

OBJETIVOS DO CAPÍTULO

- Estimar e verificar a memória computacional necessária para executar um programa
- Utilizar sub-rotina recursiva
- Controlar o fluxo de informações para dentro e para fora de sub-rotinas
- Comandos do FORTRAN: DEALLOCATE, RECURSIVE, INTENT(IN,OUT,INOUT)

Para inicializar as atividades deste capítulo, deve-se acessar o programa Fortran, no Windows, através de: **Start, Programs, Fortran PowerStation 4.0, Microsoft Developer Studio**

16.1 programa16a.f90

- 1) Objetivos do programa:
 - (a) mostrar como estimar a memória computacional necessária para executar um programa; e
 - (b) usar um novo comando do FORTRAN: DEALLOCATE.
- 2) No Fortran, seguindo o [procedimento-padrão](#), **criar um projeto** com o nome **programa16a**
- 3) No Fortran, seguindo o [procedimento-padrão](#), **criar e inserir** no projeto o programa-fonte **programa16a.f90**
- 4) Dentro do espaço de edição do Fortran, na subjanela maior, **copiar** exatamente o texto em vermelho mostrado na **Tabela 16.1**.
- 5) Comentários sobre o programa:
 - (a) O comando DEALLOCATE é usado para eliminar da memória do computador variáveis para as quais foi reservado memória por meio do comando ALLOCATE. Para usá-lo basta a palavra DEALLOCATE e, dentro de parênteses, as diversas variáveis separadas por vírgula. Não é necessário indicar novamente o tamanho das variáveis, pois isso já é conhecido do comando ALLOCATE. O comando DEALLOCATE só pode ser usado com variáveis cuja memória foi alocada com o comando ALLOCATE.
 - (b) Na linha **DEALLOCATE (A)** a variável A, do tipo inteiro, com N elementos, está sendo eliminada da memória do computador.
 - (c) A estimativa da memória necessária para executar um programa pode ser obtida através da seguinte equação:

$$Memória(kB) = \frac{4*(N_{inteiro} + N_{real\ simples}) + 8*N_{real\ dupla} + 1*(L*N_{caracter})}{1024} \quad (16.1)$$

onde $N_{inteiro}$, $N_{real\ simples}$ e $N_{real\ dupla}$ representam o número total de elementos de conjuntos ou matrizes de variáveis do tipo inteiro, real simples e real dupla, respectivamente; $N_{caracter}$ representa o número total de elementos de variáveis do tipo caracter que tem comprimento ou número de caracteres igual a L . Dentro do último parênteses devem ser incluídos os produtos de outras variáveis do tipo caracter com outros comprimentos. Os valores 4, 8 e 1 representam o número de bytes que cada um dos tipos de variáveis ocupa de memória para um único elemento. O valor 1024 é um fator usado para converter o número de bytes em kilobytes (kB).

Tabela 16.1 Programa16a.f90

```

USE PORTLIB
IMPLICIT NONE
INTEGER VER, N
INTEGER, ALLOCATABLE, DIMENSION(:) :: A
REAL*8, ALLOCATABLE, DIMENSION(:) :: B
CHARACTER(10), ALLOCATABLE, DIMENSION(:) :: C
CHARACTER(50) SAIDA, TEXTO
REAL*8 M1, M2, M3, M4
INTEGER :: UNIT = 20

WRITE(*,*) "Ver memoria em uso no primeiro comando; depois clique Enter"
READ(*,*)

VER = SYSTEM("Notepad DADOS.TXT" )

OPEN(1, file = "DADOS.TXT" )

READ(1,*) N
READ(1,*) SAIDA

CLOSE(1)

ALLOCATE ( A(N), B(N), C(N) )

WRITE(*,*) "Ver memoria apos ALLOCATE para A, B e C"
READ(*,*)

A = 1

```

```

B = 1.0D0
C = "caracteres"

WRITE(*,*) "Ver memoria apos atribuicao de valores para A, B e C"
READ(*,*)

M1 = N*4.0 / 1024
M2 = N*8.0 / 1024
M3 = N*1.0*10 / 1024
M4 = M1 + M2 + M3

OPEN(UNIT, file = SAIDA )

WRITE(UNIT,*) "Memoria estimada para A, inteiros (kiloBytes) = ", M1
WRITE(UNIT,*) "Memoria estimada para B, reais (kiloBytes) = ", M2
WRITE(UNIT,*) "Memoria estimada para C, caracteres (kiloBytes) = ", M3
WRITE(UNIT,*) "Memoria estimada para A, B e C (kiloBytes) = ", M4

CLOSE(UNIT)

TEXTO = "Notepad " // SAIDA

VER = SYSTEM( TEXTO )

DEALLOCATE ( A )

WRITE(*,*) "Ver memoria com B e C"
READ(*,*)

DEALLOCATE ( B )

WRITE(*,*) "Ver memoria com C"
READ(*,*)

DEALLOCATE ( C )

WRITE(*,*) "Ver memoria sem A, B e C"
READ(*,*)

END

```

6) Executar **Build, Compile** para compilar o programa.

- 7) Gerar o programa-executável fazendo **Build, Build**.
- 8) Antes de executar o programa, é necessário criar o arquivo de dados e inserir nele os respectivos dados. No caso do programa16a.f90, é necessário **criar o arquivo “DADOS.TXT” e inserir os dois dados que correspondem às variáveis N e SAIDA. Usar, por exemplo, os dados mostrados na Figura 16.1.**
- 9) Para ver a memória efetivamente usada pelo programa, deve-se abrir o gerenciador de tarefas ou Task Manager, executando:
 - (a) **clicar o botão do lado direito do mouse sobre a barra de tarefas do Windows**
 - (b) **clicar o botão do lado esquerdo do mouse sobre a opção “Task Manager...”**
 - (c) **clicar na opção “Processes” do menu principal do Task Manager**

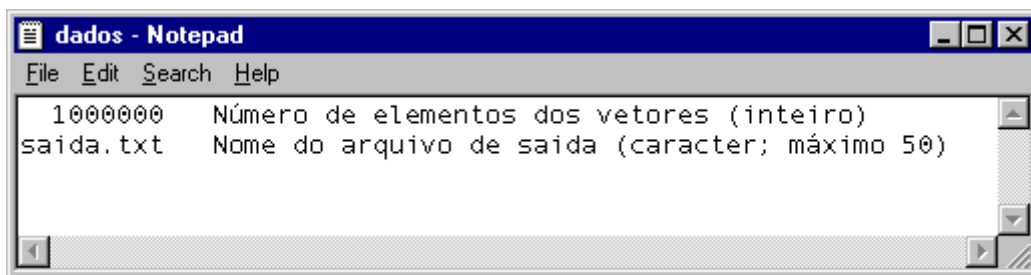


Figura 16.1 Exemplo de arquivo de dados para o programa16a.f90.

- 10) Executar o programa através de **Build, Execute**.
- 11) Conforme indica a janela DOS, **ver no Task Manager qual a memória inicial que o programa está usando**. Para isso, deve-se selecionar a janela do Task Manager e encontrar o processo chamado programa16a.exe. Na linha deste processo, na última coluna designada por “Mem Usage”, é indicada a memória em kilobytes (kB) que o programa está usando no momento. Neste exemplo, deve ser de 584 kB, como mostrado na Figura 16.2. Isso significa que, mesmo sem abrir memória para variáveis do tipo conjunto, vetores ou matrizes com o comando ALLOCATE, a execução de um programa requer uma determinada memória. Ela é necessária para guardar as instruções do programa e os outros tipos de variáveis.
- 12) **Continuar a executar o programa16a.exe, mantendo a janela do Task Manager aberta para ver a cada etapa do programa quanto de memória ele está usando**. Deve-se perceber que:
 - (a) ao ser aberto o arquivo de dados com o aplicativo Notepad, a memória aumenta para 612 kB;
 - (b) após a leitura dos dados e fechamento do aplicativo Notepad, a memória está em 648 kB;
 - (c) ao ser usado o comando ALLOCATE com as variáveis A, B e C, a memória sobe para 668 kB; isso significa que de fato não está ainda sendo usada a memória para estas variáveis, a não ser uma indicação das posições de memória que serão ocupadas;

- (d) após a atribuição de valores às variáveis A, B e C a memória sobe para 22184 kB; agora, de fato, o espaço de memória aberto com o comando ALLOCATE para as variáveis A, B e C está sendo usado;
- (e) após a eliminação da variável A, a memória cai para 18276 kB;
- (f) após a eliminação das variáveis A e B, a memória cai para 10460 kB;
- (g) após a eliminação das variáveis A, B e C, a memória cai para 676 kB;
- (h) finalmente, quando surgir na janela DOS (Figura 16.3) a frase “Press any key to continue”, significa que o programa foi encerrado e ele não ocupa mais memória alguma, como se pode ver no Task Manager.

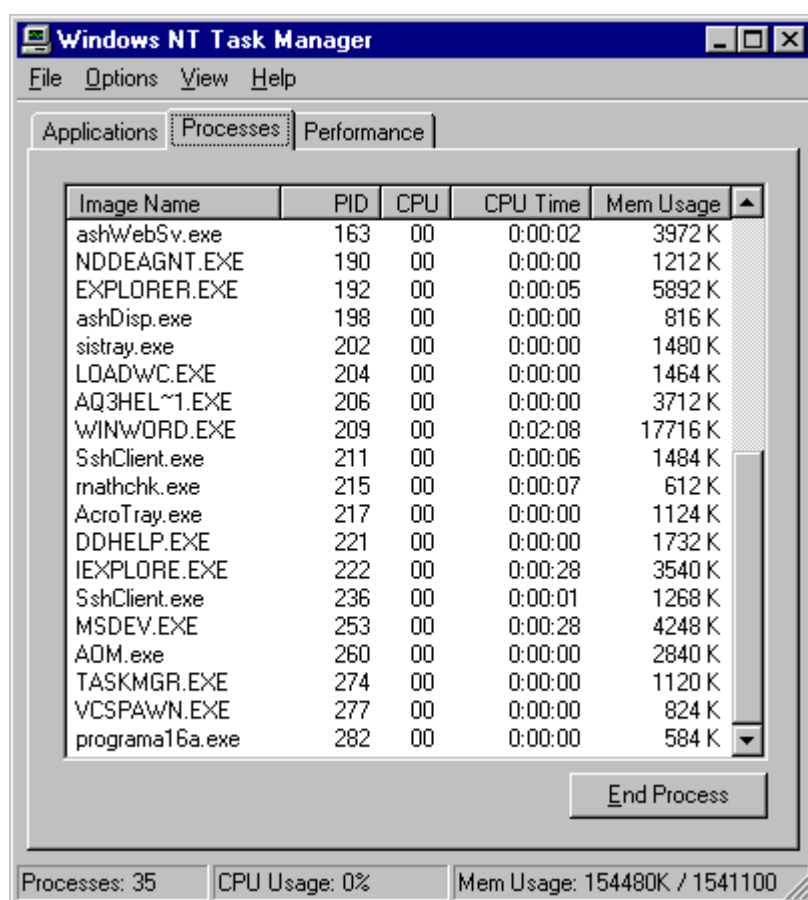


Figura 16.2 Memória inicial usada pelo programa16a.exe.

- 13) A Figura 16.4 mostra a memória estimada com a Eq. (16.1) para as variáveis A, B e C do programa16a.exe. Na Tabela 16.2 tem-se a comparação entre a memória estimada (Figura 16.4) e a memória efetivamente usada, que foi obtida do Task Manager, nos subitens (d) a (g) do item 12, acima.
- 14) **Executar novamente o programa com outros valores para N e analisar os novos resultados.**
- 15) No Fortran, para fechar o projeto atual, executar **File, Close Workspace**.

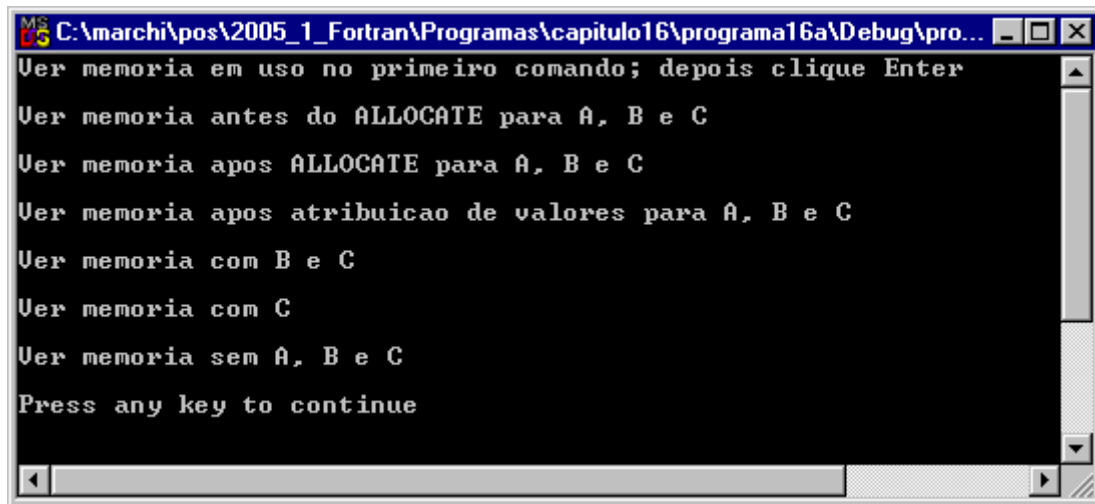


Figura 16.3 Janela DOS após a execução do programa16a.exe.

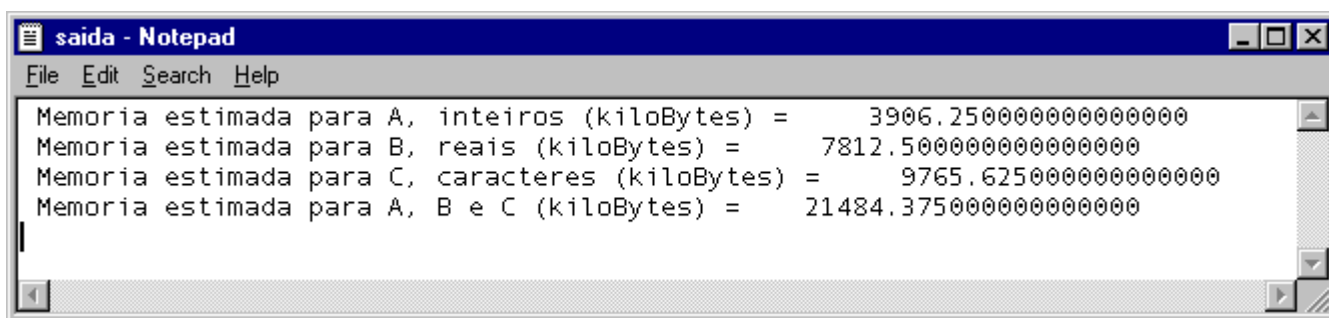


Figura 16.4 Memória estimada para executar o programa16a.exe.

Tabela 16.2 Comparação entre memória estimada e efetiva para as variáveis A, B e C do programa16a.f90.

Variável	Memória usada (kB)	Memória estimada (kB)	Erro %
A	3912	3906	0.15
B	7816	7812	0.05
C	9784	9766	0.18
A, B e C	21508	21484	0.11

16.2 programa16b.f90

- 1) Objetivos do programa:
 - (a) utilizar sub-rotina recursiva; e
 - (b) usar um novo comando do FORTRAN: RECURSIVE
- 2) No Fortran, seguindo o [procedimento-padrão](#), **criar um projeto** com o nome **programa16b**
- 3) No Fortran, seguindo o [procedimento-padrão](#), **criar e inserir** no projeto o programa-fonte **programa16b.f90**

- 4) Dentro do espaço de edição do Fortran, na subjanela maior, **copiar** exatamente o texto em vermelho mostrado na **Tabela 16.3**.
- 5) Comentários sobre o programa:
 - (a) Em algumas aplicações pode ser necessário que uma sub-rotina tenha que chamar ela própria. Neste caso, na definição da sub-rotina é necessário usar o comando “RECURSIVE” antes do nome da sub-rotina. Um exemplo é dado no programa16b.f90 na linha **RECURSIVE SUBROUTINE TESTE**
 - (b) A chamada da sub-rotina é feita da forma já descrita no capítulo 12, isto é, usando o comando CALL seguido do nome da sub-rotina. Um exemplo é dado no programa16b.f90 na linha **CALL TESTE**

Tabela 16.3 Programa16b.f90

```

PROGRAM RECURSIVO

INTEGER SOMA, K, I, MENOS

SOMA = 0
MENOS = 0

DO I = 1, 2
  WRITE(*,*) "MAIN/ciclo: I,K,SOMA,MENOS=", I, K, SOMA, MENOS
  K = 4
  CALL TESTE
END DO

WRITE(*,*) "MAIN/fora do ciclo: K, SOMA, MENOS = ", K, SOMA, MENOS

CONTAINS

! -----

RECURSIVE SUBROUTINE TESTE

  IF ( K == 1 ) THEN

    WRITE(*,*) "ROTINA,IF: K=1, SOMA, MENOS = ", K, SOMA, MENOS

  ELSE

    SOMA = SOMA + 1
  
```

```

K = K - 1
WRITE(*,*) "ROTINA,ELSE: K, SOMA, MENOS = ", K, SOMA, MENOS
CALL TESTE
MENOS = MENOS - 1
WRITE(*,*) "ROTINA,APOS CALL: K, SOMA, MENOS = ", K, SOMA, MENOS

END IF

END SUBROUTINE TESTE

! -----

END PROGRAM RECURSIVO

```

- 6) Executar **Build, Compile** para compilar o programa.
- 7) Gerar o programa-executável fazendo **Build, Build**.
- 8) Algoritmo do programa:
 - (a) definições iniciais de variáveis e suas atribuições
 - (b) no programa principal, inicia-se um ciclo para a variável I
 - (c) dentro deste ciclo, chama-se a sub-rotina TESTE, do tipo recursiva
 - (d) dentro da sub-rotina TESTE, são executados uma condição e alguns comandos, e chama-se a própria sub-rotina TESTE
 - (e) a sub-rotina TESTE continua a chamar ela própria até que seja satisfeita a condição
 - (f) em seguida, começa-se a retornar para fora da sub-rotina TESTE tantas vezes quanto ela foi chamada por ela própria, desde a última chamada até a primeira
 - (g) finalmente, retorna-se ao programa principal e encerra-se a sua execução
- 9) Executar o programa através de **Build, Execute**. O resultado é mostrado na Figura 16.5. **Analisar os resultados.**
- 10) **Alterar os valores do ciclo I e de K no programa principal. Executar novamente o programa e analisar os novos resultados.**
- 11) No Fortran, para fechar o projeto atual, executar **File, Close Workspace**.

16.3 programa16c.f90

- 1) Objetivos do programa:
 - (a) controlar o fluxo de informações para dentro e para fora de sub-rotinas; e
 - (b) usar um novo comando do FORTRAN: INTENT com parâmetros IN, OUT e INOUT.

Tabela 16.4 Programa16c.f90

```

PROGRAM CAPITULO_16C

IMPLICIT NONE

INTEGER A, B, C, D, E, F

WRITE(*,*) "Entre com o valor de A (inteiro)"
READ(*,*) A

WRITE(*,*) "Entre com o valor de B (inteiro)"
READ(*,*) B

C = 100
D = A
E = B
F = A + B

WRITE(*,*) "no programa principal, antes da sub-rotina: C = 100 = ", C
WRITE(*,*) "no programa principal, antes da sub-rotina: F = A+B = ", F

CALL SOMA ( A, B, C, D, E )

WRITE(*,*) "no programa principal, depois da sub-rotina: F = 2*(A+B) = ", F
WRITE(*,*) "no programa principal, depois da sub-rotina: C = A+B = ", C

! -----

CONTAINS

! -----

SUBROUTINE SOMA ( X, Y, Z, T, R )

INTEGER, INTENT(IN)      :: X, Y
INTEGER, INTENT(OUT)     :: Z
INTEGER, INTENT(INOUT)  :: T
INTEGER R, S

S = 50

```

```

WRITE(*,*) "dentro da sub-rotina: S = 50 = ", S
WRITE(*,*) "dentro da sub-rotina: Z = C = ", Z

Z = X + Y

WRITE(*,*) "dentro da sub-rotina: Z = A+B = ", Z
WRITE(*,*) "dentro da sub-rotina: T = A = ", T

T = T + 10
R = R + 10

WRITE(*,*) "dentro da sub-rotina: T = A+10 = ", T
WRITE(*,*) "dentro da sub-rotina: R = B+10 = ", R
WRITE(*,*) "dentro da sub-rotina: F = A+B = ", F
WRITE(*,*) "dentro da sub-rotina: A = ", A

F = 2 * F

WRITE(*,*) "dentro da sub-rotina: F = 2*(A+B) = ", F

END SUBROUTINE SOMA

! -----

END PROGRAM CAPITULO_16C

```

- (f) O comando INTENT é usado para controlar o fluxo de informações para dentro e para fora das sub-rotinas. Ele é usado quando se define cada variável empregada como argumento das sub-rotinas.
- (g) O comando INTENT é utilizado com três parâmetros (IN, OUT e INOUT) que definem o sentido do fluxo da informação.
- (h) O comando INTENT(IN) permite que a variável apenas receba informação de fora da sub-rotina. Um exemplo é dado no programa16c.f90 na linha `INTEGER, INTENT(IN) :: X, Y`. As variáveis X e Y só podem receber informação de fora da sub-rotina TESTE. A tentativa de atribuir valor a elas dentro da sub-rotina gerará erro de compilação. Mas elas podem ser empregadas em expressões dentro da sub-rotina, como na linha `Z = X + Y`
- (i) O comando INTENT(OUT) permite que a variável apenas envie informação para fora da sub-rotina. Um exemplo é dado no programa16c.f90 na linha `INTEGER, INTENT(OUT) :: Z`. A variável Z só pode enviar informação para fora da sub-rotina TESTE.

- (j) O comando INTENT(INOUT) permite que a variável receba informação de fora da sub-rotina e também que ela envie informação para fora da sub-rotina. Um exemplo é dado no programa16c.f90 na linha `INTEGER, INTENT (INOUT) :: T`. A variável T pode receber e enviar informação na sub-rotina TESTE.
- (k) Quando não se usa o comando INTENT numa variável que é argumento de uma sub-rotina, por default, assume-se implicitamente que ela foi definida como INTENT(INOUT). Um exemplo é dado no programa16c.f90 na linha `INTEGER R, S`, mas apenas para a variável R, uma vez que a variável S não é argumento da sub-rotina TESTE.
- (l) Recomenda-se o uso do comando INTENT. Ele evita erros de lógica, principalmente em programas grandes com muitas variáveis.
- (m) Recomenda-se usar como variáveis globais apenas aquelas que são efetivamente usadas no programa-principal. Variáveis utilizadas em sub-rotinas devem ser definidas apenas nas próprias sub-rotinas.
- 6) Executar **Build, Compile** para compilar o programa.
 - 7) Gerar o programa-executável fazendo **Build, Build**.
 - 8) Executar o programa através de **Build, Execute, usando por exemplo A = 1 e B = 2**. O resultado é mostrado na Figura 16.6. **Analisar os resultados considerando os comentários do item 5.**
 - 9) **Executar novamente o programa com outros valores para A e B, e analisar os novos resultados.**
 - 10) Encerrar a sessão seguindo o [procedimento-padrão](#).

```

C:\marchi\pos\2005_1_Fortran\Programas\capitulo16\programa16c\Debug\programa16c...
Entre com o valor de A <inteiro>
1
Entre com o valor de B <inteiro>
2
no programa principal, antes da sub-rotina: C = 100 =          100
no programa principal, antes da sub-rotina: F = A+B =           3
dentro da sub-rotina: S = 50 =           50
dentro da sub-rotina: Z = C =          100
dentro da sub-rotina: Z = A+B =           3
dentro da sub-rotina: T = A =           1
dentro da sub-rotina: T = A+10 =          11
dentro da sub-rotina: R = B+10 =          12
dentro da sub-rotina: F = A+B =           3
dentro da sub-rotina: A =           1
dentro da sub-rotina: F = 2*(A+B) =           6
no programa principal, depois da sub-rotina: F = 2*(A+B) =          6
no programa principal, depois da sub-rotina: C = A+B =           3
Press any key to continue_

```

Figura 16.6 Resultado do programa16c.f90.

16.4 EXERCÍCIOS

Exercício 16.1

Estimar e verificar a memória computacional necessária para usar uma matriz bidimensional, cujos elementos são do tipo inteiro, com X1 por Y1 elementos.

Exercício 16.2

Estimar e verificar a memória computacional necessária para usar uma matriz bidimensional, cujos elementos são do tipo real dupla, com X2 por Y2 elementos.

Exercício 16.3

Estimar e verificar a memória computacional necessária para usar uma matriz bidimensional, cujos elementos são do tipo caracter, cada um com 100 caracteres, com X3 por Y3 elementos.

Exercício 16.4

Juntar as matrizes dos exercícios 16.1 a 16.3 num único programa para estimar e verificar a memória computacional total que é necessária para as três matrizes.

Exercício 16.5

Adaptar a sub-rotina FATORIAL, da Tabela 12.4, do capítulo 12, para ser uma sub-rotina do tipo recursiva. E implementar um programa-principal para ela.

Exercício 16.6

Adaptar o programa12d, Tabela 12.7, do capítulo 12, para:

- (a) usar onde pertinente o comando INTENT com IN, OUT e INOUT;
- (b) usar o comando IMPLICIT NONE no programa-principal;
- (c) ter uma sub-rotina com a finalidade de ler os dados de um arquivo;
- (d) ter uma sub-rotina com a finalidade de escrever os resultados num arquivo; e
- (e) ter uma sub-rotina com a finalidade de abrir os arquivos de dados e de saída com o aplicativo Notepad.

Exercício 16.7

Adaptar o programa 16c.f90, Tabela 16.4, deste capítulo, para ter como argumentos da sub-rotina TESTE variáveis do tipo real dupla, caracteres e um conjunto unidimensional (vetor) com elementos do tipo inteiro.