Universidade de São Paulo
Instituto Astronômico e Geofísico
Departamento de Ciências Atmosféricas

Curso Básico de Programação em Linguagem FORTRAN-77

Por:
Edmilson Dias de Freitas
Departamento de Ciências Atmosféricas do IAG-USP

São Paulo
Março de 2004

SUMÁRIO

1. Introdução.

Este material tem como principal objetivo apresentar alguns aspectos básicos de programação em linguagem FORTRAN. A idéia do curso surgiu pela grande utilização da linguagem FORTRAN nas várias áreas de estudo do Departamento de Ciências Atmosféricas. Além da utilização para construção de programas simples durante a execução das disciplinas, acredito que o conhecimento adquirido durante o curso será de grande valia para qualquer trabalho futuro na área de meteorologia. O conteúdo deste curso foi baseado em alguns livros sobre a linguagem e na experiência adquirida durante minha permanência no IAG. Além dos aspectos básicos da linguagem FORTRAN, foi incluído um tópico dedicado ao tratamento de dados através da utilização conjunta do FORTRAN e do software GrADS (the Grid Analysis and Display System) altamente utilizado no departamento. Apesar de ser uma versão com algumas correções e inclusões sobre a primeira, escrita em 2001, é certo que ainda existam alguns aspectos que devam ser melhorados e até mesmo alguns erros. Sendo assim, sugestões e críticas serão sempre bem vindas e podem ser enviadas para o endereço efreitas@model.iag.usp.br .

1.1. Um breve histórico.

A linguagem FORTRAN foi a primeira linguagem de programação de alto nível a ser proposta (surgiu em 1956). Foi sugerida visando a resolução de problemas da área científica, através do uso de computadores. Seu nome é uma composição de FORmula TRANslation. É uma das linguagens mais difundidas no meio técnico-científico, tendo sido ao longo do tempo aprimorada, constituindo as diversas versões disponíveis. Uma das mais recentes é o FORTRAN-95. Neste curso serão apresentados aspectos da versão FORTRAN-77, mas que são válidos para as versões mais recentes, uma vez que estas, são apenas melhorias da versão FORTRAN-77.

Os computadores atuais são incrivelmente rápidos e podem executar conjuntos de instruções extremamente complexos, porém eles são apenas máquinas. O que todo computador pode realmente fazer é seguir ordens muito simples, as quais foram cuidadosamente consideradas e refletidas por um programador e escritas em uma linguagem de programação, como o FORTRAN.

Uma das primeiras coisas que se deve aprender, é como resolver problemas com o auxílio do computador, isto é, como montar, logicamente, as instruções para obter sua

tarefa realizada pelo computador. Especificamente, deve-se aprender a projetar e escrever um ***algoritmo***, que é um procedimento que o computador pode realizar.

1.2. 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. Um algoritmo correto deve possuir três qualidades:

1ª) Cada passo no algoritmo deve ser uma instrução que possa ser realizada;

2ª) A ordem dos passos deve ser precisamente determinada;

3ª) O algoritmo deve ter fim.

Vamos analisar este procedimento através de 2 exemplos

1º) Exemplo - Procedimento para contar:

- Passo 1: Faça N igual a zero.
- Passo 2: Some 1 a N.
- Passo 3: Volte ao Passo 2.

Este procedimento não é um algoritmo, pois ele não satisfaz a condição 3 de um algoritmo correto.

2º) Exemplo - Procedimento para contar até 100:

- Passo 1: Faça N igual a zero.
- Passo 2: Some 1 a N.
- Passo 3: Se N é menor que 100, volte ao Passo 2. Caso contrário, pare.

Este procedimento é um algoritmo, pois ele satisfaz as três condições de um algoritmo correto.

Durante o desenvolvimento do curso, constantemente será necessária a construção de algoritmos. Este é um procedimento indispensável antes da construção de qualquer programa.

2. Conceitos Introdutórios sobre a Linguagem FORTRAN.

Neste item, serão apresentados alguns fundamentos de programação particularizados para o contexto da linguagem FORTRAN.

2.1. Caracteres válidos na Linguagem FORTRAN.

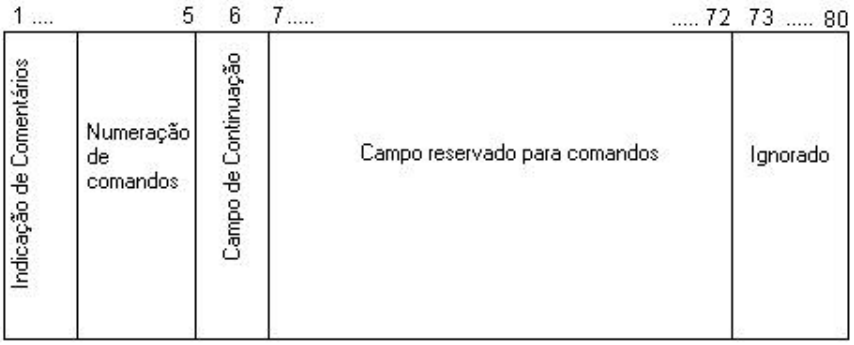
Os caracteres válidos são:

- ***Caracteres Alfabéticos*** : todas as letras maiúsculas e minúsculas do alfabeto latino (A até Z). Embora existam alguns tipos de plataformas que aceitem a alternância entre maiúsculas e minúsculas, algumas não o fazem. Por exemplo: é definida em algum ponto do programa a variável ***var***. Na maioria das plataformas pode-se fazer referência à esta variável utilizando-se de letras maiúsculas ou mesmo uma mistura entre elas, tais como ***Var***, ***VAR***, etc., sem que haja problemas de identificação da variável, mas essa não é uma regra geral. Sendo assim, recomenda-se o uso de letras maiúsculas.
- ***Caracteres numéricos*** : todos os dígitos (0 até 9).
- ***Caracteres especiais***: alguns exemplos são:

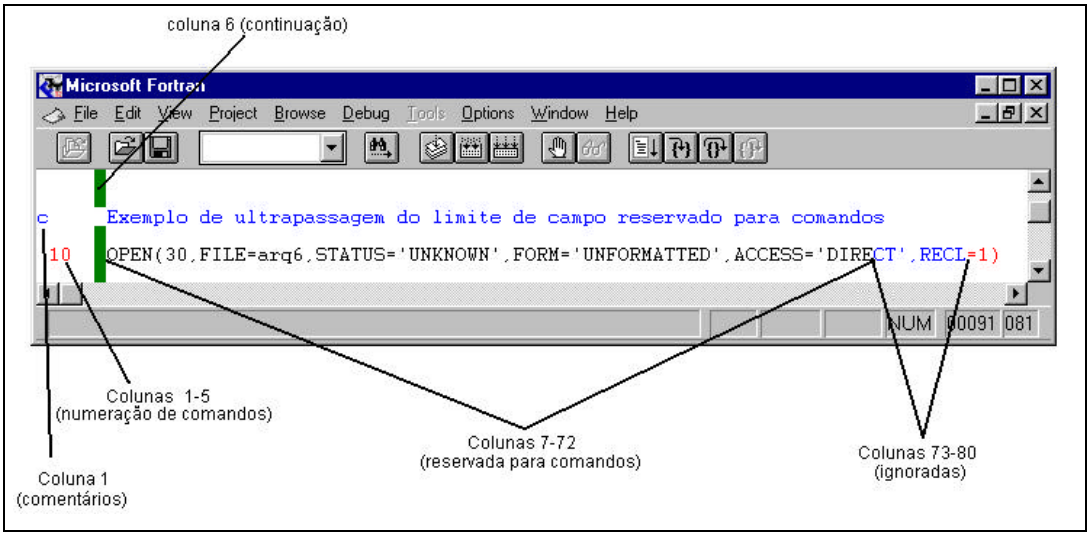
<branco> ou	)
. (ponto)	(
, (vírgula)	‘ (apóstrofo)
+	$
-	:
*	“(aspas)
/	&
=	?

2.2. Folha de Codificação FORTRAN.

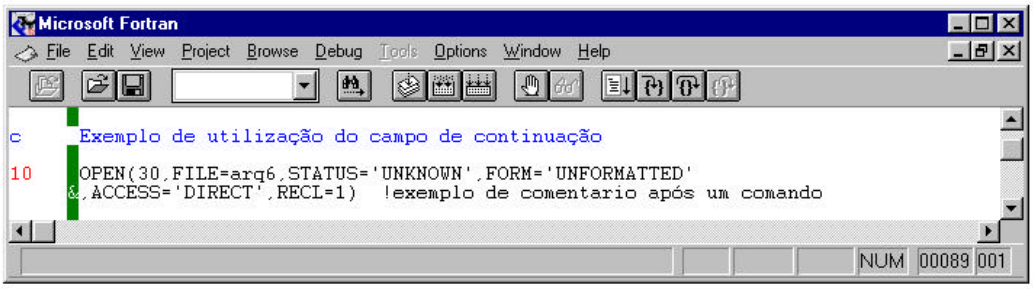
Num programa FORTRAN-77, os comandos devem ser escritos em campos delimitados e fixos. Para facilitar o processo de codificação de programas, existem formulários especiais. Estes formulários são organizados em linhas de 80 colunas, sendo que cada coluna pode conter apenas um caractere e cada linha corresponde a, no máximo, um comando da linguagem. Note-se que há comandos que podem ocupar mais que uma linha. O esquema abaixo é uma indicação da utilidade de cada campo.



Em alguns editores existe uma indicação dos campos, através da utilização de cores, como é o caso do FORTRAN Visual Workbench v 1.0 para windows. A figura abaixo mostra um exemplo de comando em linguagem FORTRAN que ultrapassa o limite permitido.



Na coluna 1, C ou * indicam linha de comentário. Comentários são ignorados pelo compilador FORTRAN, mas são essenciais a um programa bem documentado. Linhas de comentários podem ser inseridas em qualquer parte do programa-fonte. Uma outra maneira utilizada para inserir um comentário é através da utilização do caractere "!" após um comando. No caso ilustrado acima, haveria um erro de compilação. A maneira de corrigir esse erro é através da utilização do campo de continuação (coluna 6). Para isso, pode ser utilizado qualquer caractere (diferente de <branco>ou zero). Pode haver no máximo 19 linhas de continuação. A figura abaixo ilustra a utilização de comentário após um comando e do campo de continuação.



Os números de comandos podem ser quaisquer (inteiros positivos) consistindo em 1 a 5 dígitos e não precisam estar em ordem. Estes números devem ser colocados no campo delimitado pelas colunas 1 e 5. Os números dos comandos não afetam a sua ordem de execução. Brancos e zeros à esquerda são ignorados. Nem todos os comandos precisam ser numerados.

As colunas 73 a 80 são ignoradas pelo compilador e podem ser utilizadas, por exemplo, para informações de identificação dos comandos e do programa.

2.3. Constantes.

Uma constante é uma quantidade fixa, invariável; um valor que não muda no decorrer dos cálculos relativos ao programa.

Existem três classes de constantes:

- ***Numéricas***: aquelas que tratam com números inteiros ou reais;
- ***Lógicas***: aquelas que tratam com valores lógicos – Verdadeiro e Falso.
- ***Cadeias de Caracteres***: aquelas que tratam de seqüências de caracteres admitidos pela linguagem.

Por hora, serão apresentadas apenas constantes numéricas. Os outros tipos serão vistos no decorrer do curso.

Constantes numéricas inteiras: representam números inteiros escritos sem o ponto decimal. Os valores máximo e mínimo para este tipo de constante dependem exclusivamente do sistema de computação utilizado.

Para valores positivos, o sinal pode ser omitido: Ex:

-2	+2	2
25	0	-1

Constantes numéricas reais: representam os números reais. São escritas necessariamente com um ponto decimal ou com um expoente ou com ambos. Da mesma

forma que para as constantes inteiras, os valores máximo e mínimo deste tipo de constante dependem do sistema de computação utilizado.

Algumas representações possíveis são:

± n.m	
± n.	n = parte inteira
± .m	M = parte decimal
± n.mE ± S	S = expoente na base 10 (com um ou dois dígitos)

2.4. Variáveis

Variável é uma entidade que armazena constantes e é conhecida no programa por um nome. Na realidade, é um nome representando uma área de memória. Pode-se distinguir duas classes de variáveis: as variáveis simples, que num dado instante, armazenam um único valor (ex: contadores), e as variáveis compostas, que podem armazenar um conjunto de valores do mesmo tipo, que podem ser acessados individualmente (ex: vetores, matrizes).

Para a utilização de variáveis é necessário declarar qual o tipo de variável utilizada. Declarações possíveis são:

- Declaração de variáveis inteiras:

Ex: INTEGER IANO, IMES, PDIA....

- Declaração de variáveis reais:

Ex: REAL ICHUVA, CONCENTRACAO, ...

A declaração explicita do tipo de variáveis é uma prática importante de programação por influir diretamente nos aspectos de legibilidade, documentação e manutenção dos programas. O FORTRAN não impõe esta prática pois admite uma definição implícita de tipo de variáveis, baseada unicamente no primeiro caractere do nome da variável considerada: se for I, J, K, L, M ou N, a variável será considerada do tipo inteiro, caso contrário será real. A declaração explícita de tipo de variável, embora não seja obrigatória, prevalece sobre a convenção implícita, ou seja, a definição implícita do tipo de variável só é considerada se for omitida a declaração explícita. Nos exemplos citados acima, as variáveis PDIA e ICHUVA seriam, pela definição implícita, real e inteiro,

respectivamente. Entretanto, a declaração explícita prevalece, ficando PDIA como inteiro e ICHUVA como real.

2.5. Expressões aritméticas.

As expressões aritméticas são formadas utilizando operandos numéricos combinados com operadores aritméticos. Estes operadores são:

- + (adição)
- - (subtração)
- * (multiplicação)
- / (divisão)
- ** (exponenciação)

Os operandos podem ser constantes, variáveis, funções aritméticas ou expressões aritméticas entre parênteses. É importante observar que uma expressão aritmética pode ser constituída por um único operando, sem a presença obrigatória de um operador.

Os operadores devem ser explícitos:

Exemplo: Para representar 2 vezes ICHUVA, deve-se usar 2*ICHUVA e não 2ICHUVA.

Não é permitido o uso de operadores adjacentes.

Exemplo: A*-B dever ser representado por A*(-B).

A avaliação das expressões obedece à seguinte hierarquia das operações:

No caso de operadores com mesma hierarquia, a avaliação é realizada da esquerda para a direita (exceção feita à dupla exponenciação que é feita da direita para a esquerda)

Os parênteses alteram a prioridade do cálculo: a expressão mais interna é calculada antes.

Exemplo: A/2*B é diferente de A/(2*B)

2.6. Funções Embutidas.

As *funções embutidas (Intrínsecas)* são funções de uso freqüente, já providas pela linguagem FORTRAN. Elas estão armazenadas numa biblioteca de funções do sistema computacional.

As funções embutidas de tipo numérico, cujos resultados são valores numéricos, podem ser usadas como operandos nas expressões aritméticas.

Para utilizar as funções de biblioteca, bastam referências a seus nomes, acompanhados de um conjunto de valores de entrada (argumentos). Deve-se observar, para a correta utilização destas funções, os tipos dos valores de entrada exigidos. O tipo do resultado fornecido pela função também é pré-definido pela linguagem e deve ser corretamente utilizado nas expressões aritméticas.

A seguir, são apresentadas algumas das principais funções embutidas, onde X é um argumento do tipo real e I é um argumento do tipo inteiro.

Função matemática	**FORTRAN**	**Tipo de Resultado**
SEN (X)	SIN(X)	REAL
COS(X)	COS(X)	REAL
TG(X)	TAN(X)	REAL
$\sqrt{X}$	SQRT(X)	REAL
Inteiro $\Rightarrow$ real Real $\Rightarrow$ inteiro	REAL(I) ou FLOAT (I) INT(X) ou IFIX(X)	REAL INTEIRO
|X|	ABS(X)	REAL
e^x	EXP(X)	REAL
Ln(X)	ALOG(X)	REAL
$Log_{10}(X)$	ALOG10(X)	REAL

É possível a combinação de funções:

Exemplos:

a) SQRT(REAL(I))

INT(A*SIN(X))

2.7. Comandos da Linguagem FORTRAN.

Existem duas categorias de comandos na linguagem FORTRAN:

Comandos executáveis: são os comandos que executam cálculos, E/S, testes ou alteram o fluxo de controle.

Comandos não-executáveis: são comandos que descrevem as características da unidade de programa, dos dados, de informações de edição, etc., mas não causam uma ação a ser tomada pelo computador durante a execução do programa.

Os comandos FORTRAN podem ser agrupados, de uma forma geral, como se segue:

Comando de Atribuição: este comando permite atribuir (associar) um valor a uma variável

Exemplos:

```
R= 4.0
ZETA=(Z/L)
```

Observações:

- Em expressões aritméticas mistas, isto é, com constantes e/ou variáveis reais e inteiras, o resultado é sempre real.

- Em expressões aritméticas que contenham somente entidades do tipo inteiro, os resultados das suas avaliações são inteiros.

- Cuidado com os limites máximos dos valores envolvidos. Exemplo: A =B*G/X. Se B≅G≅X≅Lmax, onde todas as variáveis são do tipo real e Lmax é a magnitude máxima dos números reais, então, com a multiplicação (primeira operação), ocorreria um estouro da capacidade de representação da máquina. Uma possível solução neste caso, seria fazer:

```
A=B*(G/X)
```

- Cuidado com os truncamentos na manipulação de números inteiros. Exemplo: cálculo de 80% de um valor X

Y=80/100*X daria zero, pois o resultado da divisão 80/100 (duas constantes inteiras) produz necessariamente um valor do tipo inteiro, no caso, zero. Uma possível solução seria utilizar valores do tipo real.

Comando de Controle: estes comandos permitem alterar a seqüência de execução dos comandos no programa.

Exemplos:

```
      IF(N.EQ.10) GOTO 10
      IF(N.EQ.11)GOTO 11
10    WRITE(*,*) ' N igual a 10'
11    WRITE(*,*) ' N igual a 11'
```

Comando de E/S: estes comandos permitem realizar a transferência de informações entre o usuário (meio externo) e a máquina (computador)

Exemplos:

```
OPEN(10, FILE='EXEMPLO.DAT',STATUS='OLD')
WRITE(20,100) ICHUVA
READ(50,*) VENTO(I)
```

Comando de Programa Principal – PROGRAM: este comando atribui um nome ao programa principal. É um comando opcional e não-executável. Deve ser o primeiro comando utilizado em um programa.

Exemplo:

```
PROGRAM CALC_MEDIA
```

Comandos de Declaração ou Especificação: são comandos não-executáveis.

Exemplos:

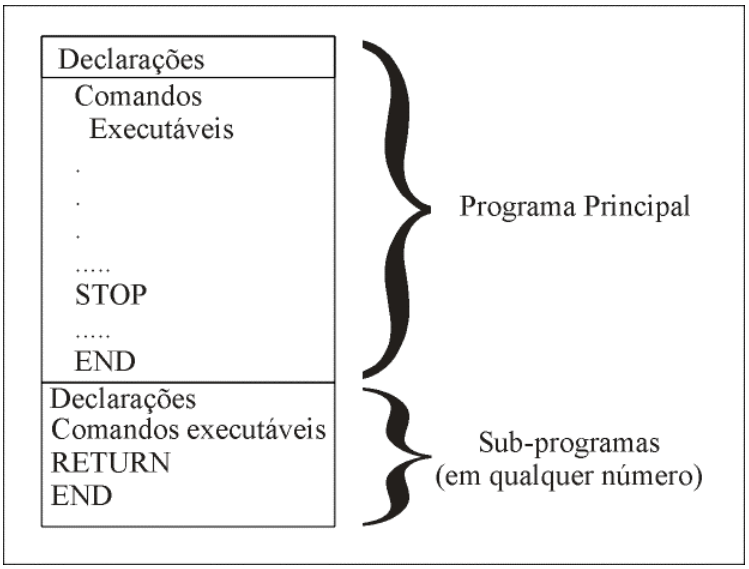
```
DIMENSION MEDIA(20,50), ICHUVA (20,50,30)
PARAMETER (N=10, M=30)
REAL IPREC, CHUVA
INTEGER VAR1
CHARACTER AUX2*3,NOME*10
```

Comando de Subprogramas: estes comandos permitem dar nomes e especificar parâmetros para funções e sub-rotinas.

Exemplos:

```
SUBROUTINE MEDIA(VAR1,INIV,P)
REAL FUNCTION FAT(N)
```

Os comandos FORTRAN devem obedecer a uma ordem pré-estabelecida na formação das unidades de programa. A figura abaixo apresenta, de um modo geral, uma estrutura simplificada desta ordem.

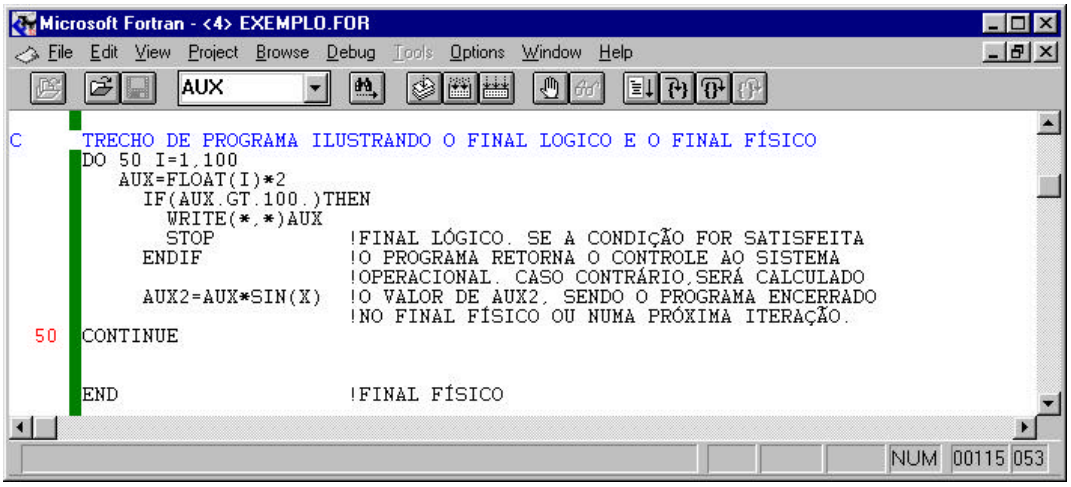


O comando *END* tem como único objetivo, indicar ao compilador o ***fim físico*** da unidade de programa. Deste modo, ele só deve ocorrer como o último comando de uma unidade, e é obrigatório.

O comando *STOP* encerra a execução da unidade de programa e retorna o controle ao sistema operacional, ou seja, indica o ***final lógico*** da execução programa. Para este mesmo fim, alternativamente, em alguns ambientes de programação, pose-se utilizar o comando *CALL EXIT*; esta utilização não é recomendável por questões de compatibilidade entre ambientes de programação distintos.

Na ausência de um *STOP*, o compilador assume que o fim lógico da unidade de programa coincide com o seu fim físico.

A figura a seguir apresenta um exemplo dessa situação.



2.8. Exercícios Práticos.

1) Instalação e Utilização do Microsoft Fortran Visual Workbench.
2) Construção de programas simples.
    a. Programa para somar uma variável
    b. Programa para calcular uma média entre dois valores
    c. Programa para calcular o seno, o cosseno, a tangente, a raiz quadrada, e o módulo de uma variável.
    d. Programa para obter a parte inteira de uma variável real.
    e. Programa para verificar o funcionamento dos comandos *STOP* e *END*.

3. Comandos de Entrada e Saída – E/S.

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

3.1. Entrada de Dados.

A entrada de dados em FORTRAN é feita através do comando *READ* e pode ser feita através do teclado ou de um arquivo externo. O comando *READ* tem a forma geral:

READ(i,n) VAR1, VAR2, VAR3, ...

Onde:

- ***i*** indica o número da unidade de entrada (para entrada via teclado usa-se o sinal *)
- ***n*** indica o número do comando *FORMAT*, que deve ser indicado entre as colunas 1 e 5, e especifica o formato dos dados de entrada. Para o caso de formato livre é utilizado o sinal *. Neste caso, os dados devem ser separados por um espaço em branco ou estar em linhas diferentes.

3.2. O 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 as E/S são mais elaboradas.

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

As especificações de formato mais usuais são:

- especificação numérica para inteiro;
- especificação numérica para real sem expoente;
- especificação numérica para real com expoente;
- especificação para caractere.

3.2.1. Especificação numérica para inteiro.

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

aIw

onde:

- **a** é uma constante inteira e sem sinal que indica o número de vezes que esta especificação I deve ser obedecida, repetitivamente;
- **I** é a indicação para inteiro;
- **w** é uma constante inteira e sem sinal (não nula) que indica o comprimento do campo do dado (em caracteres).

Como um exemplo, a figura abaixo ilustra o procedimento para ler o arquivo DATA.DAT, que possui variáveis do tipo inteiro na seguinte seqüência: dia, mês, ano, hora, minuto, segundos.

Estrutura do arquivo DATA.DAT:

15 02 2001 17 34 38

No exemplo acima são apresentados três modos diferentes para leitura dos dados no arquivo DATA.DAT. O primeiro comando, OPEN, faz a abertura de um arquivo externo que contém os dados de interesse. Este comando será visto em mais detalhes posteriormente. No primeiro modo, utilizou-se a opção de formato livre. Esta é uma das maneiras mais utilizadas por não exigir o conhecimento prévio do tamanho de cada variável. Entretanto, é necessário que se conheça o tipo de variável (real, inteiro ou caractere). No modo 2, cada variável é identificada por um formato próprio, explícito no comando *FORMAT* associado, neste caso de número 100. As indicações "1X" nesta declaração de formato, indicam um espaço em branco existente entre os dados. Pode-se omitir esta indicação incluindo o espaço em branco ao tamanho da variável lida. No nosso exemplo, para a variável IDIA e para as outras que são formadas por dois dígitos, poderia ser atribuído o valor 3 (inteiros com três dígitos) e eliminar a utilização da indicação de espaço em branco. Este procedimento é utilizado no modo 3, que, além dessa, faz outra simplificação para o caso em que variáveis com o mesmo número de dígitos podem ser representados pelo fator de repetição **a** anteriormente citado. Por exemplo, sabe-se que as duas primeiras variáveis que serão lidas possuem dois dígitos, seguidos por um espaço em branco. A primeira simplificação é considerar essas variáveis como tendo três dígitos. A segunda simplificação, é utilizar o fator de repetição. Sendo assim, a leitura dessas variáveis fica 2I3, conforme ilustrado na figura.

3.2.2. Especificação numérica para real sem expoente.

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

aFw.d

onde:

- **a** é uma constante inteira e sem sinal que indica o número de vezes que a especificação **F** deve ser obedecida, repetitivamente;
- **F** é a indicação para real;
- **w** é uma constante inteira e sem sinal (não nula) que indica o comprimento do campo do dado (em caracteres).
- **d** é uma constante inteira e sem sinal, que indica o número de casas decimais.

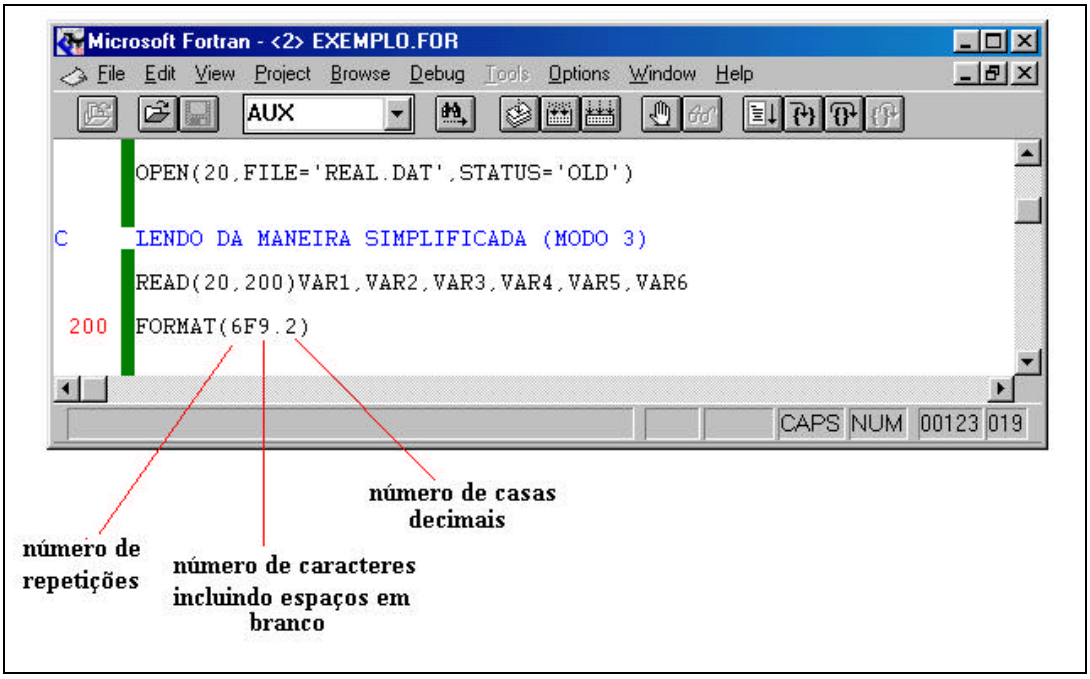
ou seja,

$$aFw.d \Rightarrow \underbrace{XXX.XX}_{w \text{ caracteres}} \quad \left.\begin{matrix}\\ \\ \\ \\ \end{matrix}\right\} d \text{ casas decimais}$$

Como exemplo, a figura a seguir apresenta o modo de leitura para valores reais existentes no arquivo REAL.DAT. Os três modos apresentados para leitura de valores inteiros, também são válidos para reais. Por simplicidade, apenas o modo 3 é apresentado nessa figura.

Estrutura do arquivo REAL.DAT (existem três espaços entre cada valor):

```
   602.83   602.61   601.36   604.44   609.58   610.81
```



3.2.3. Especificação numérica para real com expoente.

A especificação numérica para real com expoente é utilizada para definir o comprimento do campo e o número de casas decimais associados aos valores reais com expoente, lidos ou escritos. Sua forma geral é:

aEw.d

onde:

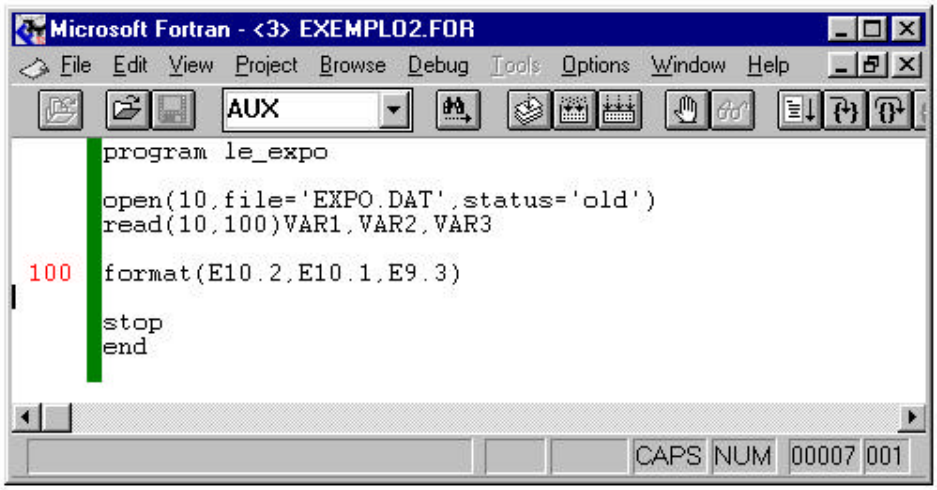
- **a, w e d** têm o mesmo significado das especificações anteriores
- **E** é a indicação para real com expoente.

Na entrada de dados, valem as mesmas convenções sobre o ponto do formato **Fw.d**. Deve-se, entretanto, acrescentar ao valor de w o número de dígitos utilizados a partir do indicador E. Na saída, o dígito mais significativo aparece logo à direita do ponto. O número de dígitos significativos depende de **d**. Havendo mais espaços, descontado o campo do sinal negativo quando for o caso, são impressos brancos. Por garantia, deve-se fazer w≥ d+7.

A figura a seguir ilustra o caso para abertura do arquivo EXPO.DAT.

Estrutura do arquivo EXPO.DAT (para facilitar, os espaços em branco estão denotados pelo sinal "_".

```
_ _ _ _6.32E2 _6320.0E-1 _0.632E03
```



3.3. Comandos *OPEN* e *CLOSE*.

Em alguns exemplos vistos anteriormente, vimos que quando a entrada de dados é feita através de um arquivo externo, é necessária a abertura do mesmo pelo FORTRAN. Isto é feito através do comando *OPEN*.

O comando *OPEN* inicia o arquivo para que as operações de E/S possam ser realizadas sobre ele. Através deste comando um número de unidade é associado a um arquivo específico. Também, através do comando OPEN, são especificados os atributos do arquivo.

A forma geral deste comando é:

```
OPEN (<unidade>,FILE='<nome do arquivo>',STATUS='<status>',
      ACESS = '<acesso>',FORM='<formato>',IOSTAT='< iocheck >,
      ERR=<errlabel>,RECL=<comprimento do registro>)
```

Dependendo da versão do compilador FORTRAN, outros especificadores podem ser incluídos no comando *OPEN*.

<unidade> - expressão do tipo inteiro, que indica o número da unidade associado ao arquivo que esta sendo iniciado.

<nome> - é o nome do arquivo. Deve ser especificado de acordo com as convenções do sistema operacional. Geralmente, é uma expressão do tipo caractere. Pode ser incluído ao nome do arquivo o diretório em que ele se encontra. Se nenhuma indicação é feita, o FORTRAN irá procurar o arquivo no diretório corrente.

<status> - pode ser: OLD, NEW, SCRATCH ou UNKNOWN. Utiliza-se OLD quando o arquivo já existe. NEW, para criar novos arquivos. Quando esse status é utilizado, a cada vez que o programa é rodado, é necessário que o arquivo seja apagado previamente. SCRATCH, para arquivos que existirão somente durante a execução do programa. Neste caso, o especificador FILE deve ser omitido (pouco utilizado). UNKNOWN, quando não se aplica nenhum dos casos anteriores. Geralmente é utilizado para arquivos de saída. A vantagem desse status é poder rodar o programa várias vezes sem ter a necessidade de apagar previamente o arquivo de saída.

<acesso> - pode ser: SEQUENTIAL ou DIRECT, para indicar acesso seqüencial (arquivos formatados, ASCII) ou direto (arquivos binários). O padrão (default) é seqüencial.

<formato> - pode ser: FORMATTED ou UNFORMATTED, para indicar arquivo formatado ou não-formatado. O padrão é formatado para acesso seqüencial e não-formatado para acesso direto.

< iocheck > é uma variável do tipo inteiro que armazena um valor indicativo do status da operação de E/S sendo utilizada, podendo indicar, numa condição de erro, o tipo do erro ocorrido. O valor assumido por < iocheck > é dependente do sistema utilizado. Entretanto, em geral, 0 (zero) indica ausência de erros; valores negativos indicam fim de arquivo; valores positivos indicam erro.

<errlabel> - o rótulo de uma afirmação executável na mesma unidade de programa. Um erro de E/S causa a transferência do controle para a afirmação em errlabel. Se for omitido, o efeito de um erro de E/S é determinado pela presença ou ausência de iocheck

<número de registro> - este especificador só deve ser utilizado em comandos de E/S relacionados com arquivos de acesso direto (ou aleatório). É uma expressão do tipo inteiro, positiva, e especifica o comprimento, em bytes, do registro que está sendo

acessado. Quando for apresentada a maneira de escrever um arquivo para ser lido no Software GrADS, este especificador será mais bem explicado.

O comando CLOSE tem o efeito inverso ao do comando OPEN. Após a execução do CLOSE, o número de unidade é desconectado do arquivo em questão. Deste modo, este número pode ser utilizado para um outro arquivo diferente. Se este comando não for utilizado para fechar um arquivo, a execução dos comandos STOP ou END irá fecha-lo automaticamente. Neste sentido, o uso do comando CLOSE é opcional, embora seja recomendável e, algumas vezes, indispensável.

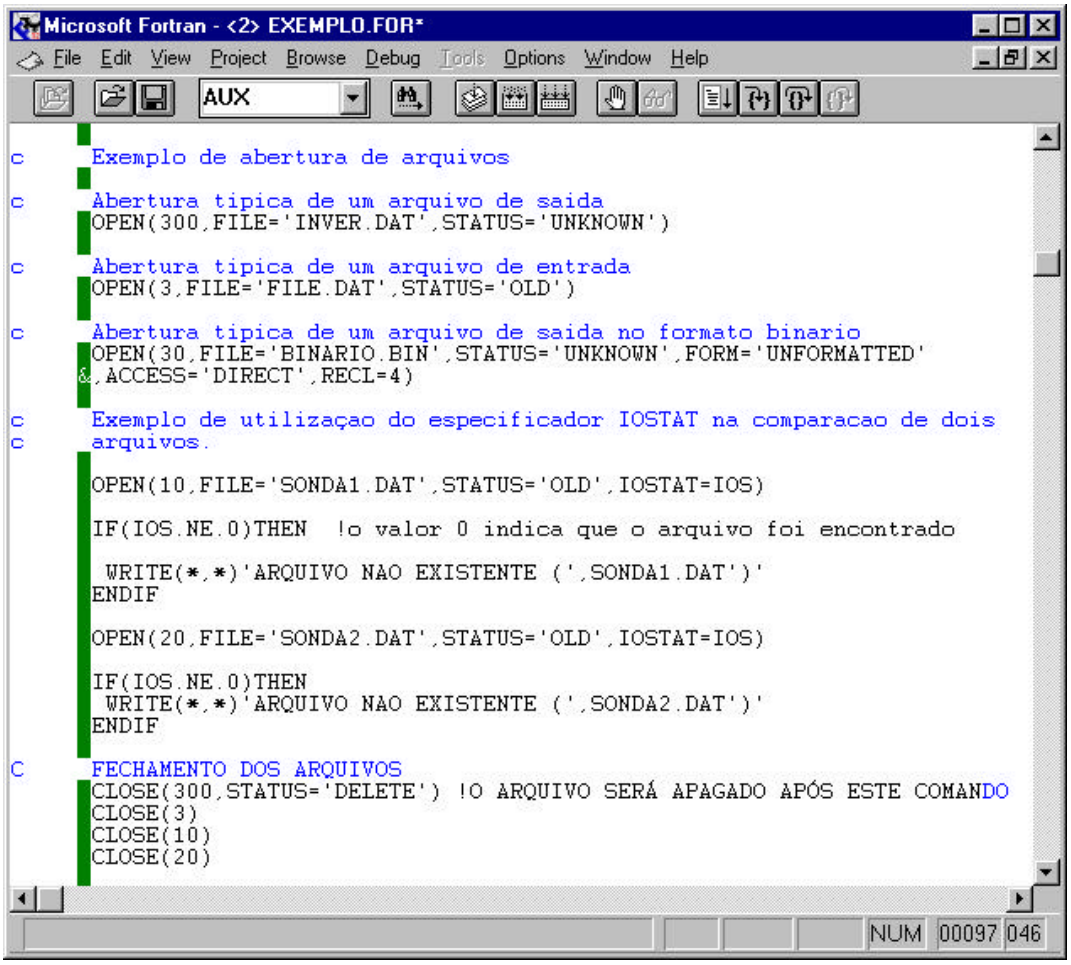
Seu formato geral é:

```
CLOSE(<unidade>,STATUS='<status>',IOSTAT=< iocheck >)
```

<unidade> e <iocheck> têm os mesmos significados descritos anteriormente.

<status>- pode ser KEEP ou DELETE, indicando se o arquivo deve ser mantido ou apagado após a execução do comando CLOSE. O padrão é KEEP, exceto para os arquivos criados sem o especificador FILE (arquivos do tipo SCRATCH).

A figura abaixo dá um exemplo da abertura e do fechamento de alguns arquivos com diferentes características.



Observe que a unidade 30 não foi fechada. Isto será feito automaticamente pelos comandos STOP e END ao término do programa.

3.4. Saída de dados.

A saída de dados em FORTRAN é feita através do comando *WRITE* e pode ser feita através do monitor (tela) ou para um arquivo externo. O comando *WRITE* tem a forma geral:

WRITE(i,n) VAR1, VAR2, VAR3, ...

onde **i** e **n** têm os mesmos significados apresentados para o comando *READ*.

As especificações de formato para o comando *WRITE* seguem as mesmas regras do comando *READ*. Exceção é feita para valores reais com expoente. Neste caso, os valores serão impressos na forma:

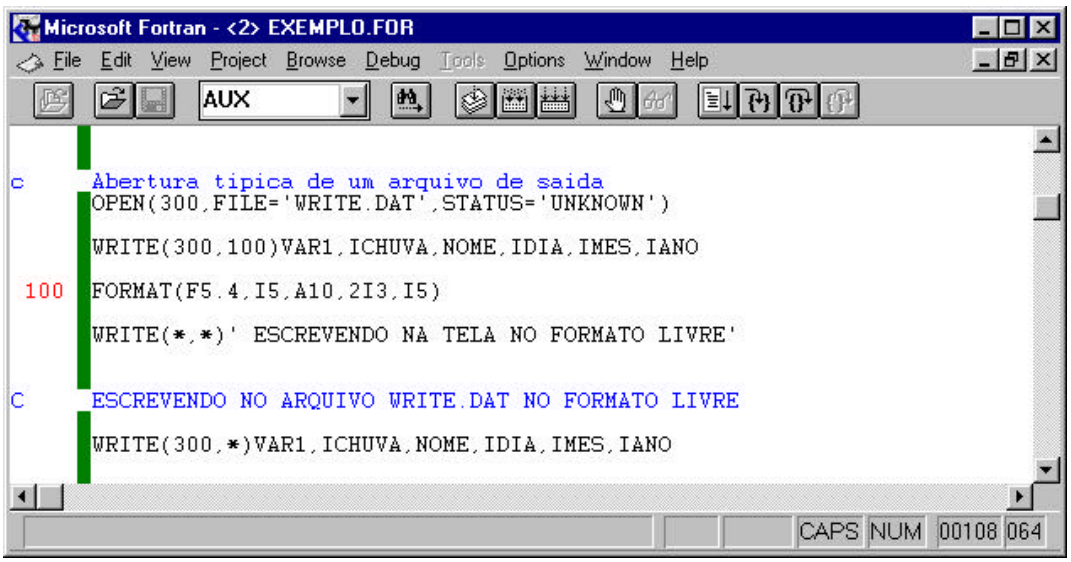
<valor>E<expoente com dois dígitos e sinal>

onde <valor> será sempre escrito com o primeiro algarismo significativo à direita do ponto. Ex: o valor 1,6 x 10^{-19} será impresso, utilizando o formato **E10.2**, como:

_ _ _.16E-18

Deve-se tomar cuidado com o número de dígitos reservados para a escrita. No exemplo acima, o número mínimo de dígitos é 7, ou seja, E7.2. Caso o número seja insuficiente, serão impressos na tela sinais do tipo '********', indicando que o formato utilizado para a escrita é insuficiente.

A figura a seguir apresenta alguns exemplos da utilização do comando *WRITE.*



3.5. Exercícios práticos.

- Abertura de arquivos, leitura de variáveis, operações com variáveis, escrita de resultado.

4. Conjuntos e variáveis indexadas

Como citado anteriormente, as variáveis de um programa podem ser classificadas em variáveis simples e variáveis compostas. As variáveis simples armazenam, num dado instante, um único valor. Por outro lado, as variáveis compostas homogêneas podem armazenar um conjunto de valores de mesmo tipo, que podem ser acessados individualmente. As variáveis compostas homogêneas (*arrays*) permitem a implementação de estruturas de dados mais complexas que as permitidas pelo uso das variáveis simples. Em FORTRAN, os conjuntos homogêneos são representados por seqüências de variáveis indexadas.

4.1. Vetores

Um vetor é um conjunto unidimensional de variáveis. Um nome é associado ao vetor como um todo, sendo cada elemento deste vetor referenciado por um índice, entre parêntesis, que acompanha o nome atribuído ao vetor. O índice que discrimina os elementos constituintes do vetor pode ser qualquer expressão do tipo inteiro. Consideremos, por exemplo, uma variável que contenha dados horários de precipitação para um dia. Esta variável pode ser tratada como um vetor contendo 24 elementos, na forma:

```
chuva(1)
chuva(2)
chuva(3)
.
.
.
chuva(24)
```

As variáveis indexadas podem ter até 7 índices. Para o caso de um vetor, todos os elementos têm o mesmo tipo. Os elementos de um vetor são armazenados em posições consecutivas da memória do computador.

O número de elementos de um vetor e os limites superior e inferior para o índice devem ser declarados no início do programa. Uma das possíveis formas é a utilização da declaração *DIMENSION*.

4.2. Declaração *DIMENSION*

DIMENSION é uma declaração e, portanto, deve vir antes do primeiro comando executável do programa e após as declarações de tipo e de constantes simbólicas.

Se o limite inferior do índice não for especificado, ele é assumido como sendo igual a 1.

Sua forma geral é:

DIMENSION var1(e_{1i}:e_{1s}), var2(e_{2i}:e_{2s}), ... varN(e_{Ni}:e_{Ns})

Sendo:

- vari os nomes dos vetores;

- e_{ii} limites inferiores para o vetor i;

- e_{is} limite superior para o veto i.

Os limites inferior e superior devem ser constantes inteiras e devem satisfazer a condição ($e_{is} > e_{ii}$)

Exemplo:

DIMENSION chuva(100), media(0:25)

A declaração acima informa ao compilador que a variável chuva é um vetor com 100 elementos. Como não foi declarado o limite inferior, o compilador assumirá o valor 1. Para a variável média o compilador entenderá que esta variável é um vetor que possui 26 elementos, sendo o limite inferior igual a 0.

Outra maneira de declaração do número de elementos de um vetor é através da declaração de tipo. A declaração de tipo pode substituir completamente a declaração *DIMENSION*, como será visto a seguir.

4.3. Declarações de tipo.

Alguns exemplos de declaração de tipo são: *REAL* e *INTEGER*. O nome, o número de elementos e os limites para o índice de um vetor podem ser especificados por declarações de tipo. Neste caso, a declaração *DIMENSION* não deve ser utilizada.

Exemplos:

```
REAL chuva(0:50), rad(24)
INTEGER ihora(24), dia(31)
```

4.4. Declarações *DIMENSION* para matrizes.

A declaração *DIMENSION* para matrizes é semelhante àquela para vetores. A diferença é que todas as dimensões devem ser indicadas. Este tipo de declaração é muito utilizado na criação de grades regulares para serem lidas no GrADS.

Exemplos:

DIMENSION topo(100,100), solo(100,100), vento(100,100,20) etc.

O exemplo acima indica que a variável topo é uma matriz de 100 colunas por 100 linhas. O mesmo acontece para a variável solo. A variável vento esta disposta na mesma grade que as variáveis anteriores mas possui uma terceira dimensão, por exemplo, os níveis verticais.

As declarações de tipo, semelhantemente ao caso vetorial, podem substituir a declaração DIMENSION.

Exemplo:

INTEGER isolo(50,50), indk(50,50,15)

REAL vento(100,100,15), geop(100,100,15), relhum(100,100,10), etc.

Nos exemplos acima, isolo é uma variável do tipo inteira disposta numa matriz de 50 x 50 elementos, indk é uma variável do tipo inteira com três dimensões tendo 50 x 50 elementos na horizontal e 15 na vertical. Vale lembrar que a declaração de tipo sobrepõe-se a declaração explicita para inteiros e reais, ou seja, independente da letra que inicie o nome da variável, ela será tratada no FORTRAN como sendo real ou inteira, de acordo com a declaração de tipo utilizada. As variáveis reais do exemplo acima, muito comuns em Meteorologia, indicam que todas estão dispostas em grades de 100 x 100 elementos sendo que as variáveis vento e geop possuem 15 níveis verticais e a variável relhum possui apenas 10. Esta situação é muito comum em dados de reanálise e previsões dos modelos globais.

Na seção 6 será visto como trabalhar com variáveis complexas utilizando a linguagem FORTRAN.

5. Cadeias de Caracteres

5.1. Constantes do tipo Caractere

Um caractere é uma letra, digito ou símbolo especial do conjunto de caracteres da linguagem FORTRAN.

Uma cadeia de caracteres é uma seqüência de um ou mais caracteres.

Uma constante caractere é uma cadeia de caracteres entre apóstrofos.

Ex: ‘nome’, ‘CURSO D’’AGUA’, etc.

O comprimento máximo de uma constante caractere depende do tipo de plataforma utilizada. Como exemplo, para a IBM são permitidos 1316 caracteres. Vale lembrar que, quando tratamos de caracteres, espaços em branco devem ser considerados.

5.2. Variáveis do tipo Caractere

As variáveis do tipo caractere são variáveis que armazenam constantes do tipo caractere.

Uma variável caractere é declarada através de especificações explicitas, declaração CHARACTER, ou através da declaração IMPLICIT.

Ex:

```
IMPLICIT CHARACTER*28 (X,Y,C)
CHARACTER*36 LINHA
CHARACTER NOME*15, ARQ*12
```

Nestes exemplos, as variáveis X,Y e C têm comprimento 28, LINHA tem comprimento 36, NOME tem comprimento 15 e ARQ tem comprimento 12. Se nenhum comprimento é declarado, assume-se que cada variável tem comprimento 1.

5.3. Conjuntos de Caracteres.

Da mesma maneira que números inteiros e reais podem ser parte de conjuntos e declarados pelas declarações de tipo REAL ou INTEIRO e pela declaração DIMENSION, cadeias de caracteres também podem ser representadas por conjuntos tais como vetores, matrizes, etc.

Na declaração de conjuntos do tipo caractere, a especificação de comprimento, quando associada a uma variável da lista, deve vir após a especificação das dimensões do conjunto.

Ex:

```
CHARACTER ARQUIVO(30)*12
```

No exemplo anterior, a variável ARQUIVO é um conjunto de 30 elementos tendo cada um deles o comprimento 12.

5.4. Constantes Simbólicas do Tipo Caractere.

Para a associação de nomes às constantes do tipo caractere, pode-se usar a declaração PARAMETER como no caso das constantes numéricas.

Ex:

```
CHARACTER NOME*12
PARAMETER (NOME='12345678.DAT')
```

Uma outra forma válida para conseguir o mesmo efeito (somente com a declaração PARAMETER) é:

```
CHARACTER NOME*(*)
PARAMETER (NOME='12345678.DAT')
```

Esta forma é mais conveniente, principalmente quando não se conhece o comprimento da constante.

É possível a utilização de expressões com cadeias de caracteres, ou seja, uma combinação de operandos através de operadores. Expressões do tipo caractere podem combinar operandos do tipo caractere, através do operador de concatenação (//). Deve-se notar que não se pode misturar, numa única expressão, dados numéricos e caracteres.

Ex:

```
CHARACTER N*5, B*4,C*7,CURSO*16
PARAMETER (N='CURSO',B=' DE ',C='FORTRAN')

CURSO=N//B//C
WRITE(*,*) CURSO

RESULTADO =  CURSO DE FORTRAN
```

No exemplo acima, foi utilizado um comando de atribuição para a variável CURSO.

Se o comprimento da variável é maior que o do resultado da expressão, então o valor resultante da expressão é armazenado na parte mais à esquerda da variável. O restante do espaço é preenchido com brancos. Se o comprimento da variável é menor

que o do resultado da expressão, então somente os caracteres mais à esquerda serão armazenados.

Ex:

```
CHARACTER NOME*4

NOME='CURSO DE FORTRAN'

WRITE(*,*) NOME

RESULTADO = CURS
```

5.5. Subcadeias de Caracteres.

Uma subcadeia é uma seqüência de uma ou mais unidades consecutivas de armazenamento de caractere, dentro de uma variável. As unidades dentro da variável são numeradas da esquerda para a direita, a partir de 1. O recurso de subcadeia permite a manipulação de partes de uma determinada cadeia de caracteres. Sua forma geral é:

< nome da variável caractere> (e_1:e_2)

sendo e_1 e e_2 expressões aritméticas inteiras, denominadas expressões de subcadeias. O índice 1 especifica a posição do primeiro caractere da subcadeia e o índice 2 a posição do último caractere. Se e_1 for omitido, considera-se que todos os caracteres até e_2 serão considerados. Se e_2 for omitido, considera-se que todos os caracteres a partir de e_1 serão considerados. Se e_1 e e_2 forem omitidos, todos os caracteres serão considerados.

Ex:

```
CHARACTER*5 VAR
VAR='LABEL'

VAR(1:2) = 'LA'
VAR(3:4) = 'BE'
VAR(3:) = 'BEL'
VAR(:3) = 'LAB'
VAR(:) = 'LABEL'
```

5.6. O comando FORMAT para caracteres.

Com os comandos de entrada e saída para caracteres, pode-se utilizar a Especificação de formato A ou o Formato Livre.

O Formato A tem a forma geral:

aAw

sendo a o fator de repetição e w o número de caracteres do campo.

Se w > comprimento da variável (**n**), na entrada de dados, somente os caracteres mais à direita são armazenados. Na saída **n** caracteres são impressos à direita no campo e o restante do espaço é preenchido com brancos.

Se w < comprimento da variável (**n**), na entrada, w caracteres são lidos e armazenados nas posições mais a esquerda da variável. O restante do espaço é preenchido com brancos. Na saída, os w caracteres mais à esquerda da variável serão impressos e o resto é truncado.

Quando se usar o formato livre com caracteres, deve-se observar que, na entrada, a lista de variáveis pode conter nomes de variáveis ou subcadeias. Os dados de entrada devem vir entre apóstrofos

Ex:

```
CHARACTER NOME*20, CURSO*10
READ(*,*) NOME (11:20), NOME(11:10),CURSO
```

Dados de entrada:

'SILVA' 'MARCOS' 'METEO'

Valores armazenados

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Nome	M	A	R	C	O	S					S	I	L	V	A					

	1	2	3	4	5	6	7	8	9	10
Curso	M	E	T	E	O					

Quando o comprimento dos dados é diferente do comprimento da variável, as regras são as mesmas válidas para o comando de atribuição. Num mesmo READ, podem ser lidos valores numéricos junto com caracteres. Os tipos de variáveis na lista devem ser correspondentes aos tipos dos dados.

Na saída, a lista de variáveis ou de expressões pode conter nomes de variáveis, constantes ou subcadeias.

Todos os operadores relacionais, já vistos, podem ser utilizados com operandos do tipo caractere. Entretanto, não se podem comparar expressões de caracteres com expressões numéricas.

5.7. Funções intrínsecas para caracteres.

Existem algumas funções intrínsecas específicas para o tratamento de caracteres. Neste curso serão vistas as funções LEN, INDEX, ICHAR e CHAR.

5.7.1. Função LEN

Esta função fornece o comprimento da cadeia de caracteres dada como argumento. O resultado é um número inteiro.

Ex:

```
LEN('CASA') retorna o valor 4.
LEN('D''AGUA') retorna o valor 6

CHARACTER C*7
C='CURSO'
```

LEN(C) – Embora a cadeia de caracteres atribuída a variável C tenha cinco caracteres a função LEN retornará o valor 7, pois o comprimento da variável C é fixo e igual a 7.

5.7.2. Função INDEX.

Esta função localiza a primeira ocorrência de uma subcadeia a_2 na cadeia a_1. O resultado é um valor do tipo inteiro. Se a subcadeia a_2 não for encontrada na cadeia a_1, o valor resultante será zero. Se houver mais de uma ocorrência, àquela mais a esquerda será indicada.

Ex:

```
INDEX('CURSO','R') retorna o valor 3
INDEX('FORTRAN','R') retorna o valor 3
```

5.7.3. Função ICHAR.

Esta função converte um caractere ASCII em um número inteiro. O intervalo é de 0 a 255. O argumento utilizado deve ser composto de apenas um caractere.

Ex:

```
ICHAR('A') retorna o valor 65
ICHAR('2') retorna o valor 50
```

Um exemplo de correspondência entre caracteres ASCII e números inteiros pode ser visto na tabela a seguir.

Alguns valores inteiros e sua correspondência na tabela ASCII.

Valor inteiro	Car. ASCII	Valor inteiro	Car. ASCII	Valor inteiro	Car. ASCII
33	!	48	0	63	?
34	“	49	1	64	@
35	#	50	2	65	A
36	$	51	3	66	B
37	%	52	4	89	Y
38	&	53	5	90	Z
39	‘	54	6	91	[
40	(	55	7	92	\
41	)	56	8	93	]
42	*	57	9	94	^
43	+	58	:	95	_
44	,	59	;	96	`
45	-	60	<	97	a
46	.	61	=	98	b
47	/	62	>	122	z

5.7.4. Função CHAR

Executa o inverso da função ICHAR, ou seja, converte um número inteiro num caractere ASCII. A tabela mostrada acima foi criada com a utilização desta função. Abaixo, segue o trecho de programa para criar esta tabela.

```
      program int_car
      character*1 a
            open(10,file='ASCII.dat',status='unknown')
       do i=0,255
            a=char(i)
            write(10,100)i,a
        enddo
100   format (i4,1x,a1)
      stop
      end
```

6. Estruturas de Controle

As estruturas de controle permitem ao programador a especificação do fluxo de controle de um programa; em outras palavras, permitem alterar o fluxo de execução dos componentes de um programa, através de comandos que direcionem a execução sob uma determinada condição ou de maneira iterativa.

As estruturas de controle compreendem seqüência, seleção e iteração.

A estrutura de seqüência é naturalmente implementada em FORTRAN a medida que os comandos são organizados seqüencialmente em um programa. Os dois outros tipos de estruturas serão vistos a seguir.

6.1. Estrutura de Seleção.

O comando FORTRAN que implementa a estrutura de seleção é o *IF-BLOCO*, através do uso de operadores relacionais em expressões lógicas.

As expressões lógicas são aquelas expressões que quando avaliadas resultam num valor lógico: verdadeiro ou falso. São constituídas pela combinação de operandos lógicos, através de operadores lógic os que podem ser: constantes lógicas; variáveis lógicas; funções lógicas; expressões relacionais ou expressões lógicas entre parênteses.

Constantes lógicas especificam um valor lógico: verdadeiro ou falso. Na linguagem FORTRAN:

.TRUE.

.FALSE.

As expressões relacionais constituem uma forma particular de expressões lógicas e viabilizam o estabelecimento de relações entre entidades do programa.

Os operadores relacionais possíveis na linguagem FORTRAN são:

.EQ.	=
.NE.	≠
.GT.	>
.GE.	≥
.LT.	<
.LE.	≤

O valor de uma expressão relacional é sempre .TRUE. ou .FALSE.. As expressões aritméticas comparadas podem ser de tipos diferentes (real ou inteiro).

Operadores lógicos são usados para construir condições lógicas mais complexas que aquelas descritas simplesmente por uma expressão relacional ou por um único operando lógico. Em outras palavras, eles são usados para combinar valores lógicos.

Os operadores lógicos disponíveis na linguagem FORTRAN são mostrados abaixo. Nesta tabela, A e B representam operandos lógicos quaisquer.

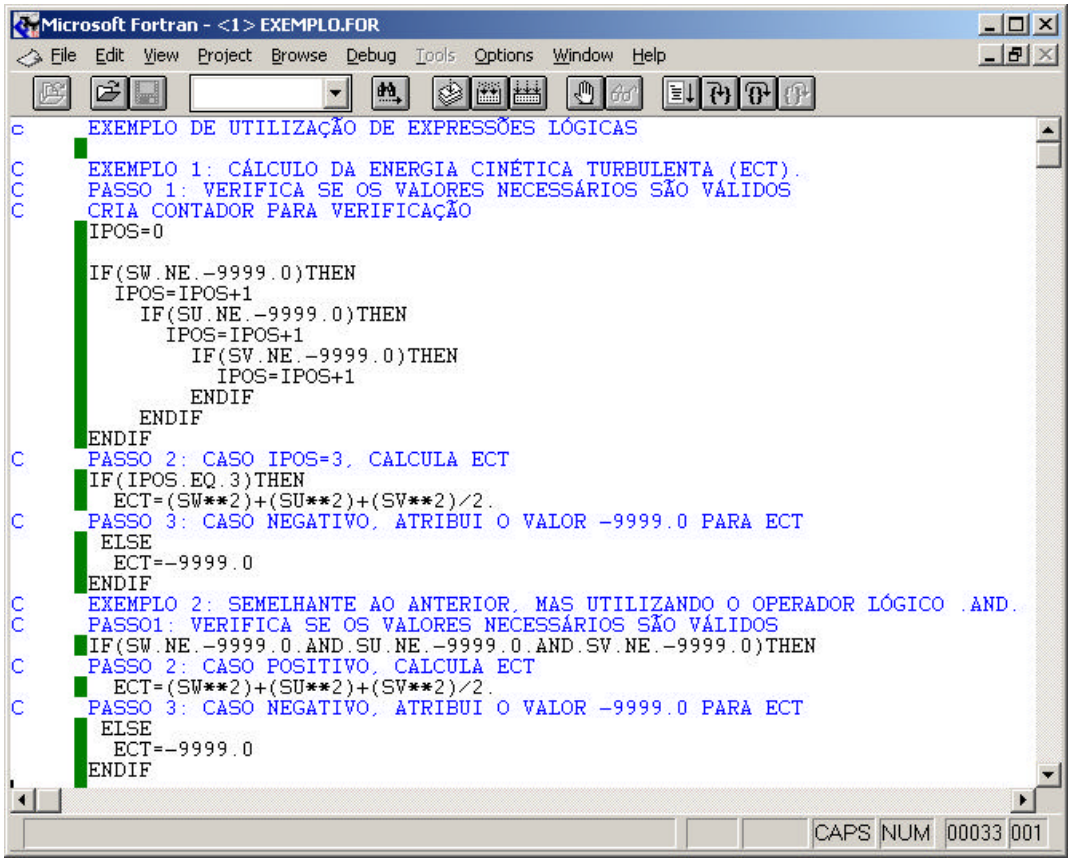
Operadores lógicos	**significado**	**Exemplo**
.NOT.	negação	.NOT.A
.AND.	"E" lógico	A.AND.B
.OR.	"OU" lógico	A.OR.B
.EQV.	Equivalência lógica	A.EQV.B
.NEQV.	Não equivalência	A.NEQV.B

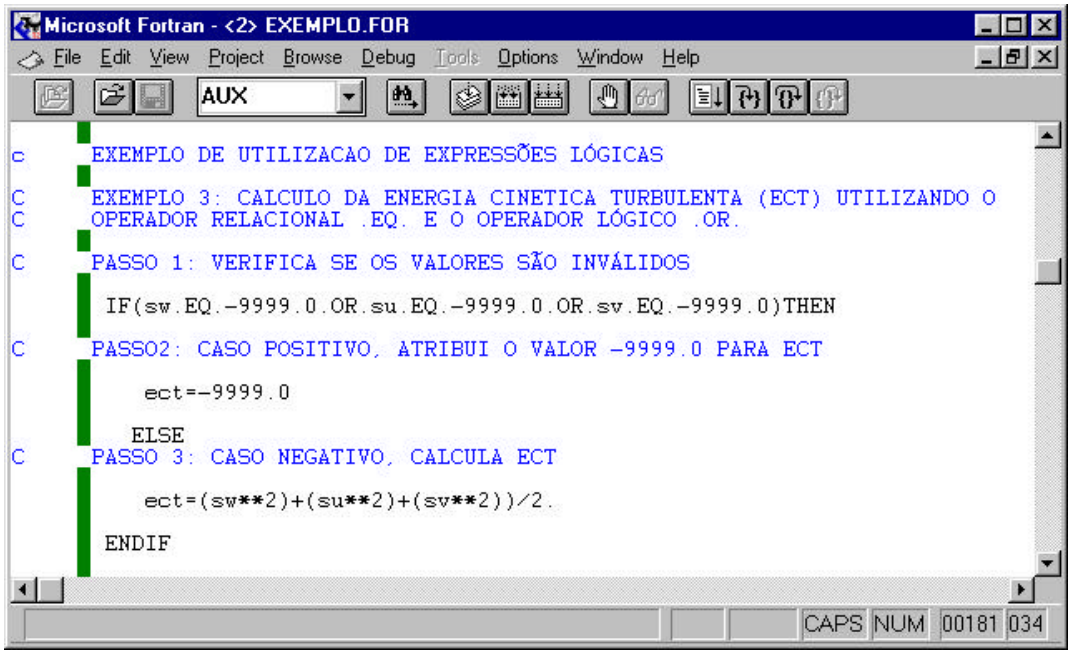
A tabela abaixo sintetiza o funcionamento dos operadores lógicos. Nesta tabela, A e B representam operandos lógicos quaisquer e as letras V e F representam, respectivamente, os valores lógicos verdadeiro e falso.

Tabela verdade dos operadores lógicos.

A	**B**	**.NOT.A**	**A.AND.B**	**A.OR.B**	**A.EQV.B**	**A.NEQV.B**
F	F	V	F	F	V	F
F	V	V	F	V	F	V
V	F	F	F	V	F	V
V	V	F	V	V	V	F

Nas figuras a seguir são apresentados alguns exemplos de utilização de estruturas de seleção.


Microsoft Fortran - <1> EXEMPLO.FOR
File Edit View Project Browse Debug Tools Options Window Help
c EXEMPLO DE UTILIZAÇÃO DE EXPRESSÕES LÓGICAS
C EXEMPLO 1: CÁLCULO DA ENERGIA CINÉTICA TURBULENTA (ECT).
C PASSO 1: VERIFICA SE OS VALORES NECESSÁRIOS SÃO VÁLIDOS
C CRIA CONTADOR PARA VERIFICAÇÃO
IPOS=0
IF(SW.NE.-9999.0)THEN
IPOS=IPOS+1
IF(SU.NE.-9999.0)THEN
IPOS=IPOS+1
IF(SV.NE.-9999.0)THEN
IPOS=IPOS+1
ENDIF
ENDIF
ENDIF
C PASSO 2: CASO IPOS=3, CALCULA ECT
IF(IPOS.EQ.3)THEN
ECT=(SW**2)+(SU**2)+(SV**2)/2.
C PASSO 3: CASO NEGATIVO, ATRIBUI O VALOR -9999.0 PARA ECT
ELSE
ECT=-9999.0
ENDIF
C EXEMPLO 2: SEMELHANTE AO ANTERIOR, MAS UTILIZANDO O OPERADOR LÓGICO .AND.
C PASSO1: VERIFICA SE OS VALORES NECESSÁRIOS SÃO VÁLIDOS
IF(SW.NE.-9999.0.AND.SU.NE.-9999.0.AND.SV.NE.-9999.0)THEN
C PASSO 2: CASO POSITIVO, CALCULA ECT
ECT=(SW**2)+(SU**2)+(SV**2)/2.
C PASSO 3: CASO NEGATIVO, ATRIBUI O VALOR -9999.0 PARA ECT
ELSE
ECT=-9999.0
ENDIF
CAPS NUM 00033 001

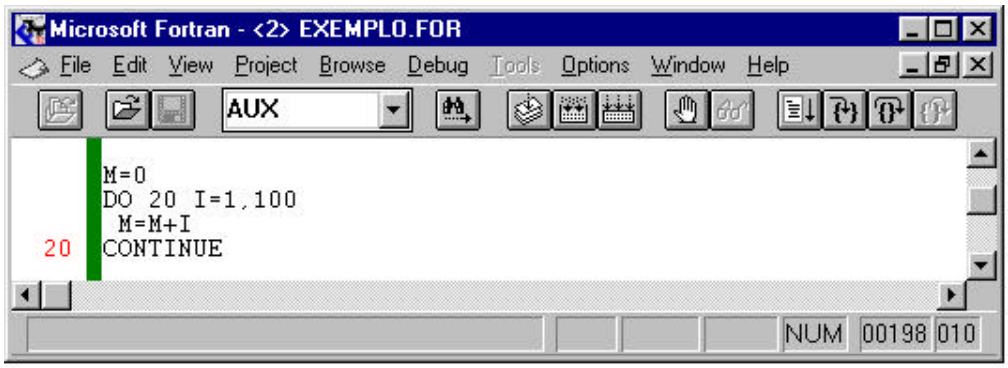

Microsoft Fortran - <2> EXEMPLO.FOR
File Edit View Project Browse Debug Tools Options Window Help
AUX
c EXEMPLO DE UTILIZACAO DE EXPRESSÕES LÓGICAS
C EXEMPLO 3: CALCULO DA ENERGIA CINETICA TURBULENTA (ECT) UTILIZANDO O
C OPERADOR RELACIONAL .EQ. E O OPERADOR LÓGICO .OR.
C PASSO 1: VERIFICA SE OS VALORES SÃO INVÁLIDOS
IF(sw.EQ.-9999.0.OR.su.EQ.-9999.0.OR.sv.EQ.-9999.0)THEN
C PASSO2: CASO POSITIVO, ATRIBUI O VALOR -9999.0 PARA ECT
ect=-9999.0
ELSE
C PASSO 3: CASO NEGATIVO, CALCULA ECT
ect=(sw**2)+(su**2)+(sv**2))/2.
ENDIF
CAPS NUM 00181 034

6.2. Estrutura de Iteração.

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

Existem, basicamente, duas formas gerais para utilização de estruturas de iteração. A primeira utiliza o número do comando que delimita a abrangência da iteração do *DO*.

Ex:



Neste modo, a forma geral para a utilização do *DO*, fica:

DO n i = m_1, m_2, m_3

Comandos

n CONTINUE

onde:

- **n** é o número do comando que delimita a abrangência da iteração;
- **i** é a variável inteira a qual o DO se refere. Funciona como um índice.
- **m_1** é o valor inicial de **i**.
- **m_2** é o valor final de **i**.
- **m_3** é o incremento dado a **i** a cada iteração.

No inicio da iteração, i recebe o valor da expressão m_1. A ação especificada pelos comandos é executada repetitivamente enquanto:

$m_1 \leq m_2$ se $m_3 > 0$

$m_1 \geq m_2$ se $m_3 < 0$

A cada iteração, a variável i é atualizada com o valor i + m_3.

A execução do laço para quando:

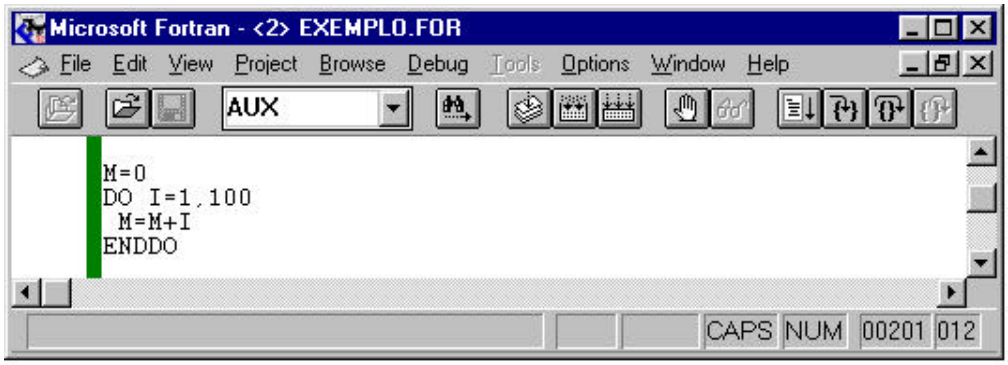
$m_1 > m_2$ se $m_3 > 0$

$m_1 < m_2$ se $m_3 < 0$

Quando o incremento m_3 é omitido, como no exemplo acima, o FORTRAN assume o valor 1.

O comando que termina o laço do *DO* pode ser qualquer comando executável da linguagem, com exceção aos comandos *GOTO*, *IF*, *END*, *RETURN* e *DO*. Preferencialmente, deve-se utilizar o comando *CONTINUE*. O *CONTINUE* é um comando executável que não realiza qualquer ação. O comando terminal da iteração do *DO* dever ser escrito após o comando *DO*, na mesma unidade de programa.

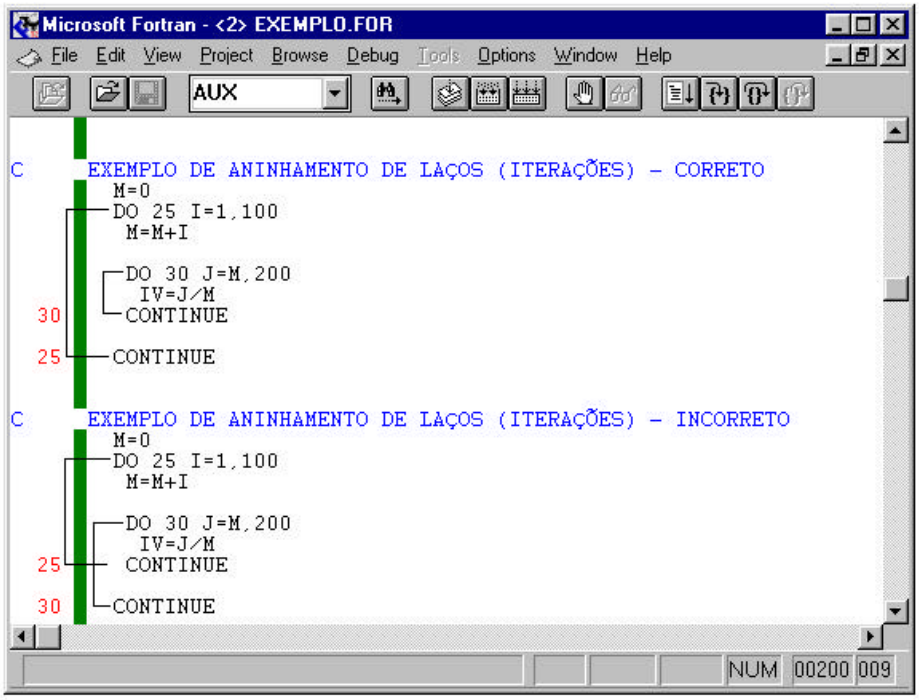
A segunda maneira de utilização do DO é sem a utilização de número de comando. Ex:



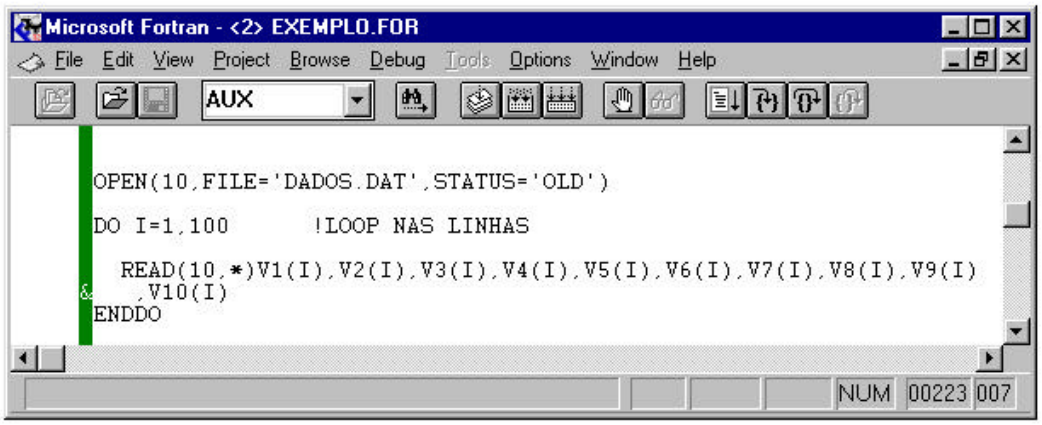
O valor da variável do DO (i) não pode ser alterado por qualquer comando dentro do laço.

Não é permitido saltar de um ponto fora do laço do DO para outro dentro dele.

Pode haver aninhamento de laços de DO, mas os laços internos devem estar contidos no laço mais externo, ou seja, não é permitido o cruzamento de laços. Ex:

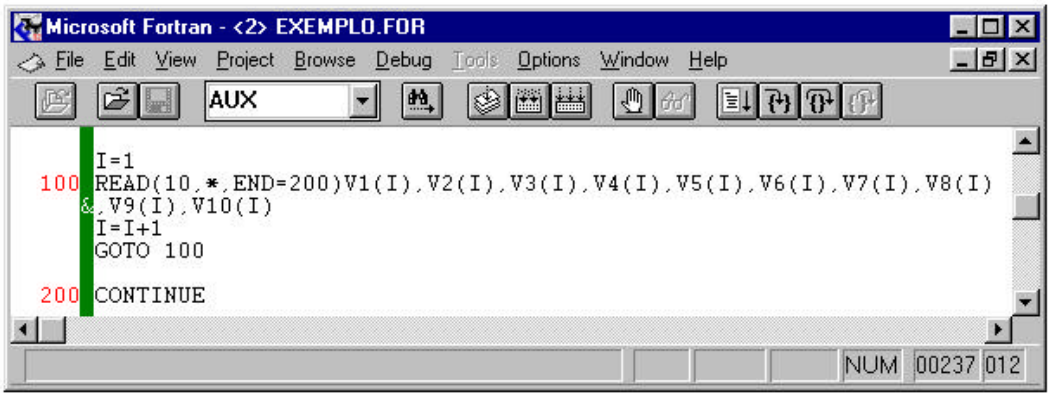


O uso do DO é de grande utilização na grande maioria dos programas, principalmente naqueles utilizados na Meteorologia que, geralmente, estão em forma de grade, conforme será visto posteriormente. Entretanto, é muito comum o fato de não serem conhecidas as dimensões dos dados com que se trabalha e a utilização do DO pode ficar comprometida. Como um exemplo, suponha que se deseja efetuar a leitura de um arquivo com 10 dados colocados lado a lado (colunas) e que estes dados estejam disponíveis para cada hora, dispostos em linhas diferentes dentro do arquivo, num total de 100 linhas. Uma maneira para essa leitura seria:



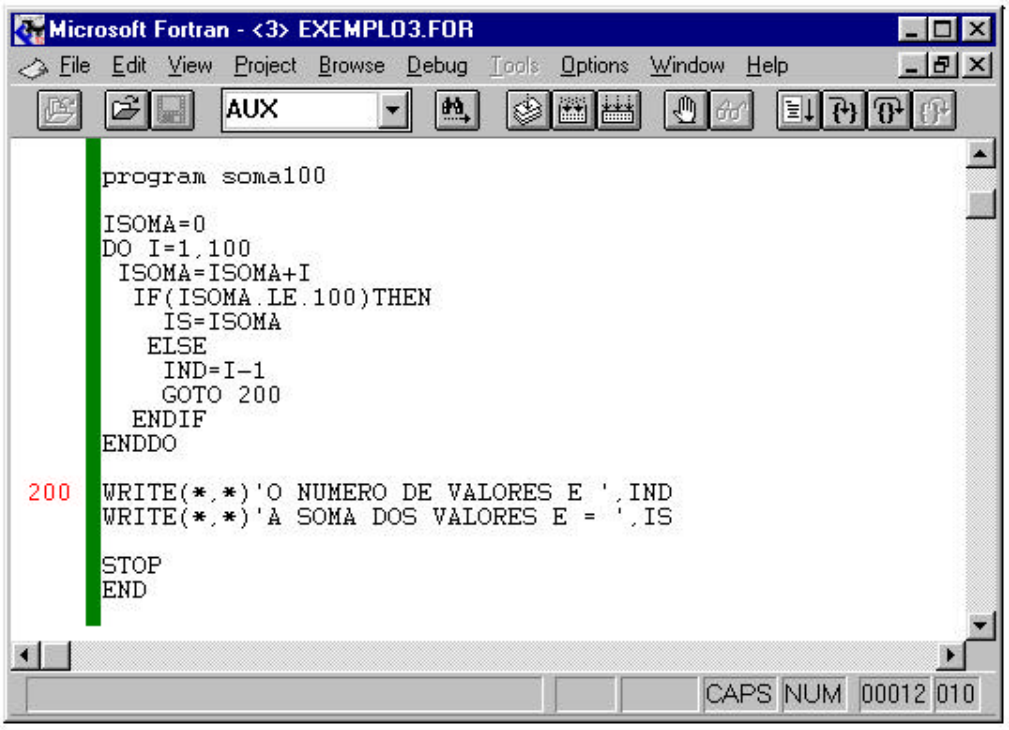
Supondo agora que o número de linhas desse arquivo é desconhecido, de que maneira essas 100 linhas poderiam ser lidas? Uma das possibilidades é a utilização de comandos que realizem saltos incondicionais para o comando cujo número é n. O comando GOTO realiza esta tarefa.

Exemplo:



A forma utilizada no exemplo acima é conhecida por “método fim-de-arquivo”.

O comando *GOTO* pode também ser utilizado dentro de laços existentes no programa, seguindo alguma condição imposta. Como um exemplo, suponha que desejamos saber quantos números são necessários para obter o maior valor inferior a 100, através da soma dos mesmos. O programa abaixo é uma solução para esta tarefa.

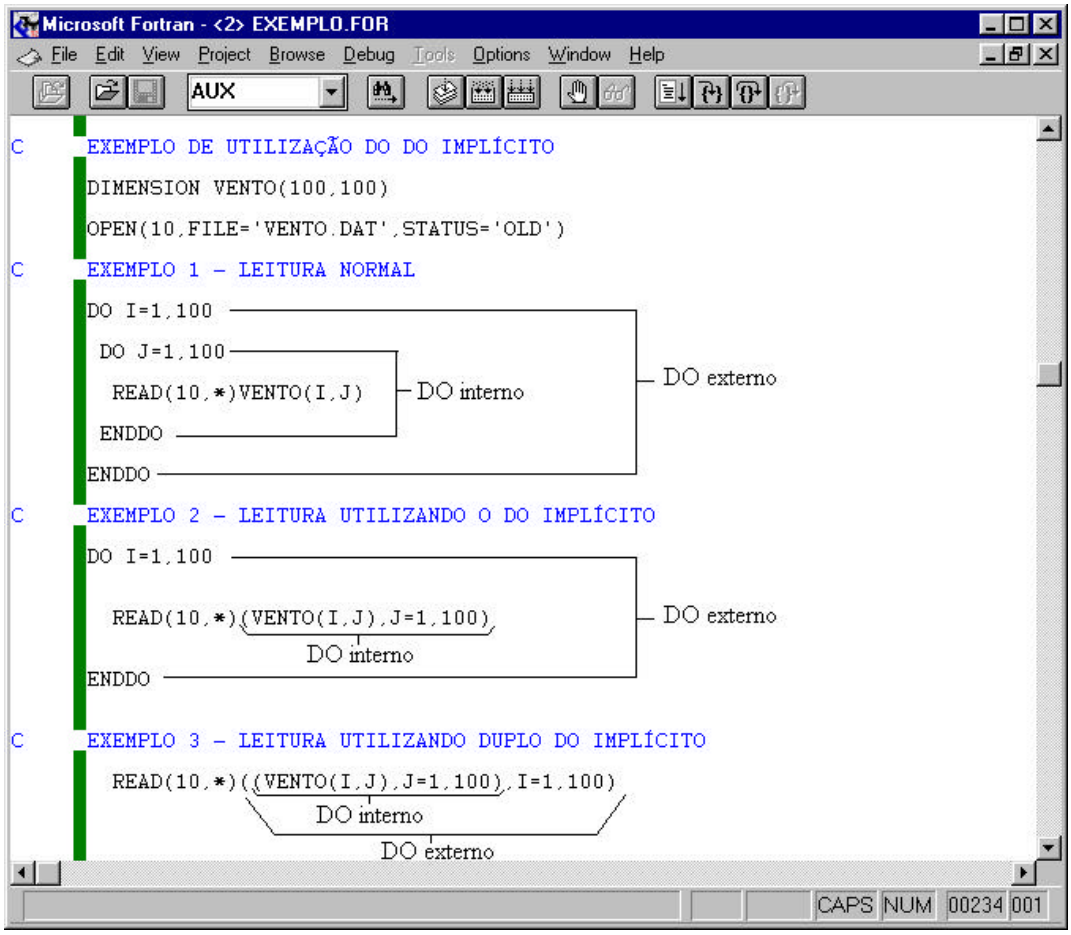


No exemplo acima, o número de valores é 13 e soma dos mesmos é 91. Caso não fosse imposta nenhuma condição, o laço só terminaria após a soma de todos os números de 1 até 100.

6.3. Entrada e saída de conjuntos – *DO Implícito.*

Uma das facilidades existentes na linguagem FORTRAN é a possibilidade da leitura e escrita de arquivos, geralmente na forma matricial, utilizando-se do *DO Implícito*. O *DO Implícito* deve estar sempre associado a um comando de entrada ou saída de dados e não pode ser utilizado durante cálculos. Por exemplo, suponha que seja necessária a leitura de uma matriz que possui dados de vento dispostos numa grade regular. Uma maneira simplificada para efetuar esta tarefa é apresentada na figura a seguir.

Nesta figura, fica clara a facilidade oferecida pela utilização do *DO Implícito*. Embora os exemplos apresentem apenas leituras, a escrita de matrizes também pode ser feita da mesma maneira.



6.4. Exercícios Práticos.

1) Faça um programa que determine valores pares e impares entre 1 e 100, através da utilização de operações lógicas. Escreva os resultados em dois arquivos distintos, sendo os dados escritos em uma única coluna. Ex (par.dat e impar.dat)

2) Faça um programa que leia os arquivos criados no exercício 1. No mesmo programa, calcule o valor médio e o desvio padrão para os dois arquivos. Em seguida, determine a variância para cada valor e escreva os resultados em arquivos separados de maneira tal que os resultados possam ser "plotados" num editor gráfico (Origin ou Excel).

3) Utilizando-se do recurso de DO Implícito, faça um programa que execute a mesma tarefa do exercício 1 sem a realização de qualquer cálculo.

7. Subprogramas.

Em muitas ocasiões, a construção de um programa simples acaba tornando o mesmo muito extenso e, às vezes, muito caro computacionalmente. A construção de um programa mais complexo, embora exija um maior tempo e muito mais raciocínio, é de grande utilidade na programação estruturada. Muitas vezes é necessária a criação de módulos independentes no programa para a realização de determinadas tarefas. A modularização de um programa é realizada através da utilização de *subprogramas*.

Um subprograma é uma unidade de programa independente, relativamente simples, que realiza funções específicas e que possui um identificador. Uma *unidade de programa* é uma seqüência de comandos terminada pelo comando *END.*

Existem dois tipos básicos de subprogramas utilizados na linguagem FORTRAN: as funções e as sub-rotinas.

Em qualquer um dos casos, o uso de subprogramas afeta o fluxo de controle de um programa. Com a chamada ao subprograma, o controle passa aos comandos do subprograma. Após a execução dos comandos deste subprograma, o controle retorna ao ponto de chamada. Um mesmo subprograma pode ser chamado várias vezes, reduzindo o número de comandos existentes no programa principal. Esse é um dos grandes atrativos da utilização de subprogramas.

7.1. Funções.

Existem três classes de funções na linguagem FORTRAN:

- Funções Intrínsecas (vistas na seção 2.6);
- Funções de Comando;
- Subprogramas FUNCTION.

As funções realizam um processamento específico com base em parâmetros estabelecidos, retornando às unidades de programa que as utilizam um único valor que é o resultado deste processamento. Analogamente ao que ocorre com as funções matemáticas, este resultado depende dos valores atribuídos aos parâmetros.

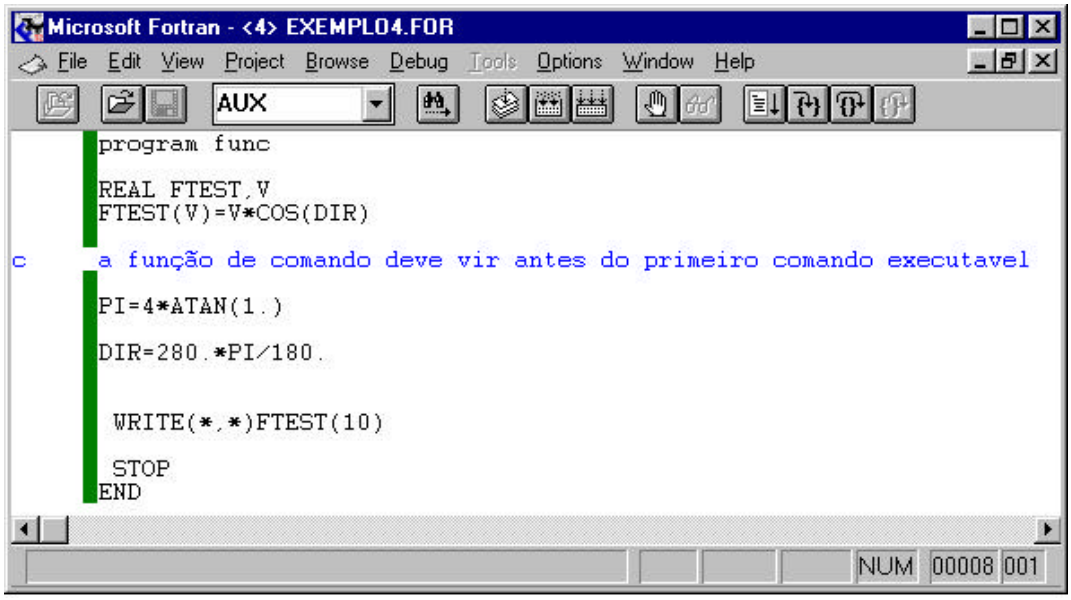
7.1.1. Funções de Comando.

A função de Comando é um tipo de função que é definida pelo próprio programador. É definida num único comando, através de uma declaração. Só pode ser utilizada na unidade de programa que a define. Internamente a uma unidade de programa, uma função de comando deve aparecer após os comandos de especificação e

antes do primeiro comando executável na unidade de programa. A referência a este tipo de função é feita da mesma forma que para as funções intrínsecas. Sua forma geral é:

<nome da função> (parâmetros ou variáveis simples envolvidas)=<expressão>

Exemplo:



7.1.2. Subprograma *FUNCTION.*

Assim como para a função de comando, esta espécie de função é também definida pelo usuário e pode compreender mais de um comando da linguagem FORTRAN.

Um subprograma função *FUNCTION* é usado do mesmo modo que uma função intrínseca.

O subprograma *FUNCTION* sempre começa com o comando não-executável *FUNCTION*, seguido por uma seqüência de comandos que realiza um procedimento de computação e termina com um comando *END*. O controle de execução é retornado para a unidade de programa que o chama quando é encontrado um comando *RETURN* ou um comando *END*.

Existem algumas regras que devem ser seguidas quando da utilização de um subprograma *FUNCTION*. Os argumentos mudos do subprograma devem concordar em ordem de apresentação, quantidade e tipo com os argumentos atuais usados na unidade de programa que o chama. Se um dos argumentos mudos é um conjunto, o número de seus elementos deve, geralmente, ser igual ao número de elementos do correspondente

conjunto no programa que o chama. O único valor retornado ao programa que o chama está armazenado no nome da função.

Um subprograma *FUNCTION* é, normalmente, colocado após o programa principal, porém ele pode ser colocado antes dele. Em qualquer caso, ele é compilado como um programa separado. Havendo mais de um subprograma *FUNCTION*, a ordem entre eles pode ser qualquer.

O comando *FUNCTION* deve aparecer como o primeiro comando num subprograma *FUNCTION*. Este comando especifica o nome simbólico do subprograma, que é usado como o ponto principal de entrada no subprograma. Especifica também a lista de argumentos mudos.

Sua forma geral é:

[tipo] *FUNCTION* nome ([a[,a]...])

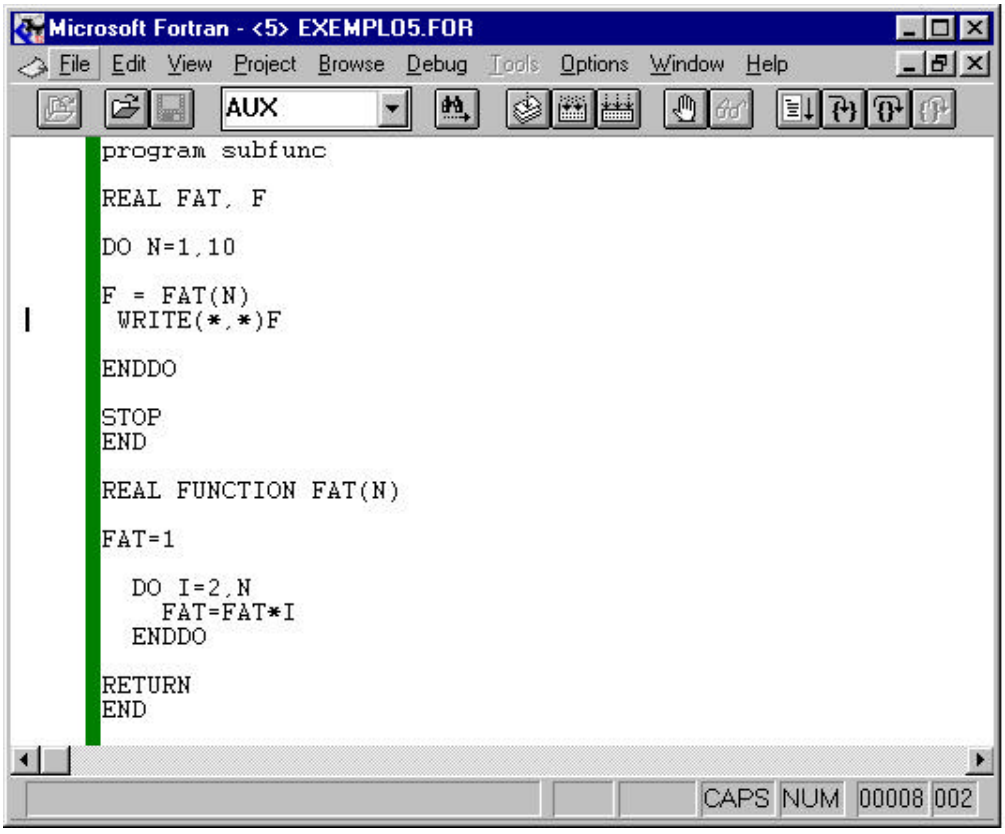
sendo:

- tipo: pode ser *INTEGER*, *REAL*, *CHARACTER*, que estão sendo tratadas nesse curso, além de *DOUBLE PRECISION*, *COMPLEX*, *BOOLEAN* e *LOGICAL*;
- nome: é o nome do subprograma *FUNCTION*, seguindo as mesmas regras de nome de variáveis;
- a: é um argumento mudo, que pode ser um nome de variável, nome de conjunto, ou nome de procedimento mudo.

Se uma *FUNCTION* é uma função do tipo *CHARACTER*, então o seu comprimento em número de caracteres deve ser o mesmo, tanto no subprograma como no programa que o utiliza. Esta concordância de comprimento sempre existe quando é utilizado o comprimento (*) no subprograma para especificar o comprimento da função.

Argumentos mudos que representam nomes de conjuntos devem ser dimensionados por um comando *DIMENSION* ou por um comando de declaração de tipo.

Exemplo:



No exemplo apresentado na figura anterior, a *FUNCTION* FAT(N) calcula o fatorial de um número N qualquer, neste exemplo de 1 até 10.

7.2. Sub-rotinas

As *sub-rotinas* constituem o outro tipo fundamental de subprograma permitido na linguagem FORTRAN. Existem algumas diferenças básicas entre *funções* e *sub-rotinas.* A primeira delas é que a *sub-rotina* não retorna necessariamente apenas um valor, como no caso das *funções*. Nenhum valor é associado ao nome da sub-rotina. A *sub-rotina* pode retornar nenhum, um ou vários valores através de seus argumentos. A segunda diferença é que a sub-rotina é chamada através de um comando específico, o comando *CALL*.

A forma geral de uma sob-rotina é:

SUBROUTINE <nome> (a_1, a_2,...., a_n)

Declarações

Comandos Executáveis

RETURN

END

Sendo <nome> o nome do subprograma, a_i os parâmetros (formais), que podem ser nomes de variáveis simples ou de conjuntos. A lista de parâmetros pode ser inteiramente omitida, ou seja, a sub-rotina pode ter nenhum parâmetro.

7.2.1. O Comando *CALL*.

O comando *CALL* permite referenciar uma sub-rotina a partir de uma outra unidade de programa. Além da chamada à sub-rotinas, o comando *CALL* pode ser utilizado para a chamada de outros programas disponíveis no computador, externos ao FORTRAN, e utilizar os resultados dos mesmos. Esta chamada é feita através dos comandos *CALL SYSTEM* (em ambiente Unix ou Linux) e *CALL SYSTEMQQ* (em ambiente windows/DOS) Como um exemplo, é possível utilizar-se de recursos do sistema operacional, tais como o comando copy (cp), ren (mv), dir (ls), entre outros. Um outro exemplo, bastante comum, é a utilização de resultados gerados pelo FORTRAN para a visualização do software GrADS. Com a utilização do comando *CALL* é possível executar tal visualização fazendo a chamada do GrADS dentro do programa que esta gerando o resultado. Posteriormente, serão mostrados vários exemplos deste tipo de utilização bem como algumas características necessárias na formatação dos dados a serem utilizados no GrADS.

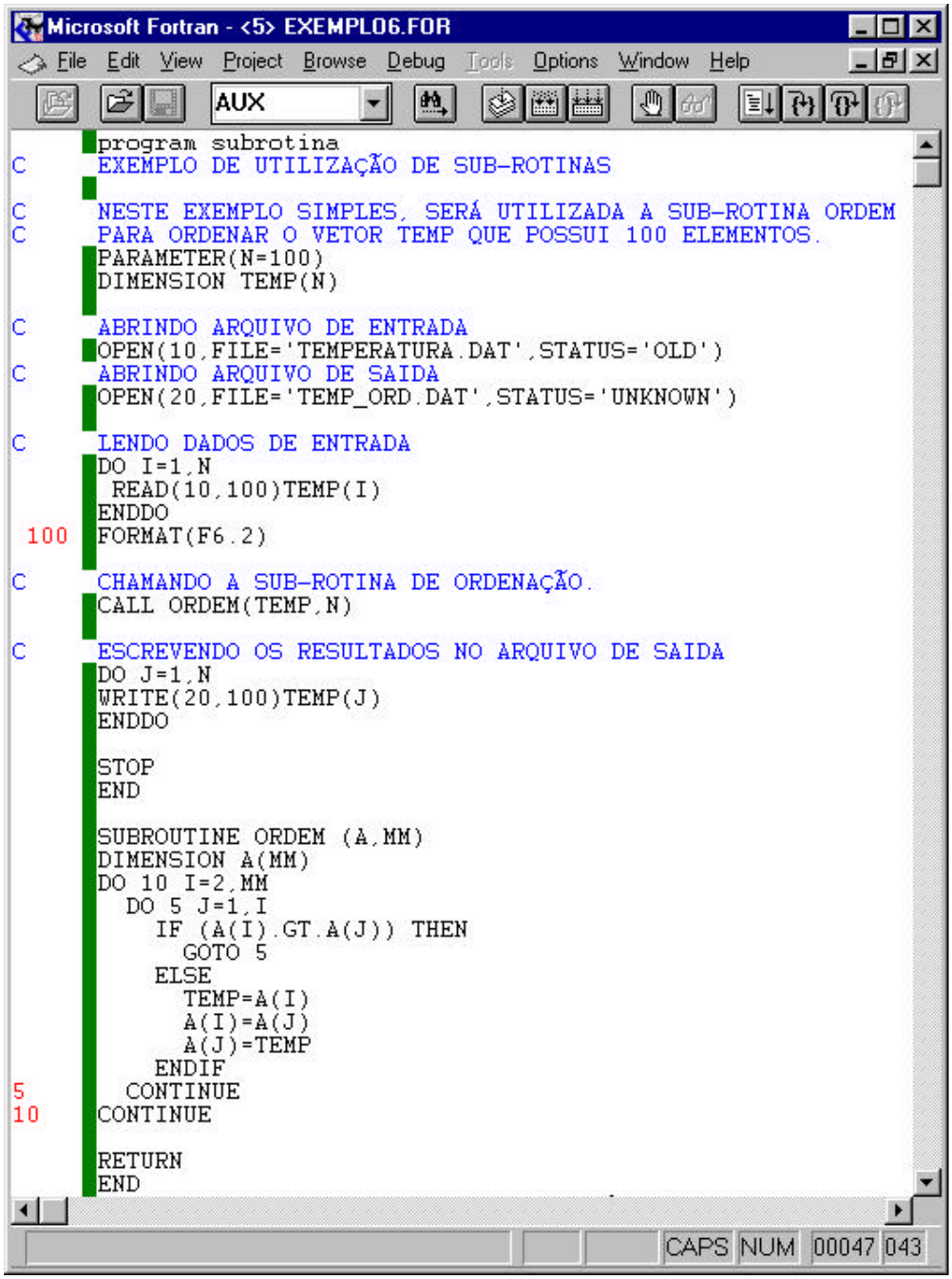
A forma geral do comando *CALL* é a seguinte:

CALL <nome> (arg_1, arg_2,..., arg_n)

sendo <nome> o nome da sub-rotina ou comando de chamada à função externa (*SYSTEM* no UNIX e *SYSTEMQQ* no windows), arg_i os argumentos da sub-rotina ou comando externo.

O comando *CALL* transfere o controle para o subprograma referenciado e substitui os parâmetros pelos argumentos correspondentes. No final da execução do subprograma, através da utilização do comando *RETURN*, o controle é devolvido para o comando imediatamente abaixo ao comando *CALL*.

Na figuras a seguir, é apresentado um exemplo de utilização do comando *CALL* para chamadas de sub-rotinas

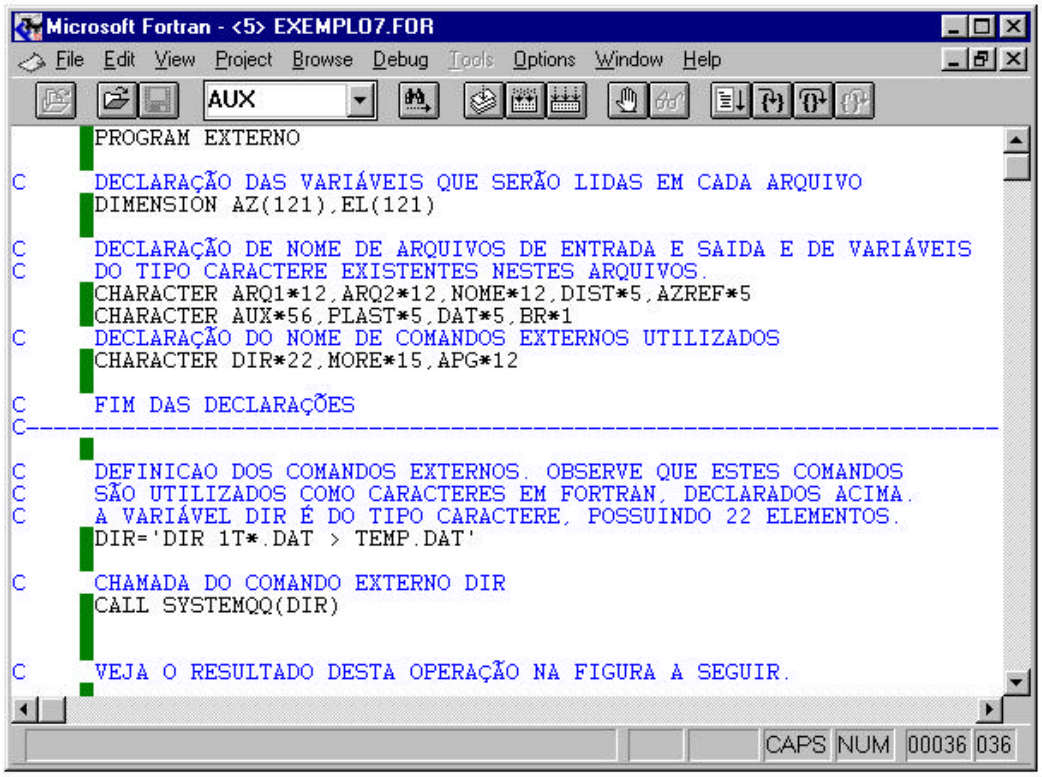



```
      program subrotina
C     EXEMPLO DE UTILIZAÇÃO DE SUB-ROTINAS

C     NESTE EXEMPLO SIMPLES, SERÁ UTILIZADA A SUB-ROTINA ORDEM
C     PARA ORDENAR O VETOR TEMP QUE POSSUI 100 ELEMENTOS.
      PARAMETER(N=100)
      DIMENSION TEMP(N)

C     ABRINDO ARQUIVO DE ENTRADA
      OPEN(10,FILE='TEMPERATURA.DAT',STATUS='OLD')
C     ABRINDO ARQUIVO DE SAIDA
      OPEN(20,FILE='TEMP_ORD.DAT',STATUS='UNKNOWN')

C     LENDO DADOS DE ENTRADA
      DO I=1,N
       READ(10,100)TEMP(I)
      ENDDO
 100  FORMAT(F6.2)

C     CHAMANDO A SUB-ROTINA DE ORDENAÇÃO.
      CALL ORDEM(TEMP,N)

C     ESCREVENDO OS RESULTADOS NO ARQUIVO DE SAIDA
      DO J=1,N
      WRITE(20,100)TEMP(J)
      ENDDO

      STOP
      END

      SUBROUTINE ORDEM (A,MM)
      DIMENSION A(MM)
      DO 10 I=2,MM
        DO 5 J=1,I
          IF (A(I).GT.A(J)) THEN
            GOTO 5
          ELSE
            TEMP=A(I)
            A(I)=A(J)
            A(J)=TEMP
          ENDIF
5       CONTINUE
10    CONTINUE

      RETURN
      END
```

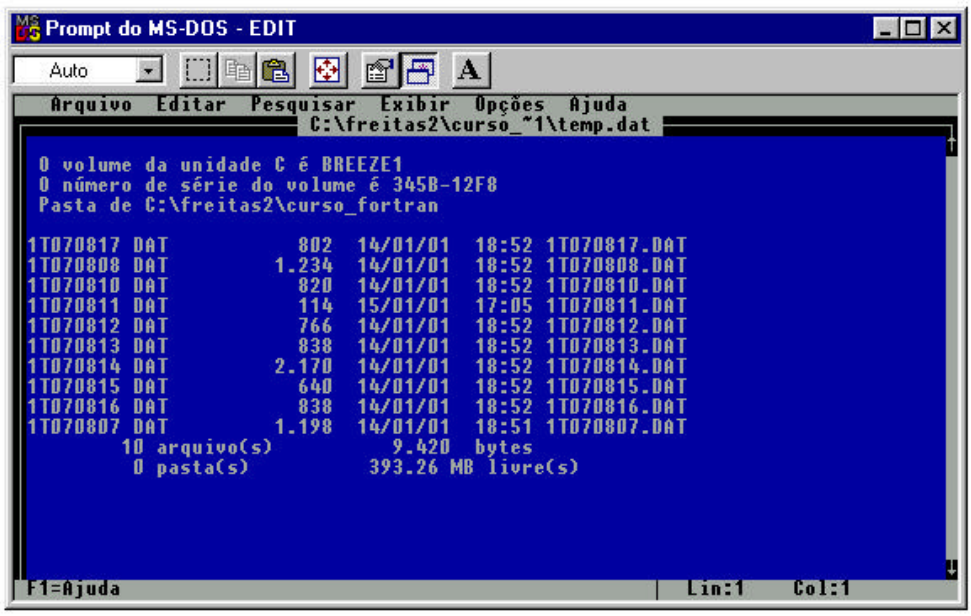
No exemplo acima, podemos observar que o nome da variável TEMP é modificado dentro da sub-rotina ORDEM, ou seja, o nome da variável dentro da sub-rotina pode ser qualquer. Entretanto, as dimensões utilizadas, quando necessário, e a ordem das variáveis e parâmetros, devem ser mantidas. Posteriormente, serão apresentados vários exemplos de utilização de sub-rotinas e essas condições ficarão mais claras.

Agora, vamos supor que seja necessária a abertura de vários arquivos, de formatos semelhantes, para a realização de alguma operação. Supondo que, a priori, não se conhece o número de arquivos que serão abertos, como é possível a realização desta tarefa? A utilização dos comandos *CALL SYSTEM* ou *CALL SYSTEMQQ* é uma das respostas para o problema.

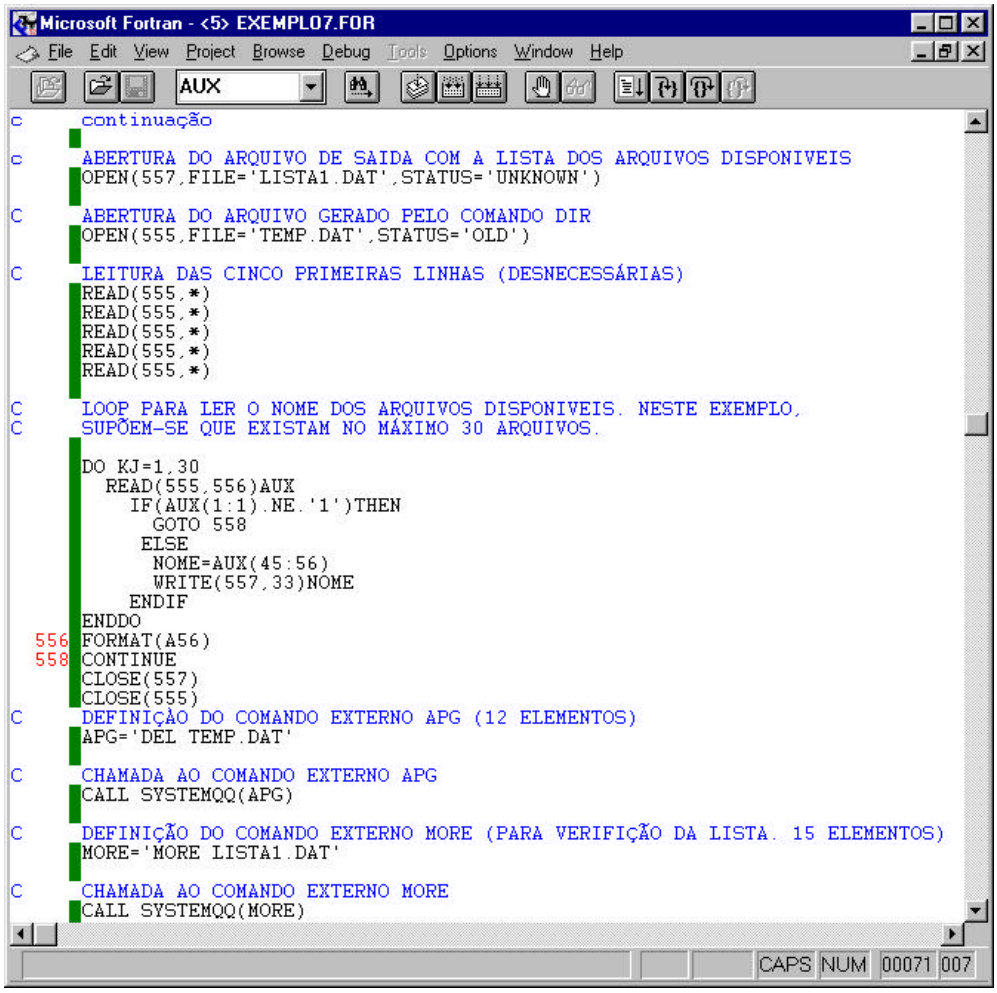
A figura a seguir é um exemplo de solução para este problema. Deseja-se saber quais os arquivos disponíveis para a operação. Isto pode ser feito através do comando DIR (no ambiente DOS/Windows) ou do comando ls (em ambiente Unix ou Linux).



O resultado do trecho de programa apresentado acima, pode ser visto na figura a seguir. Note que existem várias informações desnecessárias no arquivo TEMP.DAT. A continuação do programa acima realiza uma "limpeza" neste arquivo e extrai as informações necessárias. Note também que o número máximo de caracteres em cada linha é 56.



Continuação do programa:



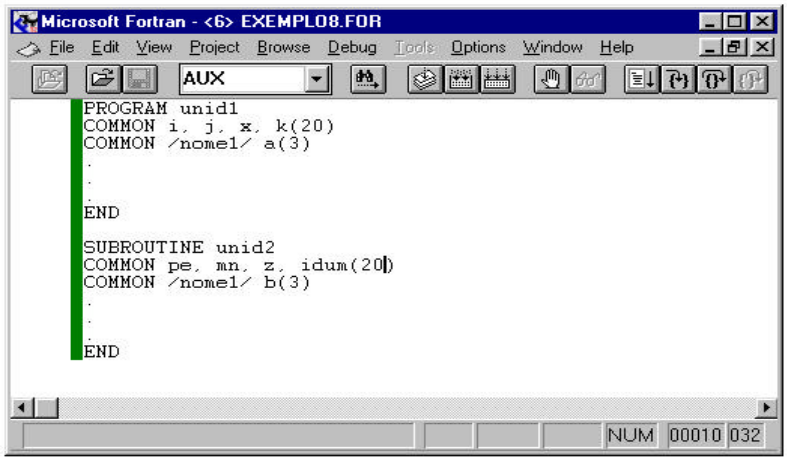
No exemplo acima foram lidas as cinco primeiras linhas como sendo linhas em branco pois a informação nelas contidas é desnecessária. Em seguida, realizou-se a leitura das outras linhas do arquivo (máximo de 30) cujo identificador da utilidade da linha é o primeiro caractere que deve ser igual a 1. Se isso acontecer, o programa utiliza os caracteres de 45 a 56 da variável de entrada AUX, para formar o nome dos arquivos de interesse e posteriormente escreve-los no arquivo LISTA1.DAT. Caso contrário, o programa utiliza o recurso *GOTO* para sair do laço do *DO* (indicação de comando 558, *CONTINUE*). Em seguida, são fechados os dois arquivos. Como o arquivo TEMP.DAT não é mais necessário, foi utilizado um comando externo (APG) para apagar este arquivo após a extração das informações necessárias. Finalmente, foi utilizado o comando externo *MORE* para a visualização do resultado.

7.3. O comando *COMMON*

Quando se trabalha com várias unidades de programa, como é o caso para as sub-rotinas, é feita uma declaração de variáveis em cada uma delas. Este procedimento pode ocupar uma grande quantidade de memória e, algumas vezes, dependendo da capacidade do computador em que o programa está sendo rodado, pode haver um estouro de memória. Para contornar esse problema é utilizado o comando *COMMON*. O comando *COMMON* é utilizado como uma declaração para fazer o compartilhamento de memória entre duas ou mais unidades de programa, e para especificar os nomes das variáveis que devem ocupar esta área comum. Consegue-se assim um meio alternativo de comunicação entre unidades de programa (através da utilização de variáveis globais). Deve ser lembrado que a única forma de comunicação entre unidades de programa utilizada até aqui, foi feita pela correspondência argumento-parâmetro. Esta forma é a mais adequada e deve ser utilizada preferencialmente.

A área de *COMMON* pode ser entendida como uma seqüência de posições na memória principal. A declaração *COMMON* permite indicar variáveis específicas a serem mantidas nesta área de armazenamento comum e especificar a ordem em que são armazenadas.

Na figura a seguir é apresentado um exemplo de utilização da declaração COMMON.

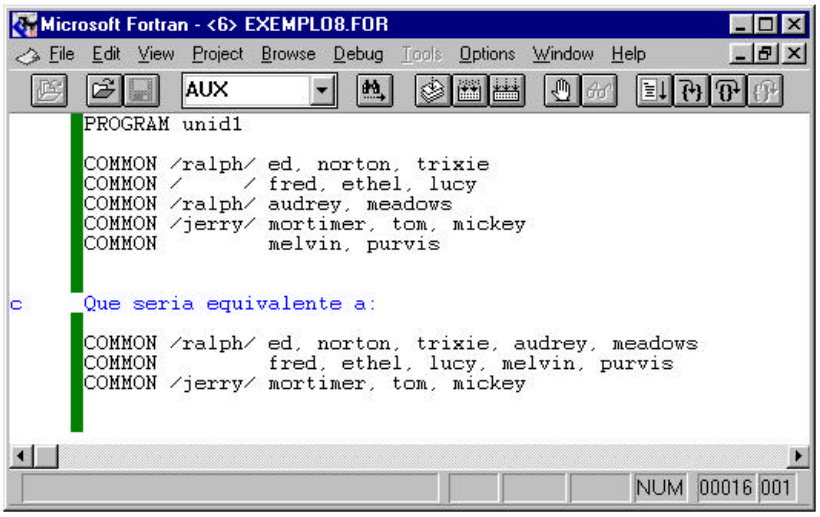


No exemplo acima, as variáveis pe, mn, z e idum(20) ocupam a mesma posição na memória que as variáveis i, j, x, k(20), respectivamente. Vale notar que as variáveis k(20) e idum(20) foram dimensionados com a declaração *COMMON*. Quando isto acontece, a declaração *DIMENSION* não deve ser utilizada. No *COMMON* não devem aparecer dimensões ajustáveis. Outra observação é que o compartilhamento de memória para essas duas variáveis é feito para cada par de elementos: idum(1) com k(1), idum(2) com k(2), ..., idum(20) com k(20).

Com a utilização da declaração *COMMON* no exemplo acima, foi possível fazer a metade da alocação de memória que seria feita sem essa declaração.

Se a declaração COMMON aparecer numa unidade de programa, ela deve aparecer em pelo menos um outra.

Numa unidade de programa podem aparecer vários comandos COMMON. Neste caso, eles têm efeito cumulativo. Veja o exemplo abaixo.

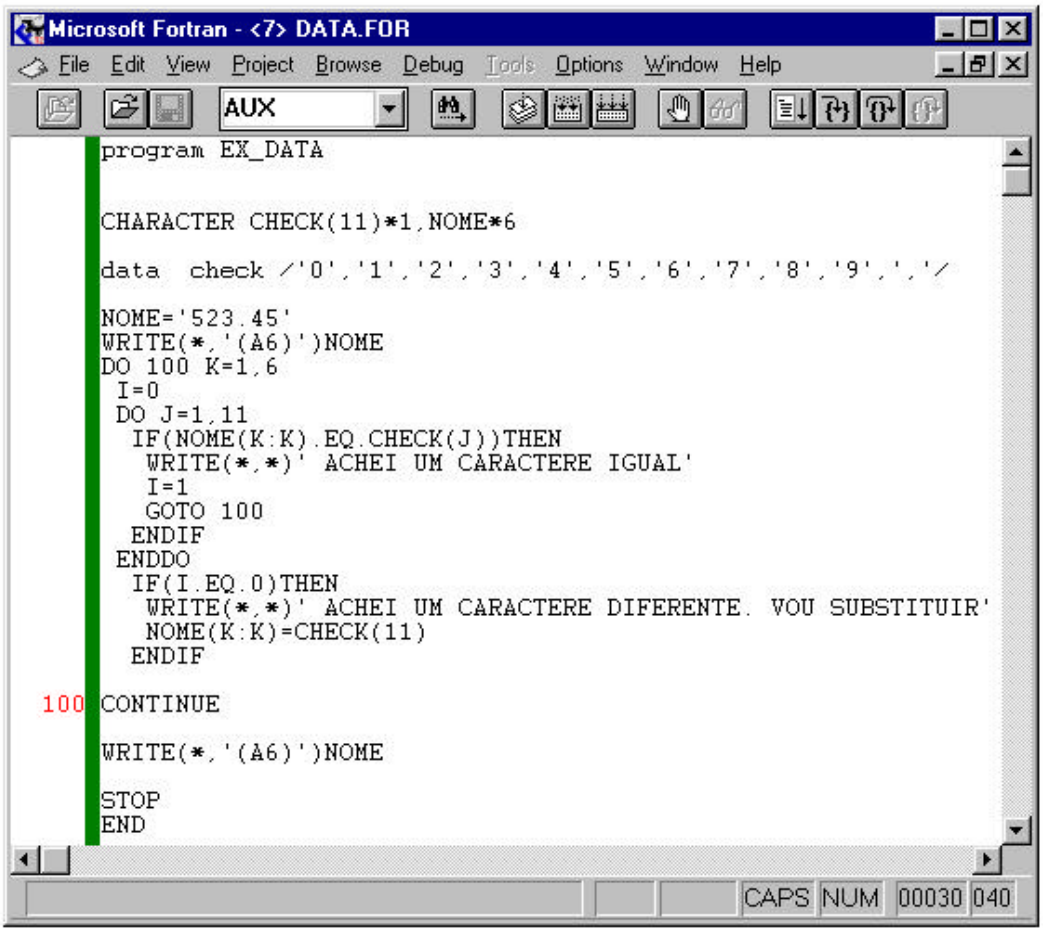


As variáveis em COMMON, de diferentes unidades de programa, devem corresponder em tipo e comprimento. É usual que elas tenham o mesmo nome, mas não é obrigatório. Um bloco comum que contenha variáveis ou conjuntos do tipo caractere só pode conter variáveis ou conjuntos deste tipo.

7.4. Tópico adicional - O comando *DATA*.

O comando *DATA* permite definir valores iniciais de variáveis.

Exemplo:



No exemplo acima, foi utilizado o comando *DATA* para definir os elementos da variável CHECK que possui 11 elementos e é do tipo caractere. Este programa faz a substituição de ponto por virgula na variável, também do tipo caractere, NOME. Este procedimento é bastante útil quando utilizamos os dados de um programa que exijam a virgula em lugar de ponto em números reais.

O comando DATA é um comando não executável e deve vir após todos os comandos de declaração e, preferencialmente, antes do primeiro comando executável. As constantes devem concordar em tipo com as variáveis correspondentes da lista. Deve

haver uma correspondência um a um entre o número total de variáveis na lista e o número total de constantes em um comando DATA.

7.5. Exercícios práticos

8. Formato de dados utilizados no GrADS.

O GrADS (the **Gr**id **A**nalysis and **D**isplay **S**ystem) é uma ferramenta interativa utilizado na análise e exibição de dados em vários campos da ciência, especialmente em meteorologia. Pode ser utilizado em vários tipos de plataforma em diferentes ambientes (Unix, Linux, Windows e DOS, entre outros.) além de ser gratuitamente distribuído na internet (http://grads.iges.org/grads/). O GrADS trabalha com dados de modelos em 4 dimensões, geralmente latitude, longitude, níveis verticais e tempo. Cada conjunto de dados está localizado dentro de um espaço 4-D através do uso de um arquivo descritor de dados. Dados de estações também podem ser descritos. A representação de dados internos em um arquivo pode ser feita através dos formatos binário e Grib. Apesar da existência de um grande conjunto de funções embutidas no GrADS, algumas vezes faz-se necessária a utilização de programas FORTRAN para criação e tratamento dos dados. Neste caso, o GrADS seria utilizado apenas para visualização.

A inclusão deste tópico no curso, tem como objetivo principal habilitar os usuários da linguagem FORTRAN na criação e leitura de arquivos em conjunto com o GrADS. Em algumas disciplinas oferecidas no DCA esse conhecimento é de grande importância e, certamente, trará muitas facilidades na realização de problemas onde é necessário o posicionamento geográfico de informações tais como grades regulares e dados de estações.

8.1. O arquivo descritor de dados no GrADS.

Para a utilização do GrADS, é necessária, primeiramente, a construção de um arquivo descritor. Este arquivo descritor (CTL) informa ao GrADS de que forma os dados estão dispostos no arquivo binário/GRIB.

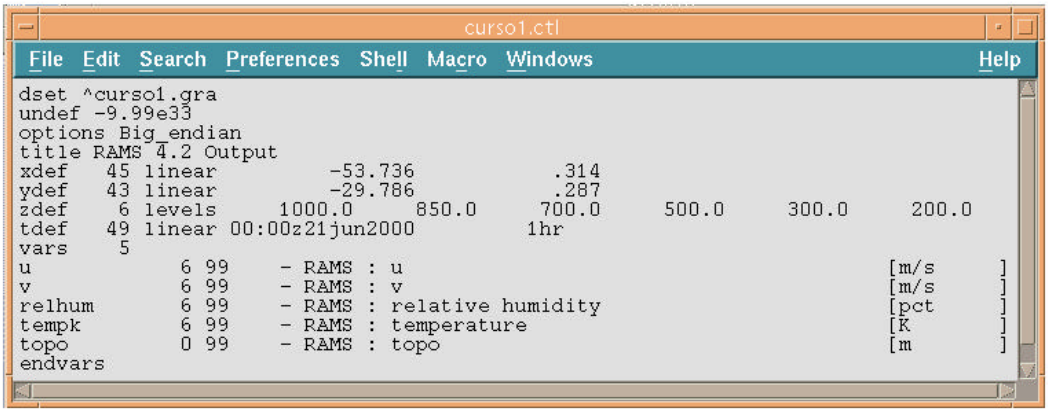
Os dados em pontos de grade podem conter qualquer número de variáveis em longitude, latitude, níveis verticais e tempo específicos. Latitudes podem variar de norte para sul ou de sul para norte e os níveis verticais podem variar de cima para baixo ou de baixo para cima.

O arquivo descritor de dados tem formato livre e pode ser criado em qualquer editor de textos e, como um dos nossos objetivos, pelo próprio FORTRAN. Podem ser utilizados comentários através da utilização do caractere (*) na primeira coluna. Entretanto, eles não podem aparecer na lista de registros de variáveis (vars – endvars).

Os registros não podem ser mais longos que 255 caracteres.

Existe uma ordem que deve ser obedecida na criação ou leitura de um arquivo binário. Esta ordem é também informada pelo arquivo descritor.

A figura abaixo apresenta um exemplo de arquivo descritor.



No exemplo acima é apresentado o arquivo “curso1.ctl”. Descrição:

- dset : indica o nome do arquivo binário que será aberto pelo GrADS, neste caso o arquivo “curso1.gra”. A utilização do caractere ^, informa ao GrADS que o arquivo binário está no diretório corrente. Caso contrário, este caractere deve ser substituído pelo caminho do arquivo;
- undef: indica qual o valor utilizado para localizações onde não existem dados, ou seja, dados indefinidos;
- options: informa ao GrADS o tipo de plataforma em que o arquivo binário foi criado e qual a disposição dos bytes (invertido ou não). Neste caso, a informação Big_endian, refere-se à um arquivo criado numa plataforma de 32 bits (IBM, SUN, SILICON, etc). Esta informação é importante quando é utilizada uma plataforma diferente daquela em que o arquivo foi criado, por exemplo, se o arquivo acima fosse utilizado no Linux (32 bits – little_endian; bytes invertidos).
- title: nome do experimento, simulação, previsão, etc., a que se refere o arquivo;
- xdef: informa ao GrADS como os dados estão dispostos em longitude (número de pontos, se a distribuição é linear ou não, longitude do ponto SW, espaçamento entre os pontos de grade).
- ydef: informa ao GrADS como os dados estão dispostos em latitude (número de pontos, se a distribuição é linear ou não, latitude do ponto SW, espaçamento entre os pontos de grade);
- zdef: informa ao GrADS como os dados estão dispostos nos níveis verticais (número de níveis, tipo de níveis – pressão ou altura, especificação de cada nível);

- tdef: informa ao GrADS o número de tempos existentes no arquivo, se o espaçamento entre tempos é linear ou não, data/hora do primeiro tempo do arquivo, passo no tempo.
- vars: indica o número de variáveis existentes no arquivo. Também funciona como um delimitador para declaração das variáveis existentes. Deste ponto até a indicação *endvars*, não é possível fazer nenhum comentário, como dito anteriormente.
- endvars: indica o final da declaração de variáveis

Cada variável, possui a indicação do nome, número de níveis, unidades, etc. No exemplo acima, as variáveis u, v, tempk e relhum estão dispostas em 6 níveis verticais, enquanto que a variável topo, representada pelo valor 0, refere-se a um dado de superfície. Algumas vezes, é comum o aparecimento de variáveis com diferentes números de níveis, por exemplo, a umidade relativa nos dados de reanálise do NCEP possuem menor número de níveis que vento, temperatura, etc.

O segundo valor após cada nome de variável (99) é utilizado para dados GRIB especiais e não serão discutidos neste curso (utilize sempre 99 para arquivos binários).

8.2. Seqüência dos dados.

Como citado anteriormente, é necessário o conhecimento da seqüência de distribuição dos dados no GrADS. Principalmente quando um programa FORTRAN é utilizado para a construção ou leitura do arquivo. A disposição dos dados descritos pelo arquivo "curso1.ctl" é a seguinte:

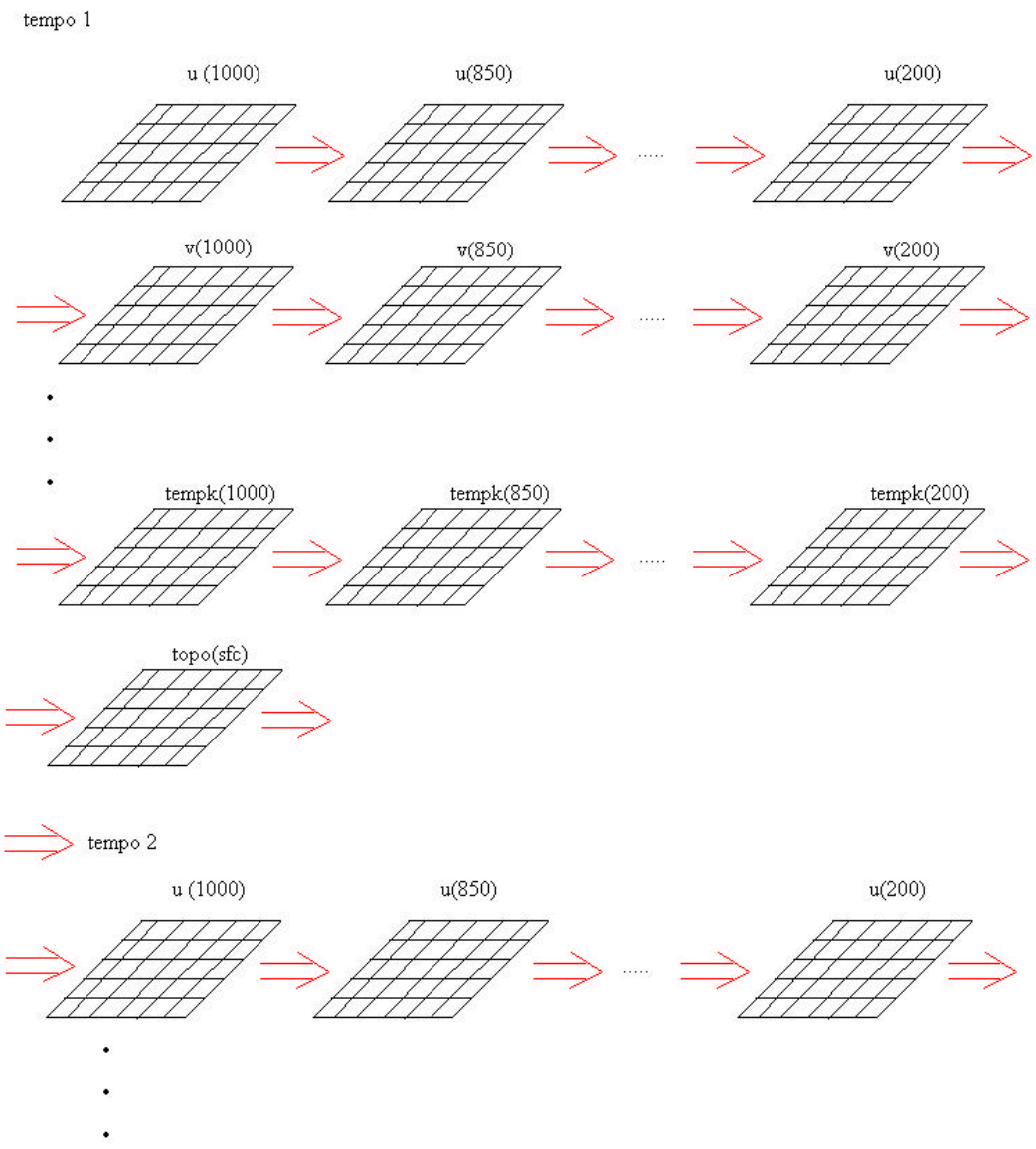
-Tempo;

-variável;

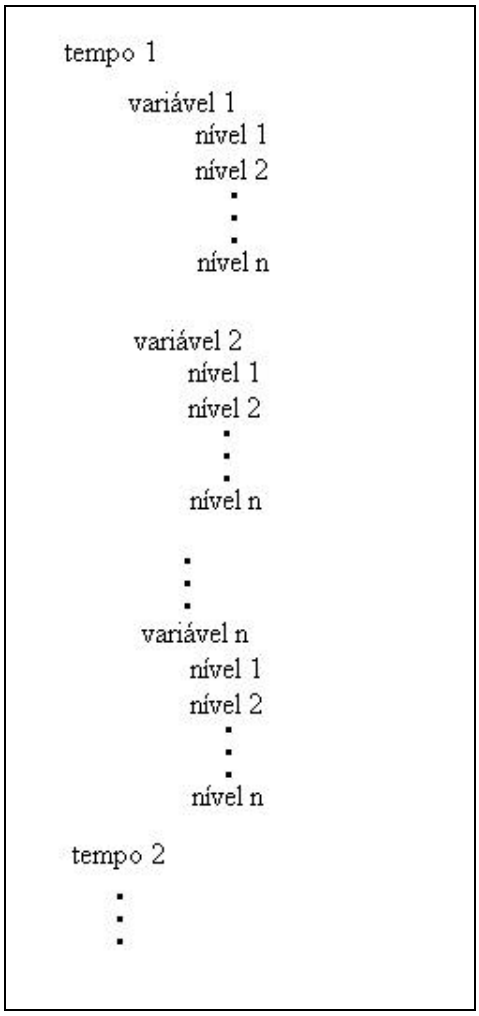
-níveis.

-coordenadas

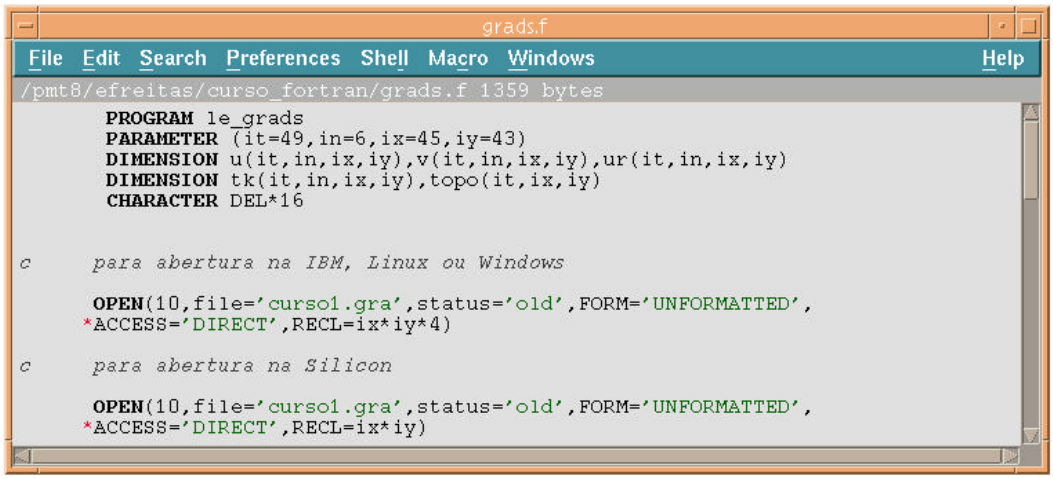
Esta seqüência é utilizada em qualquer arquivo lido pelo GrADS. A figura a seguir ilustra esta seqüência para a grade do arquivo curso1.gra.



Neste exemplo, cada variável é representada por uma grade de 45 x 43 pontos. Cada grade é colocada uma após a outra, obedecendo o esquema a seguir:



No formato binário (arquivos de acesso direto) do GrADS, cada variável é definida utilizando-se *"records"*. O tamanho do record deve ser especificado quando da utilização de um programa FORTRAN através da especificação *RECL*(tamanho do record) nos comandos *OPEN*. Geralmente, cada *"record"* ocupa 4 bytes (32 bits). É necessário portanto, saber como cada plataforma trabalha com o número de bytes de cada *"record"*. Por exemplo, nas plataformas SILICON, cada *"record"*, por default, possui 4 bytes. Na IBM, por default, cada *"record"* possui 1 byte. O exemplo abaixo dá uma amostra da abertura do arquivo curso1.gra nos dois tipos de plataforma.



No primeiro exemplo, o tamanho de cada *"record"* RECL é definido pelo produto ix (número de elementos em longitudes) $\times$ iy (número de elementos em latitude) $\times 4$, ou seja, o número total de elementos vezes 4 (por default cada informação é assumida ocupando 1 byte). No segundo exemplo o fator 4 é omitido pois este tipo de plataforma assume que cada informação (pontos de grade neste caso) ocupa 4 bytes.

Existe uma seqüência para as longitudes e latitudes também. Para cada ponto de latitude, começando por aquele mais ao sul, é feito um laço em todas as longitudes, começando por aquela mais a oeste.

Quando da leitura de um arquivo binário é necessário que todo o arquivo seja lido, ou, pelo menos, que seja lido até o *"record"* de interesse, mesmo quando se deseja trabalhar com apenas uma variável. Pode-se fazer a leitura por saltos, desde que sejam contados os *"records"* intermediários entre cada variável, entretanto, este procedimento é perigoso e deve ser evitado, principalmente quando não se tem muita prática no tratamento desse tipo de dado.

Um ponto de grande importância e que se deve tomar muito cuidado é que, no exemplo acima, foram lidos para cada tempo, nível e variável, todos os pontos de grade de uma só vez (ix*iy*4). O tamanho do *"record"* corresponde ao produto entre o número de pontos em longitudes o número de pontos em latitudes e o número de bytes ocupados por cada ponto de grade (na verdade, a informação atribuída a cada ponto). Poderia ter sido feita a leitura ponto a ponto. Neste caso, o tamanho de cada *"record"* seria de 4 bytes e não seria possível a utilização do *DO IMPLICITO* para a leitura ou escrita desses dados, devendo existir um *DO* individual para cada direção. Este aspecto ficará mais claro com o exemplo a seguir.

grads.f

File Edit Search Preferences Shell Macro Windows Help

/pmt8/efreitas/curso_fortran/grads.f 1818 bytes

```
      PROGRAM le_grads
      PARAMETER (it=49,in=6,ix=45,iy=43)
      DIMENSION u(it,in,ix,iy),v(it,in,ix,iy),ur(it,in,ix,iy)
      DIMENSION tk(it,in,ix,iy),topo(it,ix,iy)
      CHARACTER DEL*16

c O comando a seguir eh necessario quando da criacao de arquivos
c binarios. Se existir um arquivo com o mesmo nome que ocupe um maior
c numero de bytes, a informacao anterior nos bytes que nao estao sendo
c ocupados permanece, mesmo utilizando a opcao de status 'UNKNOWN'

      DEL='rm -f curso1.BIN'
      CALL SYSTEM (DEL)


c     para abertura na IBM, Linux ou Windows

      OPEN(10,file='curso1.gra',status='old',FORM='UNFORMATTED',
     *ACCESS='DIRECT',RECL=ix*iy*4)

       IREC=0
       DO I=1,IT         ! LACO NO TEMPO (TODAS AS VARIAVEIS SERAO LIDAS)

        DO J=1,IN     !LACO NOS NIVEIS (INDIVIDUAL PARA CADA VARIAVEL)

         IREC=IREC+1 !CONTAGEM DO NUMEROS DE RECORDS (NESTE CASO, PARA CADA GRADE)

C     LEITURA DA PRIMEIRA VARIAVEL

         READ(10,REC=IREC)((U(I,J,NX,NY),NX=1,45),NY=1,43)

        ENDDO    !FIM DO LOOP NOS NIVEIS PARA A VARIAVEL U (TEMPO 1)

C  CONTINUA PARA AS OUTRAS VARIAVEIS (VER PROXIMA FIGURA)
```

A figura a seguir apresenta a leitura da mesma variável sendo feito ponto a ponto.

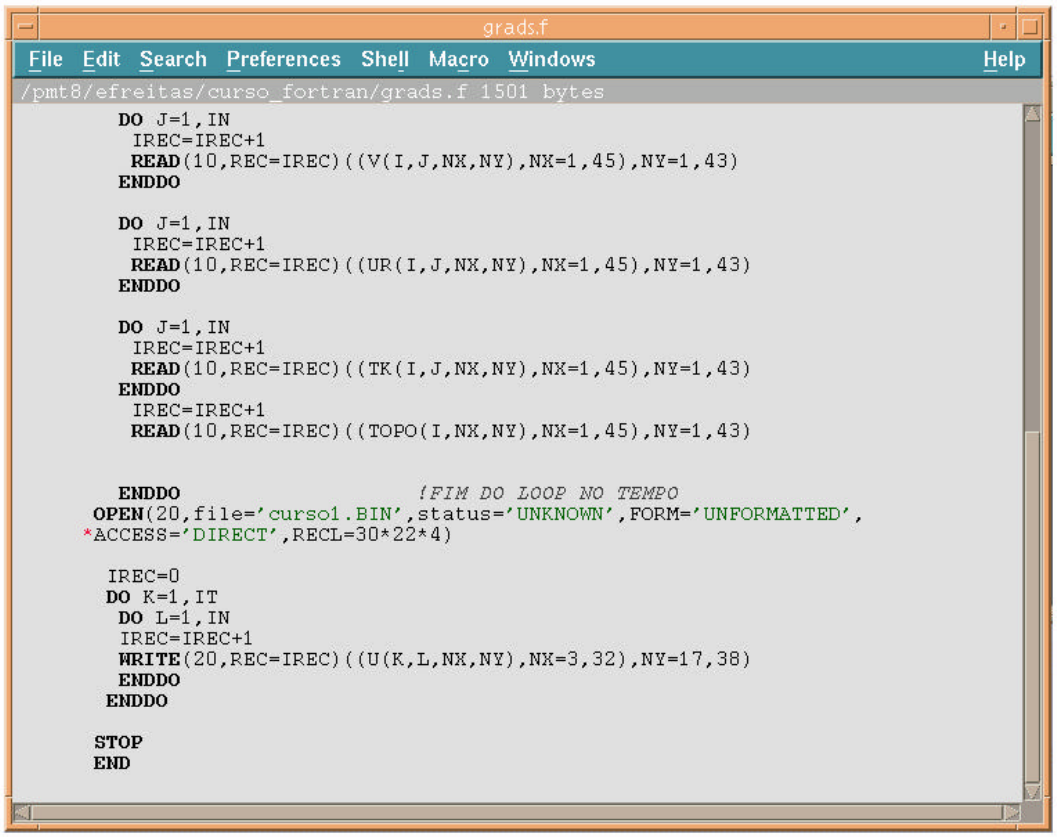
exe_grads.f

File Edit Search Preferences Shell Macro Windows Help

/pmt8/efreitas/curso_fortran/exe_grads.f 2014 bytes

```
c     para aberturs na IBM, Linux ou Windows

c  lendo cada ponto
      OPEN(10,file='curso1.gra',status='old',FORM='UNFORMATTED',
     *ACCESS='DIRECT',RECL=4)

       IREC=0
       DO I=1,IT         ! LACO NO TEMPO (TODAS AS VARIAVEIS SERAO LIDAS)

        DO J=1,IN     !LACO NOS NIVEIS (INDIVIDUAL PARA CADA VARIAVEL)

           DO NY=1,IY !LACO NAS LATITUDES

              DO NX=1,IX  !LACO NAS LONGITUDES

         IREC=IREC+1 !CONTAGEM DO NUMEROS DE RECORDS (NESTE CASO, PARA CADA PTO DE GRADE)

C     LEITURA DA PRIMEIRA VARIAVEL

          READ(10,REC=IREC) U(I,J,NX,NY)

               ENDDO      !FIM DO LOOP NAS LONGITUDES

             ENDDO        !FIM DO LOOP NAS LATITUDES

         ENDDO    !FIM DO LOOP NOS NIVEIS PARA A VARIAVEL U (TEMPO 1)
```

A figura a seguir apresenta a continuação do programa, sendo a leitura feita através da utilização do *DO IMPLICITO*, por ser este o método mais simples. Neste programa, deseja-se extrair a variável ***U*** para um outro arquivo (curso1.BIN). Note que o número de pontos nas direções X e Y foi diminuído também. Deve-se tomar cuidado durante a criação de arquivos binários. Diferente dos arquivos ASCII, quando um arquivo binário é re-escrito, se o novo arquivo for menor que o arquivo anterior (menor número de *"records"*), as informações contidas nos records seguintes permanecerão no novo arquivo. Sendo assim, por garantia, é melhor apagar o arquivo de saída sempre que o programa for rodado. No exemplo anterior (1ª figura) esta tarefa é feita automaticamente pelo próprio programa, através da chamada do comando externo DEL (através do comando *CALL*, visto anteriormente).



8.3. Exercícios Práticos.

1) Escreva o programa do exemplo dado para a leitura do arquivo "curso1.gra".
2) Crie um arquivo CTL para ler o arquivo binário "curso1.BIN" criado pelo programa do exercício anterior.

3) Refaça o programa do exercício 1 utilizando uma sub-rotina para leitura e uma para escrita das variáveis U e V, invertendo a ordem de escrita das latitudes. Crie um arquivo CTL para ver os resultados no GrADS. Veja o que acontece com a disposição dos dados. Acrescente ao CTL, na linha de comando options, o comando YREV. Verifique os resultados.

4) Acrescente ao programa uma sub-rotina que calcule a vorticidade em todos os níveis disponíveis. Faça com que o resultado seja colocado num arquivo que contenha também as variáveis U e V. O nome do arquivo deve ser criado pelo próprio programa utilizando cadeias de caracteres da linguagem FORTRAN.

5) Faça o mesmo para calcular a divergência.

6) Calcule o valor médio das variáveis U e V entre os níveis de 1000 e 500 hPa. Calcule também a vorticidade e a divergência para o valor médio. Coloque os resultados num único arquivo.

7) Utilize o recurso de comandos externos (*CALL* <comando externo>) para abrir os resultados no GrADS. Lembre-se de criar, através do mesmo programa, um arquivo CTL.

9. Referências Bibliográficas

Cereda, R. L. D. e Maldonado, J. C., 1987: Introdução ao FORTRAN 77 para microcomputadores. *McGraw-Hill*, 211 pp.

Doty, B., 1995: The Grid Analysis and Display System – GrADS. User's Guide. http://www.iges.org/grads.

Hehl, M. E., 1986: Linguagem de programação estruturada FORTRAN 77. *McGraw-Hill*, --- pp.

Setzer, V. W., Simon, I, Kowaltowski, T., 1972: Curso de FORTRAN IV básico. *Editora Edgard Blücher*, 102 pp.