Capítulo 16. **Memória e sub-rotinas 2**

30 Out 2024

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 novos do FORTRAN: DEALLOCATE, RECURSIVE, INTENT(IN,OUT,INOUT)

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, **criar um projeto** com o nome **programa16a**
3) No Fortran, **criar e inserir** no projeto o programa-fonte **programa16a.f90**
4) No Fortran, **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 já 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 as variáveis, em MegaBytes (MB), para executar um programa pode ser obtida através da seguinte equação:

$$Memória\ (MB) = \frac{4 * (Ni + Nrs) \quad + \quad 8 * Nrd \quad + \quad 1 * L * Nc}{1024 * 1024}$$

onde *Ni*, *Nrs* e *Nrd* representam o número total de variáveis simples, de elementos de conjuntos e matrizes do tipo inteiro, real simples e real dupla, respectivamente; *Nc* representa o número total de variáveis simples e de elementos do tipo caracter que têm comprimento ou número de caracteres igual a *L*; também devem ser incluídos os produtos *L*Nc* 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*1024 é um fator usado para converter o número de bytes em MegaBytes (MB). A estimativa da equação deve ser somada à memória necessária para executar o programa mesmo que ele não tenha qualquer variável.

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" )

WRITE(*,*) "Ver memoria em uso apos SYSTEM-DADOS.TXT"
READ(*,*)

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

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

CLOSE(1)

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

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

OPEN(UNIT, file = SAIDA )

WRITE(UNIT,*) "Memoria estimada para A, inteiros (MegaBytes) = ", M1
WRITE(UNIT,*) "Memoria estimada para B, reais (MegaBytes) = ", M2
WRITE(UNIT,*) "Memoria estimada para C, caracteres (MegaBytes) = ", M3
WRITE(UNIT,*) "Memoria estimada para A, B e C (MegaBytes) = ", 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 do Sistema Operacional Windows, 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 "Gerenciador de Tarefas"
   (c) clicar na opção "Processos" do menu principal do "Gerenciador de Tarefas"

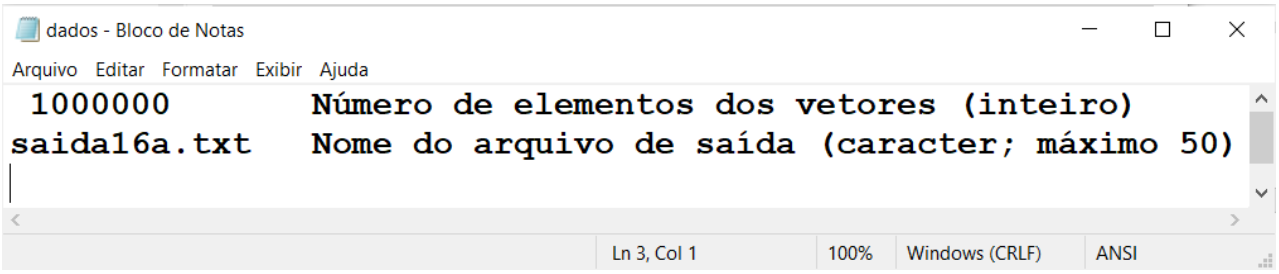

Figura 16.1 Exemplo de arquivo de dados para o programa16a.f90.

10) Executar o programa através de **Build, Execute**.
11) Conforme indica a janela DOS, **ver no "Gerenciador de Tarefas" qual a memória inicial que o programa está usando**. Para isso, deve-se selecionar a janela do **"Gerenciador de Tarefas"** e encontrar o processo chamado programa16a.exe na coluna "Usuários". Na linha deste processo, na coluna designada por "Memória", é indicada a memória em MegaBytes (MB) que o programa está usando no momento. Neste exemplo, deve ser de 0.4 MB, 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.
12) **Continuar a executar o programa16a.exe, mantendo a janela do "Gerenciador de Tarefas" 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 0.5 MB;
   (b) após a leitura dos dados e fechamento do aplicativo Notepad, a memória ainda está em 0.5 MB;
   (c) antes de usar o comando ALLOCATE com as variáveis A, B e C, a memória continua em 0.5 MB;
   (d) após ser usado o comando ALLOCATE com as variáveis A, B e C, a permanece em 0.5 MB; 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;
   (e) após a atribuição de valores às variáveis A, B e C a memória sobe para 21.5 MB; agora, de fato, o espaço de memória aberto com o comando ALLOCATE para as variáveis A, B e C está sendo usado;
   (f) após a eliminação da variável A, ou seja, mantendo as variáveis B e C, a memória cai para 17.6 MB;
   (g) após a eliminação das variáveis A e B, ou seja, mantendo apenas a variável C, a memória cai para 10.0 MB;
   (h) após a eliminação das variáveis A, B e C, a memória cai para 0.5 MB;

(i) 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 Gerenciador de Tarefas.

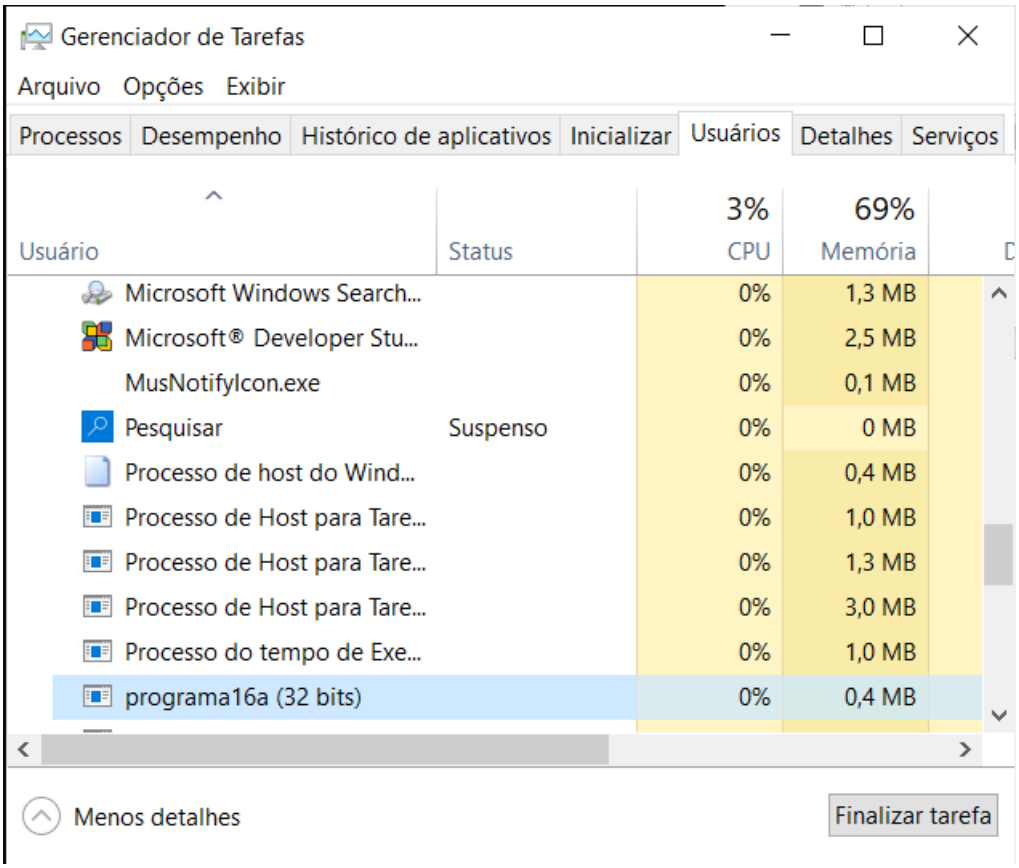

Figura 16.2 Memória inicial usada pelo programa16a.exe.

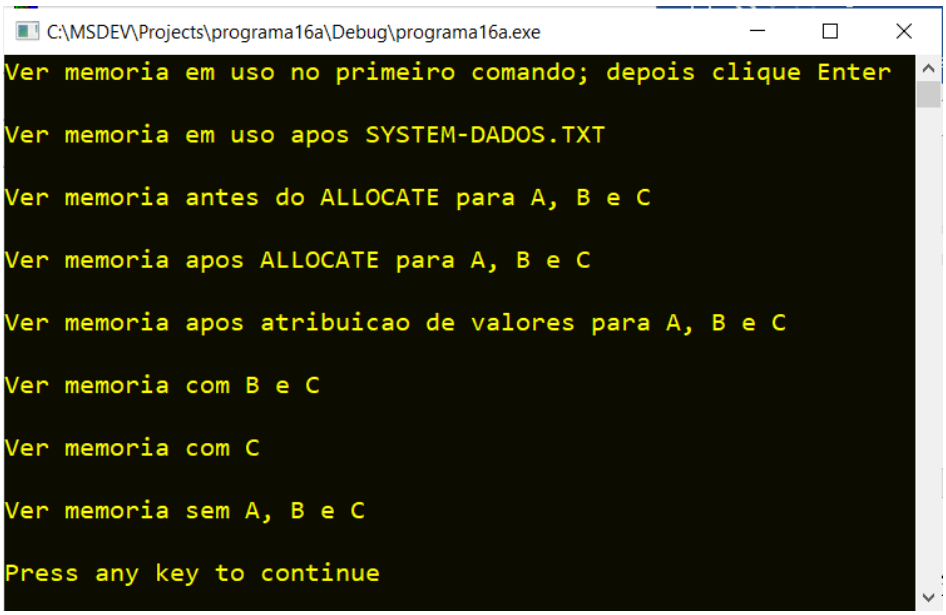

Figura 16.3 Janela DOS após a execução do programa16a.exe.

13) A Figura 16.4 mostra a memória estimada com a equação 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 Gerenciador de Tarefas, nos subitens (d) a (g) do item 12, acima.

14) **Executar novamente o programa com outros valores para N e analisar os novos resultados**.

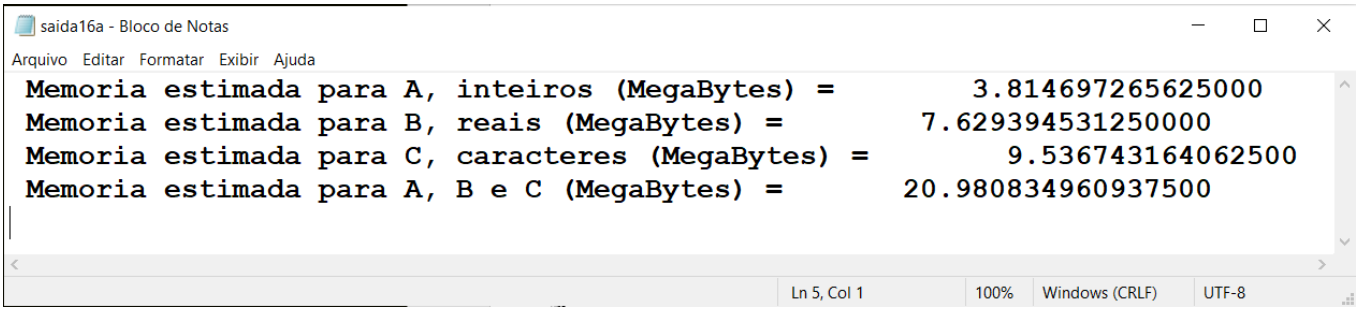

Figura 16.4 Memória estimada para executar o programa16a.exe.

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

Variável	Memória usada (MB)	Memória estimada (MB)	Erro %
A	3.9	3.81	2.3
B	7.6	7.63	0.4
C	9.5	9.54	0.4
A, B e C	21.0	20.98	0.1

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, **criar um projeto** com o nome **programa16b**
3) No Fortran, **criar e inserir** no projeto o programa-fonte **programa16b.f90**
4) No Fortran, **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 11, isto é, usando o comando CALL seguido do nome da sub-rotina. Um exemplo é dado no programa16b.f90 na linha `CALL TESTE`
6) Executar **Build, Compile** para compilar o programa.
7) Gerar o programa-executável fazendo **Build, Build**.

Tabela 16.3 Programa16b.f90

```
PROGRAM RECURSIVO

INTEGER SOMA, K, I, MENOS

SOMA = 0
MENOS = 0
K = 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
```

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**.

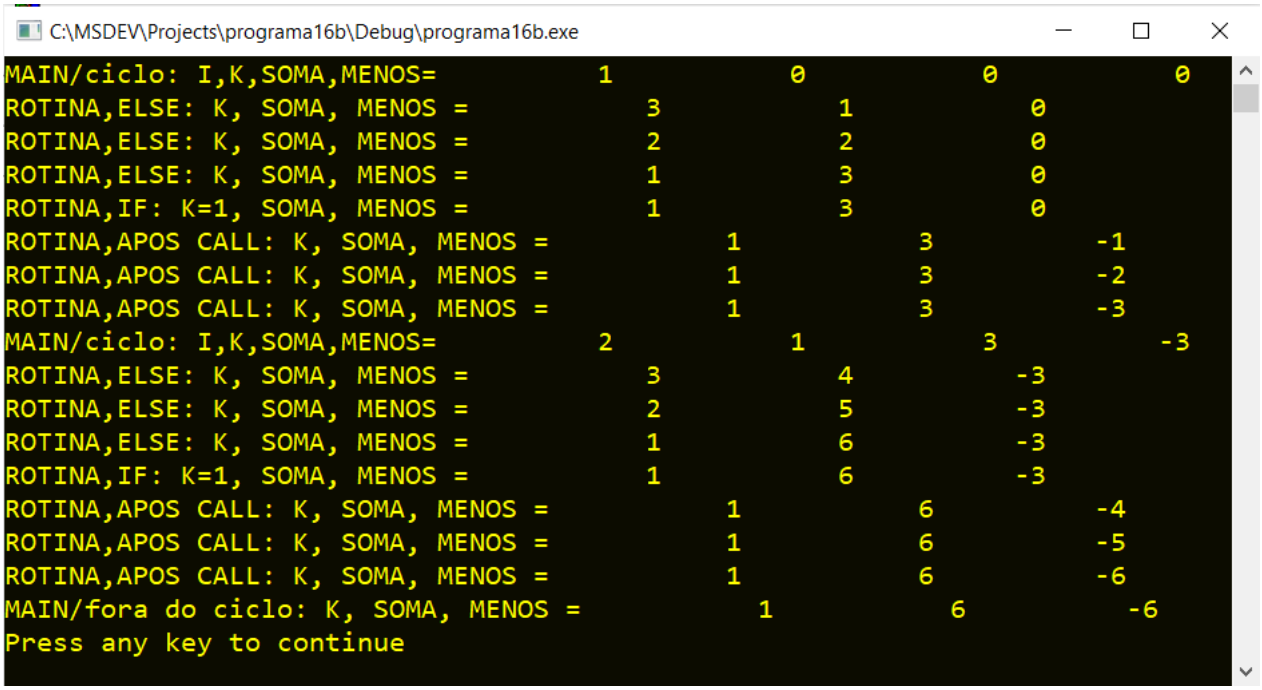

Figura 16.5 Resultado do programa16b.f90.

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.
2) No Fortran, **criar um projeto** com o nome **programa16c**
3) No Fortran, **criar e inserir** no projeto o programa-fonte **programa16c.f90**

4) No Fortran, **copiar** exatamente o texto em vermelho mostrado na **Tabela 16.4**.

5) Comentários sobre o programa:

   (a) O uso do comando IMPLICIT NONE no programa-principal obriga que sejam declaradas explicitamente todas as variáveis usadas no programa-principal e em todas as sub-rotinas do mesmo programa. Caso contrário, ocorrerá erro de compilação.

   (b) Todas as variáveis definidas no programa-principal são válidas dentro das sub-rotinas do programa. Elas são chamadas de variáveis globais. Por exemplo, as variáveis A, B, C, D, E e F do programa16c.f90, definidas na linha `INTEGER A, B, C, D, E, F`, são variáveis globais do programa.

   (c) As variáveis definidas dentro de uma sub-rotina são válidas somente dentro da própria sub-rotina. Elas são chamadas de variáveis locais. Seus valores ou conteúdos podem ser passados para outras sub-rotinas ou para o programa-principal através dos argumentos da sub-rotina. Por exemplo, as variáveis R e S do programa16c.f90, definidas na linha `INTEGER R, S`, são variáveis locais da sub-rotina SOMA.

   (d) Todas as atribuições feitas a variáveis globais são reconhecidas dentro das sub-rotinas do programa.

   (e) Todas as atribuições feitas a variáveis locais não são reconhecidas fora das respectivas sub-rotinas.

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 SOMA. 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) deveria apenas permitir que a variável enviasse informação para fora da sub-rotina. Porém, como será visto, de fato, a variável também recebe informação. Um exemplo é dado no programa16c.f90 na linha `INTEGER, INTENT(OUT) :: Z`. A variável Z só deveria enviar informação para fora da sub-rotina SOMA, mas como é mostrado na Figura 16.6, ela também recebe o valor da variável C do programa-principal.

(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 SOMA.

(k) Quando não se usa o comando INTENT, 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 SOMA.

(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.**

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.

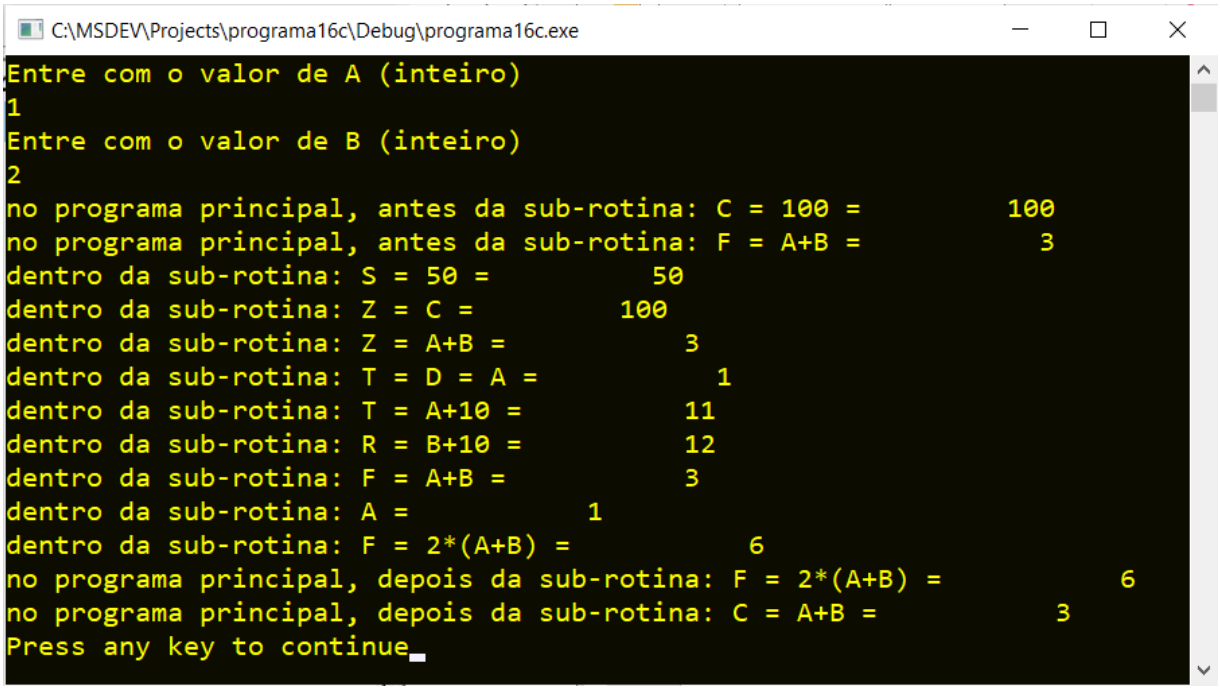

Figura 16.6 Resultado do programa16c.f90.

Exercício 16.2

Adaptar o programa16a.f90 para estimar a memória computacional necessária para uma matriz bidimensional cujos elementos são do tipo real de precisão dupla, com X2 por Y2 elementos. Verificar a memória efetivamente usada.

Usar X2 = 10 mil e Y2 = 1 mil

Resultado esperado: memória ≈ 76.3 MB

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 em um 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 11.4, do capítulo 11, para ser uma sub-rotina do tipo recursiva. E implementar um programa-principal para ela.

Exercício 16.6

Adaptar o programa11d, Tabela 11.7, do capítulo 11, 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 em um 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 programa16c.f90, Tabela 16.4, deste capítulo, para ter como argumentos da sub-rotina SOMA variáveis do tipo real de precisão dupla, caracteres e um conjunto unidimensional (vetor) com elementos do tipo inteiro.