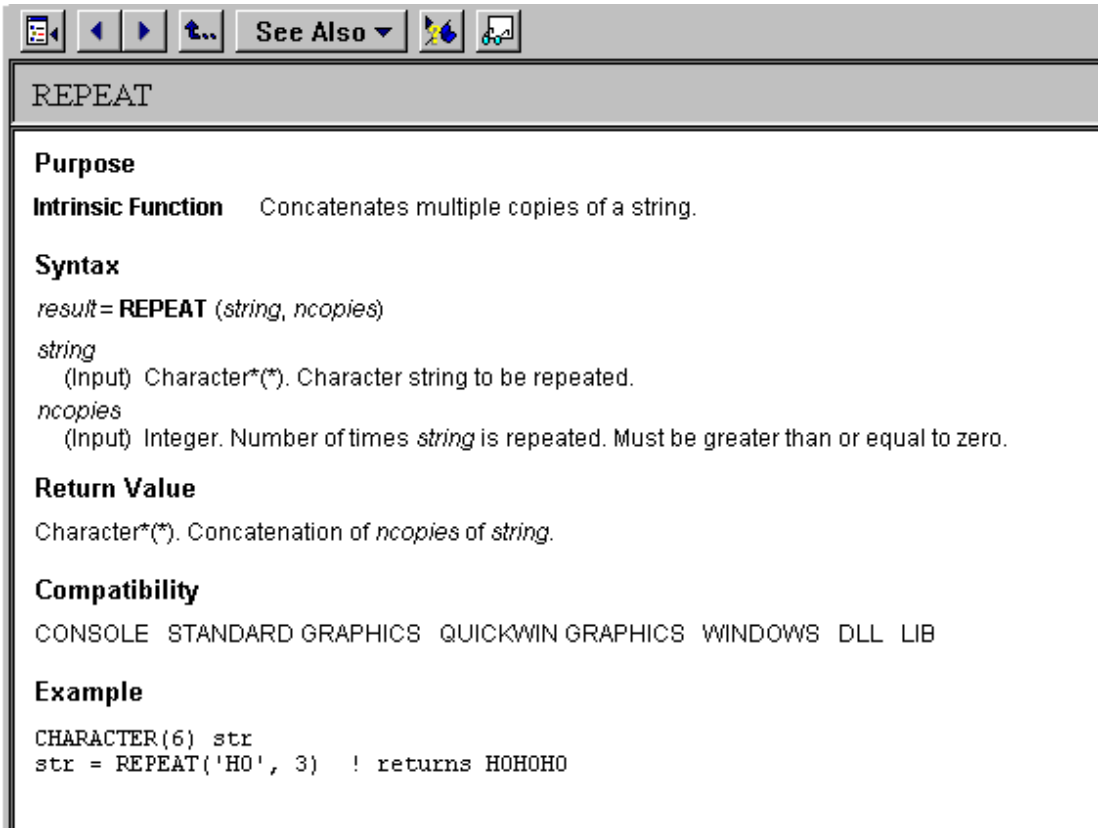
Aconselha-se o leitor percorrer cada um desses itens, para saber o conteúdo e verificar as possibilidades fornecidas pelo *PowerStation*.

Ao dá-se um click no submenu Search, a seguinte janela surgirá:



Nesta janela pode-se procurar informações diretamente, bastando para isso fornecer a palavra que se deseja (em inglês), neste caso foi REPEAT surgindo assim, apenas uma referência sobre esta palavra.

Após dá-se dois clicks na faixa azul, como indicado na figura anterior, surgirá o texto contendo as informações necessárias sobre o palavra REPEAT:



BIBLIOGRAFIA

HEHL, M. E., (1986). ***Linguagem de Programação Estruturada: FORTRAN 77***, McGraw Hill, São Paulo.

CEREADA, R.L.D.; MALDONADO,J.C., (1987). ***Introdução ao FORTRAN 77 para microcomputadores***, McGraw-Hill, São Paulo.

PORCIÚNCULA,N.M.;GONÇALVES, J.A., (1988). Noções Básicas. Apostila de auto-treinamento fornecido pelo CPD da USP de São Carlos.

MANUAIS MICROSOFT- FORTRAN PowerStation ***(User's Guide, Language Guide, Error Messages***). Versão 4.0.

SMITH, I.M. (1995). Programming in FORTRAN 90 – A first Course for Engineers and Scientists. John Wiley & Sons, England.

DRIEMEIER,L. (1996). Curso de FORTRAN oferecido em 1996 no Dept. de Eng. De Estruturas -EESC - USP.