

8.1 Sub-rotinas (programação modularizada)

Sub-rotinas, também chamadas subprogramas, são blocos de instruções que realizam tarefas específicas. O código de uma sub-rotina é carregado uma vez e pode ser executado quantas vezes forem necessárias. Como o problema pode ser subdividido em pequenas tarefas, os programas tendem a ficar menores e mais organizados.

Os programas, em geral, são executados linearmente, uma linha após a outra, até o fim. Entretanto, quando são utilizadas sub-rotinas, é possível a realização de desvios na execução dos programas. Esses desvios são efetuados quando uma função é chamada pelo programa principal. Observe o exemplo a seguir (a numeração das linhas à esquerda está sendo utilizada apenas para facilitar a explicação).

```

1. ALGORITMO
2. DECLARE sal, aum, novo_sal NUMÉRICO
3. LEIA sal
4. aum ← calculo (sal)
5. novo_sal ← sal + aum
6. ESCREVA "Novo salário é", novo_sal
7. FIM_ALGORITMO.

8. SUB-ROTINA calculo (sal NUMÉRICO) // passagem de parâmetro por valor
9. DECLARE perc, valor NUMÉRICO
10. LEIA perc
11. valor ← sal * perc / 100
12. RETORNE valor
13. FIM_SUB_ROTINA calculo

```

O algoritmo apresentado tem como objetivo receber o valor atual do salário de um funcionário e calcular o novo salário. Para resolver esse problema, utilizou-se o programa principal (representado pelo bloco de instruções entre as linhas 1 e 7) e uma sub-rotina (representada pelo bloco de instruções entre as linhas 8 e 13).

O programa principal é executado linearmente até a linha 4. Nesse ponto, existe uma chamada à sub-rotina `calculo` (que recebe como parâmetro o valor do salário inicial) e o programa principal fica temporariamente suspenso. A ordem de execução das instruções é, então, desviada para a linha 8, onde começa a sub-rotina `calculo`. A execução só volta ao programa principal quando o comando `RETORNE` for executado (linha 12). Esse comando é responsável, também, por devolver ao programa principal um valor calculado dentro da sub-rotina (nesse exemplo, foi devolvido o conteúdo da variável `valor`). A execução do programa principal é retomada exatamente no ponto em que foi interrompida; dessa maneira, o valor devolvido pela sub-rotina é atribuído à variável `aum` (linha 4). A partir daí, o programa volta a ser executado linearmente até o fim (linha 7).

O exemplo apresentado mostra a utilização de uma sub-rotina que recebe um parâmetro (o valor atual do salário) e que, ao final, retorna um valor (aumento que será dado ao salário) para quem a chamou. Porém, as sub-rotinas podem não receber parâmetros nem retornar valor.

Outro ponto que precisa ser destacado é que, dentro das sub-rotinas, pode ocorrer declaração de variáveis, chamadas variáveis locais. Elas recebem esse nome porque podem ser utilizadas apenas dentro da sub-rotina. Quando a execução desta chega ao fim, essas variáveis são destruídas e seus conteúdos são perdidos.

Variáveis declaradas fora de qualquer sub-rotina são chamadas globais. Elas recebem esse nome porque qualquer ponto do programa, incluindo as sub-rotinas, pode utilizá-las. São destruídas quando a execução do programa chega ao fim.



Observação

Não se aconselha a utilização excessiva de variáveis globais, por tornar difícil a manutenção e a busca por erros nos programas.

No algoritmo usado como exemplo anteriormente, tem-se 3 variáveis globais, declaradas na linha 2 e denominadas `sal`, `aum` e `novo_sal`. Na linha 9, existe a declaração de 2 variáveis locais da sub-rotina `calculo`. São elas: `perc` e `valor`.

É importante salientar que, na linha 4, onde a sub-rotina `calculo` é chamada, a variável `sal` está sendo passada como parâmetro e, na linha 8, cabeçalho da sub-rotina `calculo`, uma variável, que pode ou não ter o mesmo nome, está recebendo o valor do parâmetro.

O parâmetro pode ser passado por valor e, então, a variável do cabeçalho se comportará como uma variável local da sub-rotina.

O parâmetro pode ser passado por referência e, então, a variável do cabeçalho se comportará como uma variável global.

A passagem de parâmetros por valor ou por referência depende da sintaxe de cada linguagem e as próximas seções demonstrarão essas especificidades.

Exemplo de variáveis locais e globais:

```

1. ALGORITMO
2. DECLARE X, Y NUMÉRICO      //      variáveis globais
3. X ← 1
4. Y ← 2
5. ESCRIVA "VALORES INICIAIS"
6. ESCRIVA "X = ",X," e Y = ",Y
7. S1                          // chamada da sub-rotina S1, sem parâmetros
8. ESCRIVA "VALORES DEPOIS DA EXECUÇÃO DA SUB-ROTINA S1"
9. ESCRIVA "X = ",X," e Y = ",Y
10. S2(X,Y)                    //chamada da sub-rotina S2, com parâmetros por valor
11. ESCRIVA "VALORES DEPOIS DA EXECUÇÃO DA SUB-ROTINA S2"
12. ESCRIVA "X = ",X," e Y = ",Y
13. S3(X,Y)                    // chamada da sub-rotina S3, com parâmetros por referência
14. ESCRIVA "VALORES DEPOIS DA EXECUÇÃO DA SUB-ROTINA S3"
15. ESCRIVA "X = ",X," e Y = ",Y
16. FIM_ALGORITMO.

17. SUB_ROTINA S1
18. // sub-rotina sem parâmetros e sem retorno
19. DECLARE X, Y, Z NUMÉRICO // variáveis locais da sub-rotina S1
20. X ← 8
21. Y ← 10
22. Z ← 5
23. ESCRIVA "VALORES IMPRESSOS DENTRO DA SUB-ROTINA S1"
24. ESCRIVA "X = ",X

```

```

25. ESCREVA "Y = ",Y
26. ESCREVA "Z = ",Z
27. FIM_SUB_ROTINA S1

28. SUB_ROTINA S2 (X,Y NUMÉRICO)
29. // sub-rotina com parâmetro por valor e sem retorno
30. DECLARE Z NUMÉRICO // variável local da sub-rotina S2
31. X ← X + 2
32. Y ← Y * 2
33. Z ← X + Y
34. ESCREVA "VALORES IMPRESSOS DENTRO DA SUB-ROTINA S2"
35. ESCREVA "X = ",X
36. ESCREVA "Y = ",Y
37. ESCREVA "Z = ",Z
38. FIM_SUB_ROTINA S2

39. SUB_ROTINA S3 (X,Y NUMÉRICO)
40. // sub-rotina com parâmetro por referência e sem retorno
41. DECLARE A NUMÉRICO // variável local da sub-rotina S3
42. A ← X + Y
43. X ← X - 1
44. Y ← Y - 2
45. ESCREVA "VALORES IMPRESSOS DENTRO DA SUB-ROTINA S3"
46. ESCREVA "X = ",X
47. ESCREVA "Y = ",Y
48. ESCREVA "A = ",A
49. FIM_SUB_ROTINA S3

```

O algoritmo anterior gera a saída a seguir:

VARIÁVEIS												Linha executada	Saída na tela
globais		locais da sub-rotina s1			locais da sub-rotina s2			locais da sub-rotina s3					
X	Y	X	Y	Z	X	Y	Z	X	Y	A			
1											3		
	2										4		
											5	VALORES INICIAIS	
											6	X = 1 e Y = 2	
											7		
											17		
		8									20		
			10								21		
				5							22		
											23	VALORES IMPRESSOS DENTRO DA SUB-ROTINA S1	
											24	X = 8	
											25	Y = 10	
											26	Z = 5	
											8	VALORES DEPOIS DA EXECUÇÃO DA SUB-ROTINA S1	
											9	X = 1 e Y = 2	
											10		
					1	2					28		
					3						31		
						4					32		
							7				33		
											34	VALORES IMPRESSOS DENTRO DA SUB-ROTINA S2	

VARIÁVEIS													
globais			locais da sub-rotina s1			locais da sub-rotina s2			locais da sub-rotina s3			Linha executada	Saída na tela
												35	X = 3
												36	Y = 4
												37	Z = 7
												11	VALORES DEPOIS DA EXECUÇÃO DA SUB-ROTINA S2
												12	X = 1 e Y = 2
												13	
								1	2			39	
										3		42	
0								0				43	
	0								0			44	
												45	VALORES IMPRESSOS DENTRO DA SUB-ROTINA S3
												46	X = 0
												47	Y = 0
												48	A = 3
												14	VALORES DEPOIS DA EXECUÇÃO DA SUB-ROTINA S3
												15	X = 0 e Y = 0

8.2 Sub-rotinas em PASCAL (*procedures*, *functions* e *units*)

A linguagem PASCAL possibilita a modularização por meio de *procedures* (procedimentos), *functions* (funções) e *units* (unidades). As sub-rotinas do tipo *procedures* (procedimentos) e *functions* (funções) devem ter seus códigos descritos antes do **BEGIN** do programa principal. Apenas as *units* (unidades) apresentam sintaxe diferenciada e descrita na Seção 8.2.7.

As variáveis globais, ou seja, aquelas que são reconhecidas por todas as partes do programa, devem ser declaradas antes do **BEGIN** do programa principal. As variáveis locais devem ser declaradas dentro das sub-rotinas e são reconhecidas apenas na sub-rotina onde foram declaradas.

A seguir, um programa exemplo que soma dois números digitados pelo usuário será utilizado para demonstrar os diferentes tipos de sub-rotinas.

8.2.1 *Procedures* sem passagem de parâmetros

As *procedures* (procedimentos) são rotinas chamadas pelo programa principal para executar alguma operação específica, mas não retornam valor para quem as chamou. Possuem a seguinte sintaxe:

```
PROCEDURE nome_da_procedure;
  declaração_de_variáveis_locais;
BEGIN
  comandos;
END;
```

Quando o programa encontra uma linha contendo o nome da *procedure*, o fluxo da execução é desviado para as linhas contidas dentro dela. Essa execução só retornará ao fluxo normal quando a execução da *procedure* chegar ao fim.

A seguir, é apresentado um exemplo de *procedure* sem parâmetros (a numeração das linhas não faz parte do programa) com a utilização de variáveis globais.

```
1. PROGRAM EXEMPLO;
2. USES CRT;
3. VAR A, B, S : INTEGER; {variáveis globais}
4. PROCEDURE SOMAR;
5. BEGIN
```

```

6.      S := A + B;
7.      END;

8.      BEGIN                {início do programa principal}
9.      CLRSCR;
10.     WRITELN('Digite o primeiro número: ');
11.     READLN(A);
12.     WRITELN('Digite o segundo número: ');
13.     READLN(B);
14.     SOMAR;                {chamada da sub-rotina SOMAR}
15.     WRITELN('Soma = ',S);
16.     READLN;
17.     END.

```

O programa começa sua execução no **BEGIN** principal, representado no exemplo anterior pela linha 8. Posteriormente, executa as linhas 9 a 14. Na linha 14, existe a chamada a uma *procedure*. O programa principal é desviado para a *procedure*, denominada **SOMAR**. Assim, o fluxo de execução do programa vai para a linha 4, executando toda a *procedure*, ou seja, da linha 4 à 7. Em seguida, ele retorna à linha 15, exatamente abaixo da linha onde ocorreu o desvio. Executa as linhas 15, 16 e 17 e o programa é encerrado.

A seguir, é apresentado um exemplo de *procedure* sem parâmetros (a numeração das linhas não faz parte do programa) com a utilização de variáveis locais.

```

1.      PROGRAM EXEMPLO;
2.      USES CRT;
3.      PROCEDURE SOMAR;
4.      VAR A, B, S: INTEGER; {variáveis locais da sub-rotina SOMAR}
5.      BEGIN
6.      WRITELN('Digite o primeiro número: ');
7.      READLN(A);
8.      WRITELN('Digite o segundo número: ');
9.      READLN(B);
10.     S := A + B;
11.     WRITELN('Soma = ',S);
12.     END;

13.     BEGIN                {início do programa principal}
14.     CLRSCR;
15.     SOMAR;                {chamada da sub-rotina SOMAR}
16.     READLN;
17.     END.

```

O programa começa sua execução no **BEGIN** principal, representado no exemplo anterior pela linha 13. Posteriormente, executa as linhas 14 e 15. Na linha 15, existe a chamada a uma *procedure*. O programa principal é desviado para a *procedure*, denominada **SOMAR**. Assim, o fluxo de execução do programa vai para a linha 3, executando toda a *procedure*, ou seja, as linhas 4 à 12. Em seguida, ele retorna à linha 16, exatamente abaixo da linha onde ocorreu o desvio. Executa as linhas 16 e 17 e o programa é encerrado.

8.2.2 *Procedures* com passagem de parâmetros por valor

Pode-se utilizar *procedure* com passagem de parâmetros, ou seja, no momento em que a execução da *procedure* é solicitada, alguns valores lhe são fornecidos. Observe a sintaxe:

```

PROCEDURE nome_da_procedure(parâmetros:tipo_dos_dados);
declaração_de_variáveis_locais;
BEGIN
    comandos;
END;

```

A seguir, é mostrado um exemplo de *procedure* com passagem de parâmetros por valor, ou seja, os parâmetros são variáveis locais da sub-rotina (a numeração das linhas não faz parte do programa).

```

1.  PROGRAM EXEMPLO;
2.  USES CRT;
3.  VAR A, B: INTEGER; {variáveis globais}
4.  PROCEDURE SOMAR(X,Y: INTEGER);
5.  VAR S: INTEGER;      {variável local da sub-rotina SOMAR}
6.  BEGIN
7.  S := X + Y;
8.  WRITELN('Soma = ',S);
9.  END;

10. BEGIN           {início do programa principal}
11. CLRSCR;
12. WRITELN('Digite o primeiro número: ');
13. READLN(A);
14. WRITELN('Digite o segundo número: ');
15. READLN(B);
16. SOMAR(A,B);      {chamada da sub-rotina SOMAR}
17. READLN;
18. END.

```

O programa começa sua execução no BEGIN principal, representado no exemplo anterior pela linha 10. Posteriormente, executa as linhas 11 a 16. Na linha 16, existe a chamada a uma *procedure*. O programa principal é desviado para a *procedure*, denominada SOMAR, que possui as variáveis A e B como parâmetros. Assim, o fluxo de execução do programa vai para a linha 4, executando toda a *procedure*, ou seja, as linhas 5 a 9. O primeiro parâmetro é a variável A e será passada para a variável x. O segundo parâmetro é a variável B e será passada para a variável y. Em seguida, ele retorna à linha 17, exatamente abaixo da linha onde ocorreu o desvio. Executa as linhas 17 e 18 e o programa é encerrado.

A linguagem PASCAL não permite a passagem de vetores e matrizes como parâmetros da mesma forma em que são passados parâmetros de tipos primitivos (INTEGER, REAL, STRING e CHAR). Para passar um vetor ou uma matriz como parâmetro é necessário definir um novo tipo.

Exemplo de vetor passado como parâmetro para uma sub-rotina:

```

PROGRAM VETOR;
USES CRT;
TYPE X = ARRAY[1..5] OF INTEGER;
VAR I: INTEGER;
    W: X;
PROCEDURE MOSTRAR(Y: X);
BEGIN
    WRITELN('MOSTRANDO O VETOR NA SUB-ROTINA');
    FOR I:=1 TO 5 DO
    BEGIN
        WRITELN(Y[I]);
    END;
END;

BEGIN           {BEGIN DO PROGRAMA PRINCIPAL}
CLRSCR;
WRITELN('DIGITANDO OS NÚMEROS DO VETOR ');
FOR I:=1 TO 5 DO
BEGIN
    READLN(W[I]);

```

```
END;
MOSTRAR(W);      {CHAMADA DA SUB-ROTINA MOSTRAR}
READLN;
END.
```

8.2.3 *Procedures* com passagem de parâmetros por referência

Pode-se utilizar *procedure* com passagem de parâmetros, ou seja, no momento em que a execução da *procedure* é solicitada, alguns valores lhe são fornecidos. Observe a sintaxe:

```
PROCEDURE nome_da_procedure(VAR parâmetros:tipo_dos_dados);
declaração_de_variáveis_locais;
BEGIN
    comandos;
END;
```

A seguir, é mostrado um exemplo de *procedure* com passagem de parâmetros por referência, ou seja, os parâmetros se comportam como variáveis globais (a numeração das linhas não faz parte do programa).

```
1.  PROGRAM EXEMPLO;
2.  USES CRT;
3.  VAR A, B: INTEGER;    {variáveis globais}
4.  PROCEDURE SOMAR(VAR X,Y: INTEGER);
5.  VAR S: INTEGER;       {variável local da sub-rotina SOMAR}
6.  BEGIN
7.  S := X + Y;
8.  WRITELN('Soma = ',S);
9.  END;

10. BEGIN                {início do programa principal}
11. CLRSCR;
12. WRITELN('Digite o primeiro número: ');
13. READLN(A);
14. WRITELN('Digite o segundo número: ');
15. READLN(B);
16. SOMAR(A,B);           {chamada da sub-rotina SOMAR}
17. READLN;
18. END.
```

O programa começa sua execução no *BEGIN* principal, representado no exemplo anterior pela linha 10. Posteriormente, executa as linhas 11 a 16. Na linha 16, existe a chamada a uma *procedure*. O programa principal é desviado para a *procedure*, denominada *SOMAR*, que possui as variáveis *A* e *B* como parâmetros. Assim, o fluxo de execução do programa vai para a linha 4, executando toda a *procedure*, ou seja, as linhas 5 a 9. O primeiro parâmetro é a variável *A* e será passada para a variável *x*. O segundo parâmetro é a variável *B* e será passada para a variável *y*. Como os parâmetros foram passados por referência e isso é identificado pela presença da palavra *VAR* no cabeçalho da *procedure*, qualquer alteração nos valores de *x* ou de *y* será também refletida nas variáveis *A* e *B*, respectivamente. Em seguida, ele retorna à linha 17, exatamente abaixo da linha onde ocorreu o desvio. Executa as linhas 17 e 18 e o programa é encerrado.

8.2.4 *Function* sem passagem de parâmetros

Uma *function* (função) tem o mesmo objetivo que uma *procedure*, ou seja, desviar a execução do programa principal para realizar uma tarefa específica, com uma única diferença: uma *function* sempre retorna um valor. A sintaxe de uma *function* é:

```

FUNCTION nome_da_function : tipo_de_dado_do_valor_retornado;
declaração_de_variáveis_locais;
BEGIN
    comandos;
END;

```

É importante ressaltar que, para que ocorra o retorno de algum valor para quem chamou a *function*, deve-se atribuir tal valor a uma variável cujo nome seja igual ao dado à *function*.

A chamada à *function* acontece atribuindo seu nome a uma variável ou a uma condição, que receberá o retorno produzido. A seguir, é apresentado um exemplo (a numeração das linhas não faz parte do programa).

```

1.    PROGRAM EXEMPLO;
2.    USES CRT;
3.    VAR A, B, S : INTEGER; {variáveis globais}
4.    FUNCTION SOMAR: INTEGER;
5.    BEGIN
6.    SOMAR := A + B;
7.    END;

8.    BEGIN           {início do programa principal}
9.    CLRSCR;
10.   WRITELN('Digite o primeiro número: ');
11.   READLN(A);
12.   WRITELN('Digite o segundo número: ');
13.   READLN(B);
14.   S := SOMAR;      {chamada da sub-rotina SOMAR}
15.   WRITELN('Soma = ',S);
16.   READLN;
17.   END.

```

O programa começa sua execução no **BEGIN** principal, representado no exemplo anterior pela linha 8. Posteriormente, executa as linhas 9 a 14. Na linha 14, existe a chamada a uma *function*. O programa principal é desviado para a *function*, denominada **SOMAR**. Assim, o fluxo de execução do programa vai para a linha 4, executando toda a *function*, ou seja, da linha 4 à 7. Em seguida, ele retorna à linha 14, atribuindo o valor retornado à variável **s**. Por fim, executa as linhas 15, 16 e 17, e o programa é encerrado.

8.2.5 *Function* com passagem de parâmetros por valor

Uma *function* pode receber parâmetros no momento em que é chamada. Os valores informados são copiados, sequencialmente, em variáveis descritas em seu cabeçalho. A sintaxe correta é a seguinte:

```

FUNCTION nome_da_function(parâmetros:tipo_dos_dados):tipo_de_dado_do_valor_retornado;
declaração_de_variáveis_locais;
BEGIN
    comandos;
END;

```

A chamada a uma *function* acontece atribuindo seu nome a uma variável ou a uma condição, que receberá o retorno ao término de sua execução. No momento da chamada, são informados, também, os parâmetros que deverão ser levados para a *function*. A seguir, é apresentado um exemplo de *function* com passagem de parâmetros por valor, ou seja, os parâmetros são variáveis locais da sub-rotina (a numeração das linhas não faz parte do programa).


```

1.    PROGRAM EXEMPLO;
2.    USES CRT;
3.    VAR A, B, S : INTEGER; {variáveis globais}
4.    FUNCTION SOMAR(X,Y: INTEGER): INTEGER;
5.    BEGIN
6.    SOMAR := X + Y;
7.    END;

8.    BEGIN          {início do programa principal}
9.    CLRSCR;
10.   WRITELN('Digite o primeiro número: ');
11.   READLN(A);
12.   WRITELN('Digite o segundo número: ');
13.   READLN(B);
14.   S := SOMAR(A,B);      {chamada da sub-rotina SOMAR}
15.   WRITELN('Soma = ',S);
16.   READLN;
17.   END.

```

O programa começa sua execução no BEGIN principal, representado no exemplo anterior pela linha 8. Posteriormente, executa as linhas 9 a 14. Na linha 14, existe a chamada a uma *function*. O programa principal é desviado para a *function*, denominada SOMAR. Assim, o fluxo de execução do programa vai para a linha 4, executando toda a *function*, ou seja, da linha 4 à 7. O primeiro parâmetro é a variável A e será passada para a variável x. O segundo parâmetro é a variável B e será passada para a variável y. Em seguida, ele retorna à linha 14, atribuindo o valor retornado à variável s. Por fim, executa as linhas 15, 16 e 17 e o programa é encerrado.

8.2.6 *Function* com passagem de parâmetros por referência

Uma *function* pode receber parâmetros no momento em que é chamada. Os valores informados são copiados, sequencialmente, em variáveis descritas em seu cabeçalho. A sintaxe correta é a seguinte:

```

FUNCTION nome_da_function(VAR parâmetros:tipo_dos_dados):tipo_de_dado do_valor_retornado;
declaração_de_variáveis_locais;
BEGIN
    comandos;
END;

```

A chamada a uma *function* acontece atribuindo seu nome a uma variável ou a uma condição, que receberá o retorno ao término de sua execução. No momento da chamada, são informados, também, os parâmetros que deverão ser levados para a *function*. A seguir, é apresentado um exemplo de *function* com passagem de parâmetros por referência, ou seja, os parâmetros se comportam como variáveis globais (a numeração das linhas não faz parte do programa).

```

1.    PROGRAM EXEMPLO;
2.    USES CRT;
3.    VAR A, B, S : INTEGER; {variáveis globais}
4.    FUNCTION SOMAR(VAR X,Y: INTEGER): INTEGER;
5.    BEGIN
6.    SOMAR := X + Y;
7.    END;

8.    BEGIN          {início do programa principal}
9.    CLRSCR;

```

```

10.    WRITELN('Digite o primeiro número: ');
11.    READLN(A);
12.    WRITELN('Digite o segundo número: ');
13.    READLN(B);
14.    S := SOMAR(A,B);          {chamada da sub-rotina SOMAR}
15.    WRITELN('Soma = ',S);
16.    READLN;
17.    END.

```

O programa começa sua execução no `BEGIN` principal, representado no exemplo anterior pela linha 8. Posteriormente, executa as linhas 9 a 14. Na linha 14, existe a chamada a uma *function*. O programa principal é desviado para a *function*, denominada `SOMAR`. Assim, o fluxo de execução do programa vai para a linha 4, executando toda a *function*, ou seja, da linha 4 à 7. O primeiro parâmetro é a variável `A` e será passada para a variável `x`. O segundo parâmetro é a variável `B` e será passada para a variável `y`. Como os parâmetros foram passados por referência e isso é identificado pela presença da palavra `VAR` no cabeçalho da *function*, qualquer alteração nos valores de `x` ou de `y` será também refletida nas variáveis `A` e `B`, respectivamente. Em seguida, ele retorna à linha 14, atribuindo o valor retornado à variável `s`. Por fim, executa as linhas 15, 16 e 17 e o programa é encerrado.

8.2.7 Units

Uma *unit* é um arquivo (.PAS), que pode conter várias *procedures* e *functions*, e, depois de compilado, torna-se uma biblioteca (.TPU), que pode ser chamada por outros programas por meio do comando `USES`. A sintaxe de uma *unit* é a seguinte.

```

UNIT nome_da_unit;      {o nome da unit deve ser o nome do arquivo}
INTERFACE
    Cabeçalho das procedures e das functions;
IMPLEMENTATION
    Implementação das procedures e das functions;
END;

```

Exemplo de unit:

```

unit calcula;           {esse arquivo deve ser salvo como CALCULA.PAS}
interface
    procedure somar(a,b: integer);
    function multiplicar(a,b,c: integer): integer;
implementation
    procedure somar(a,b:integer);
    var s: integer;
    begin
        s := a + b;
        writeln('Soma = ',s);
    end;
    function multiplicar(a,b,c:integer):integer;
    begin
        multiplicar := a * b * c;
    end;
end.

```

Para criar uma *unit*, ou seja, uma biblioteca, é necessário abrir um novo arquivo, digitar os códigos da biblioteca, como mostra o exemplo anterior, e salvar o arquivo com o mesmo nome da *unit*. Esse arquivo será .PAS. Em seguida, no menu `COMPILE`, altere o destino da compilação para `DISK`. Assim,

depois que a *unit* for compilada e não apresentar mais erros, será gerado o arquivo *.TPU* que é a biblioteca propriamente dita.

Exemplo de programa que utiliza uma *unit* (biblioteca):

```
program sub_rotina;
uses crt, calcula;           {utilização das bibliotecas CRT e CALCULA}
var x,y,z,mult: integer;      {variáveis globais}
begin
  clrscr;
  writeln('Digite o valor de x');
  readln(x);
  writeln('Digite o valor de y');
  readln(y);
  {chamada da sub-rotina SOMAR que está na biblioteca calcula}
  somar(x,y);
  writeln('Digite o valor de z');
  readln(z);
  {chamada da sub-rotina MULTIPLICAR que está na biblioteca calcula}
  mult:=multiplicar(x,y,z);
  writeln('Multiplicação = ',mult);
  readln;
end.
```

8.3 Sub-rotinas em C/C++ (funções)

Um importante recurso apresentado nas linguagens de programação é a modularização, na qual um programa pode ser particionado em sub-rotinas bastante específicas. A linguagem C/C++ possibilita a modularização por meio de funções.

Um programa escrito na linguagem C/C++ tem, no mínimo, uma função chamada *main*, por onde a execução começa. Existem também muitas outras funções predefinidas na linguagem C/C++, por exemplo: *ceil()*, *strcmp()*, *strcpy()* etc. Essas funções são adicionadas aos programas pela diretiva *#include*, no momento da 'linkedição'.

Além disso, o usuário também pode criar quantas funções quiser, dependendo do problema que estiver sendo resolvido pelo programa. As funções às vezes precisam receber valores externos, chamados parâmetros, e também podem devolver algum valor produzido para o ambiente externo, denominado retorno.

Os parâmetros são representados por uma lista de variáveis colocadas dentro de parênteses, logo após o nome da função. Caso haja retorno, a última linha da função deverá incluir o comando *return*, seguido do valor ou variável que será devolvido a quem chamou a função. O tipo do valor retornado deverá ser exatamente igual ao tipo informado antes do nome da função. Caso não haja retorno, o tipo informado antes do nome da função será *void*. Os tipos de funções são apresentados em detalhes a seguir.

As variáveis globais, ou seja, reconhecidas por todas as partes do programa, devem ser declaradas fora de todas as funções, inclusive fora da função *main*. As variáveis locais devem ser declaradas dentro das sub-rotinas e são reconhecidas apenas na sub-rotina onde foram declaradas.

A seguir, um programa exemplo, que soma dois números digitados pelo usuário, será utilizado para demonstrar os diferentes tipos de sub-rotinas.

8.3.1 Funções sem passagem de parâmetros e sem retorno

O tipo mais simples de função é aquele que não recebe nenhuma informação no momento de sua chamada e que também não repassa nenhum valor para quem a chamou. A seguir, é apresentado um exemplo de função sem parâmetros e sem retorno (a numeração das linhas não faz parte do programa) com a utilização de variáveis globais.

```

1.  #include <stdio.h>
2.  int a, b, s;    // variáveis globais
3.  void soma()
4.  {
5.      printf("\nDigite o primeiro número: ");
6.      scanf("%d%c",&a);
7.      printf("\nDigite o segundo número: ");
8.      scanf("%d%c",&b);
9.      s = a + b;
10.     printf("\nSoma = %d",s);
11. }
12. int main()
13. {
14.     soma();
15.     getchar();
16.     return 0;
17. }

```

Como já comentado na seção 8.3, a execução de programa escrito em C/C++ sempre começa pela função `main`. No exemplo, a execução se inicia na linha 12. Na linha 14, existe uma chamada à função `soma`. Nesse ponto, o fluxo da execução é desviado para a linha 3. Depois, são executadas as linhas 4 até 11. Quando a execução atinge a linha 11, a marca de final da função é encontrada. Nesse momento, o fluxo da execução retorna para a linha 15, exatamente abaixo de onde ocorreu o desvio para a função `soma`. Na linha 17 está a marca de finalização da função `main`. Assim, a execução do programa é concluída.

Devemos destacar que, no momento em que a função `soma` foi chamada, na linha 14, nenhum valor ou variável foi colocado entre parênteses, indicando que não houve passagem de parâmetros. Além disso, dentro da função `soma` não foi utilizado o comando `return`, sinalizando que ela não retornou valor para quem a chamou. Por essa razão, seu tipo é `void`.

A seguir, é apresentado um exemplo de função sem parâmetros e sem retorno (a numeração das linhas não faz parte do programa) com a utilização de variáveis locais.

```

1.  #include <stdio.h>
2.  void soma()
3.  {
4.      int a, b, s;    // variáveis locais da sub-rotina soma
5.      printf("\nDigite o primeiro número: ");
6.      scanf("%d%c",&a);
7.      printf("\nDigite o segundo número: ");
8.      scanf("%d%c",&b);
9.      s = a + b;
10.     printf("\nSoma = %d",s);
11. }
12. int main()
13. {
14.     soma();
15.     getchar();
16.     return 0;
17. }

```

No exemplo anterior, a execução se inicia na linha 12. Na linha 14, existe uma chamada à função `soma`. Nesse ponto, o fluxo da execução é desviado para a linha 2. Depois, são executadas as linhas 2 até 11. Quando a execução atinge a linha 11, a marca de final da função é encontrada. Nesse momento, o fluxo da execução retorna para a linha 15, exatamente abaixo de onde ocorreu o desvio para a função `soma`. Na linha 17 está a marca de finalização da função `main`. Assim, a execução do programa é concluída.

8.3.2 Funções com passagem de parâmetros e sem retorno

O segundo tipo de função é representado por aquelas que recebem valores no momento em que são chamadas (parâmetros), mas que, no final, não devolvem valor para quem as chamou (retorno). A seguir, é mostrado um exemplo (a numeração das linhas não faz parte do programa, servindo apenas para facilitar a explicação).

```

1.  #include <stdio.h>
2.  void soma(int a, int b)
3.  {
4.      int s;          // variável local da sub-rotina soma
5.      s = a + b;
6.      printf("\nSoma = %d",s);
7.  }
8.  int main()
9.  {
10.     int a, b;        // variáveis locais da sub-rotina main
11.     printf("\nDigite o primeiro número: ");
12.     scanf("%d%c",&a);
13.     printf("\nDigite o segundo número: ");
14.     scanf("%d%c",&b);
15.     soma(a,b);
16.     getchar();
17.     return 0;
18. }
```

No exemplo anterior, a execução se inicia na linha 8 e as linhas 9 a 15 são executadas. Na linha 15, existe uma chamada à função soma. Nesse ponto, o fluxo da execução é desviado para a linha 2. Depois, são executadas as linhas 2 até 7. Quando a execução atinge a linha 7, a marca de final da função é encontrada. Nesse momento, o fluxo da execução retorna para a linha 16, exatamente abaixo de onde ocorreu o desvio para a função soma. Na linha 18 está a marca de finalização da função main. Assim, a execução do programa é concluída.

Devemos destacar que, no momento em que a função soma foi chamada, na linha 15, duas variáveis foram colocadas entre parênteses, indicando que houve passagem de parâmetros. Os valores dessas variáveis são copiados para as variáveis a e b, descritas no cabeçalho da função, na linha 2, sendo assim variáveis locais da função soma. Além disso, dentro da função soma não foi utilizado o comando return, indicando que ela não retornou valor para quem a chamou. Por essa razão, seu tipo foi definido como void.

8.3.3 Funções sem passagem de parâmetros e com retorno

O terceiro tipo de função é representado por aquelas que não recebem valores no momento em que são chamadas (parâmetros), mas que, no final, devolvem um valor para quem as chamou (retorno). A seguir, é apresentado um exemplo (a numeração das linhas não faz parte do programa, servindo apenas para facilitar a explicação).

```

1.  #include <stdio.h>
2.  int soma()
3.  {
4.      int a, b, s;    // variáveis locais da sub-rotina soma
5.      printf("\nDigite o primeiro número: ");
6.      scanf("%d%c",&a);
7.      printf("\nDigite o segundo número: ");
8.      scanf("%d%c",&b);
9.      s = a + b;
10.     return s;        // retorno da sub-rotina soma
11. }
```

```

12.   int main()
13.   {
14.       int s;           // variável local da sub-rotina main
15.       s = soma();      // chamada da sub-rotina soma
16.       printf("\nSoma = %d",s);
17.       getchar();
18.       return 0;
19.   }

```

No exemplo anterior, a execução se inicia na linha 12 e as linhas 13 a 15 são executadas. Na linha 15, existe uma chamada à função `soma`. Nesse ponto, o fluxo da execução é desviado para a linha 2. Depois, são executadas as linhas 2 até 10. Quando a execução atinge a linha 10, a marca de retorno da função é encontrada. Nesse momento, o fluxo da execução retorna para a linha 15, atribuindo o valor retornado à variável `s`. Por fim, executa as linhas 16 a 19. Na linha 19 está a marca de finalização da função `main`. Assim, a execução do programa é concluída.

Devemos destacar que, no momento em que a função `soma` foi chamada, na linha 15, nenhum valor ou variável foi colocado entre parênteses, o que indica que não houve passagem de parâmetros. Além disso, dentro da função `soma`, foi utilizado o comando `return s`, significando que o valor da variável `s` foi devolvido a quem a chamou. Por essa razão, o tipo da função é `int`, exatamente igual ao tipo do valor retornado.

8.3.4 Funções com passagem de parâmetros e com retorno

O quarto tipo de função é representado por aquelas que recebem valores no momento em que são chamadas (parâmetros) e que, no final, devolvem um valor para quem as chamou (retorno). A seguir, é apresentado um exemplo (a numeração das linhas não faz parte do programa, servindo apenas para facilitar a explicação).

```

1.   #include <stdio.h>
2.   int soma(int a, int b)
3.   {
4.       return a + b;
5.   }
6.   int main()
7.   {
8.       int a, b, s;
9.       printf("\nDigite o primeiro número: ");
10.      scanf("%d%c",&a);
11.      printf("\nDigite o segundo número: ");
12.      scanf("%d%c",&b);
13.      s = soma(a,b);
14.      printf("\nSoma = %d",s);
15.      getchar();
16.      return 0;
17.  }

```

No exemplo anterior, a execução iniciou na linha 6. A partir daí, são executadas sequencialmente as linhas 7 a 13. Nas linhas 10 e 12, dois valores são recebidos e armazenados nas variáveis `a` e `b`. Chegando à linha 13, o fluxo de execução é desviado para a função `soma`, levando para lá os valores das variáveis `a` e `b`. Serão, então, executadas as linhas 2 a 4. Ao chegar à linha 4, o comando `return` é encontrado. Isso indica que a execução da função chegou ao fim e que o valor da operação `a + b` será devolvido para quem a chamou. O fluxo de execução retorna à função `main`, na linha 13, e o valor retornado é atribuído à variável `s`. Depois disso, as linhas 14 a 17 são executadas e o programa chega ao fim.

Devemos destacar que, no momento em que a função `soma` foi chamada, na linha 13, duas variáveis foram colocadas entre parênteses, indicando que houve passagem de parâmetros. Assim, os valores dessas variáveis são copiados, respectivamente, para as variáveis `a` e `b`, descritas no cabeçalho da função, na linha 2. Além disso, dentro da função `soma` foi utilizado o comando `return a + b`, sinalizando que o valor

da operação $a + b$ será devolvido a quem a chamou. Por essa razão, o tipo da função é exatamente igual ao tipo do valor retornado, ou seja, `int`.

Observação

Em qualquer programa, podemos escrever funções antes ou depois da função `main`. Se optarmos por escrevê-las antes, nenhum cuidado especial será necessário. Porém, se optarmos por escrevê-las abaixo da função `main`, deveremos fazer uso dos protótipos de função. Protótipo de uma função é uma linha exatamente igual ao cabeçalho da função (terminando com um ponto e vírgula) que sempre deverá ser escrita antes da função `main`. Essa linha é responsável por informar ao compilador quais outras funções serão encontradas ao término da `main`. Observe o exemplo a seguir.

```

1.  #include <stdio.h>
2.  int soma(int a, int b); // esta linha descreve o protótipo da função
3.  int main()
4.  {
5.      int a, b, s;
6.      printf("\nDigite o primeiro número: ");
7.      scanf("%d%c",&a);
8.      printf("\nDigite o segundo número: ");
9.      scanf("%d%c",&b);
10.     s = soma(a,b);
11.     printf("\nSoma = %d",s);
12.     getchar();
13.     return 0;
14. }
15. int soma(int a, int b)
16. {
17.     return a + b;
18. }
```

8.3.5 Passagem de parâmetros por valor

Passagem de parâmetros por valor significa que a função trabalhará com cópias dos valores passados no momento de sua chamada. Para entender melhor esse processo, observe o programa a seguir (a numeração das linhas não faz parte do programa, servindo apenas para facilitar a explicação).

```

1. #include <stdio.h>
2. int soma_dobro(int a, int b);
3. int main()
4. {
5.     int x, y, res;
6.     printf("\nDigite o primeiro número: ");
7.     scanf("%d%c",&x);
8.     printf("\nDigite o segundo número: ");
9.     scanf("%d%c",&y);
10.    res = soma_dobro(x,y);
11.    printf("\nA soma do dobro dos números %d e %d = %d",x,y,res);
12.    getchar();
13.    return 0;
14. }
15. int soma_dobro(int a, int b)
16. {
```

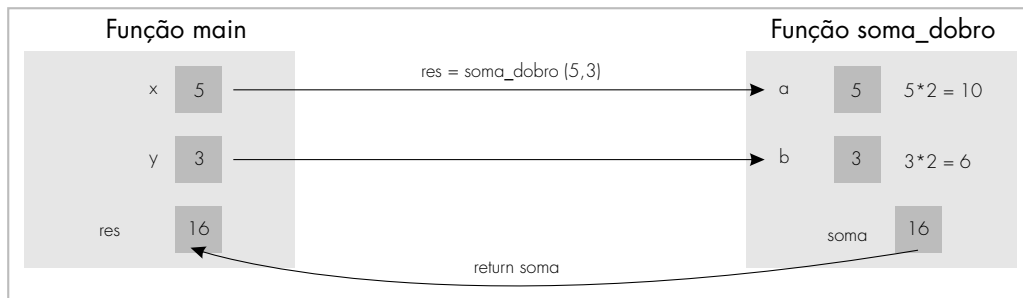
```

17.  int soma;
18.  a = 2 * a;
19.  b = 2 * b;
20.  soma = a + b;
21.  return soma;
22. }

```

Na Figura 8.1 é feita uma representação gráfica de como se dá uma passagem de parâmetros por valor, apresentada no programa anterior. Estamos supondo que os valores armazenados nas variáveis *x* e *y*, por meio da execução das linhas 7 e 9, tenham sido, respectivamente, 5 e 3. Quando a linha 10 é executada, esses valores são copiados para as variáveis *a* e *b* (pertencentes à função *soma_dobro*). Depois disso, os valores de *a* e *b* são multiplicados por 2, nas linhas 18 e 19, e depois, na linha 20, é realizada a soma. O resultado dessa soma é devolvido à função *main* pela execução da linha 21, onde o valor calculado recai sobre a variável *res* (retorno à linha 10).

Figura 8.1 Representação gráfica da passagem de parâmetros por valor.



No momento em que a função *soma_dobro* chega ao fim, as variáveis *a*, *b* e *soma* são destruídas e, portanto, as alterações realizadas pelas multiplicações por 2 são perdidas, ou seja, *x* continua valendo 5 e *y* continua valendo 3.

8.3.6 Passagem de parâmetros por referência

Passagem de parâmetros por referência significa que os parâmetros passados para uma função correspondem a endereços de memória ocupados por variáveis. Dessa maneira, toda vez que for necessário acessar determinado valor, isso será feito por meio de referência, ou seja, apontamento ao seu endereço.

```

1.  #include <stdio.h>
2.  int soma_dobro(int *a, int *b);
3.  int main()
4.  {
5.      int x, y, res;
6.      printf("\nDigite o primeiro número: ");
7.      scanf("%d%c",&x);
8.      printf("\nDigite o segundo número: ");
9.      scanf("%d%c",&y);
10.     res = soma_dobro(&x,&y);
11.     printf("\nA soma dos números %d e %d = %d",x,y,res);
12.     getchar();
13.     return 0;
14. }
15. int soma_dobro(int *a, int *b)
16. {
17.     int soma;
18.     *a = 2*(*a);

```



```

19.    *b = 2*(*b);
20.    soma = *a + *b;
21.    return soma;
22.    }

```

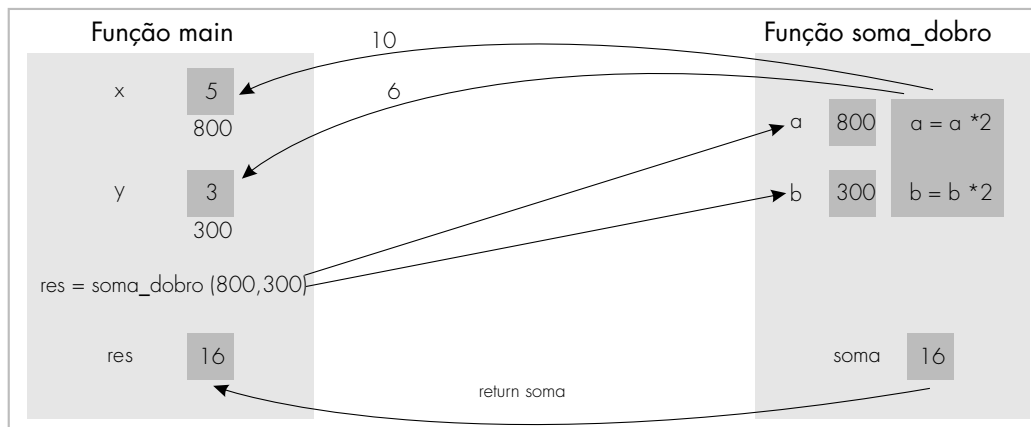
A Figura 8.2 representa graficamente o que acontece durante a execução do programa apresentado, onde ocorre a passagem de parâmetros por referência à função `soma_dobro`.

Nas linhas 7 e 9 são lidos, respectivamente, os valores para as variáveis `x` e `y` (como exemplo, supomos que tenham sido digitados os valores 5 e 3). Entretanto, quando a função `soma_dobro` é chamada, na linha 10, são passados como parâmetros para a função os endereços de memória ocupados pelas variáveis `x` e `y` (isso é feito pelo operador `&` que obtém o endereço de memória de uma variável), ou seja, pelo nosso exemplo, os valores 800 (endereço ocupado por `x`) e 300 (endereço ocupado por `y`). Dessa maneira, os valores que recaem sobre as variáveis `a` e `b` (da função) são, respectivamente, 800 e 300 (isso é correto, uma vez que `a` e `b` são ponteiros para `int`).

Nas linhas 18 e 19, os valores 5 e 3 são multiplicados por 2. Nesse momento, ocorre a ‘referência’ aos endereços de memória 800 e 300, para que sejam obtidos os valores iniciais e, após a realização das multiplicações, os valores sejam alterados. Dessa forma, no endereço 800 passamos a ter o valor 10, e no endereço 300 passamos a ter o valor 6. Na linha 20, é realizada a soma dos valores que estão nos endereços especificados por `a` e `b` (que já foram multiplicados por 2). Por fim, na linha 21, o resultado da soma é devolvido à função `main`, recaindo sobre a variável `res` (linha 10) e encerrando a função `soma_dobro`.

Quando a função `soma_dobro` chega ao fim, as variáveis `a`, `b` e `soma` são destruídas. Entretanto, as alterações decorrentes das multiplicações feitas são mantidas, pois cada alteração fez referência a endereços de memória que estavam fora da área destinada à função. Assim, após a função `soma_dobro`, o valor de `x` será 10 e o de `y` será 6.

Figura 8.2 Representação gráfica da passagem de parâmetros por referência.



Observação

A linguagem C/C++ não permite que vetores e matrizes sejam passados na íntegra como parâmetro para uma função. Para resolver esse problema, deve-se passar apenas o endereço da posição inicial do vetor ou da matriz. Esse endereço é obtido utilizando-se o nome do vetor (ou da matriz) sem o índice entre colchetes. Isso quer dizer que é possível passar um vetor para uma função somente se essa passagem for por referência. Observe o exemplo (a numeração das linhas não faz parte do programa, servindo apenas para facilitar a explicação).

```

1. #include <stdio.h>
2. void soma_linhas(float m[][5], float v[])
3. {

```

```

4.  int i, j;
5.  for (i = 0; i < 3; i++)
6.  {
7.      for (j = 0; j < 5; j++)
8.      {
9.          v[i] = v[i] + m[i][j];
10.     }
11. }
12.}
13.int main()
14.{
15. int i, j;
16. float mat[3][5], vet[3];
17. for (i = 0; i < 3; i++)
18. {
19.     vet[i] = 0;
20.     for (j = 0; j < 5; j++)
21.     {
22.         printf("\nDigite o elemento %d - %d:", i, j);
23.         scanf("%f%c", &mat[i][j]);
24.     }
25. }
26.soma_linhas(mat, vet);
27.for (i = 0; i < 3; i++)
28.{
29.    printf("\nSoma da linha %d = %f", i, vet[i]);
30.}
31.getchar();
32.return 0;
33.}

```

A execução desse programa começa na linha 13, com a função `main`. Na função `main`, são declaradas algumas variáveis, dentre elas, `mat` e `vet`. A variável `mat` representa uma matriz bidimensional com 3 linhas e 5 colunas para armazenar números reais. A variável `vet` representa um vetor com 3 posições para armazenar a soma dos números de cada linha da matriz `mat`. Da linha 17 à 25, a matriz `mat` é preenchida com números inseridos pelo usuário. Aproveitando essas estruturas de repetição, o vetor `vet` tem todas as suas posições inicializadas com zero, na linha 19.

Duas linhas merecem atenção especial: 26 e 2. A linha 26 está chamando a função `soma_linhas`, passando como parâmetros a matriz `mat` e o vetor `vet`. Observe, entretanto, que essas duas variáveis não estão acompanhadas de colchetes (`[]`). Assim, quando o nome de uma matriz ou vetor for utilizado sem apresentar colchetes contendo um índice, isso significa que estamos usando o endereço de memória ocupado pela posição 0 do vetor ou pela posição 0X0 da matriz.

Como endereços de memória só podem ser atribuídos a ponteiros, observe a linha 2, onde está o cabeçalho da função `soma_linhas`. Nela, pode-se ver que a função recebe dois parâmetros: `m[][5]` e `v[]`. Assim, toda vez que encontrar um vetor com colchetes vazios ou uma matriz com os colchetes da primeira dimensão vazios, entenda que eles são variáveis ponteiros que guardam os endereços iniciais das variáveis.

A partir daí, o programa consegue percorrer o vetor e a matriz normalmente, da linha 5 à 11. Quando a função `soma_linhas` chegar ao fim, o fluxo de execução retornará para a linha 27 e o vetor `vet`, que entrou na função `soma_linhas` contendo zero em todas as suas posições, voltará com o somatório dos números de cada linha da matriz `mat`. Esses valores serão mostrados nas linhas 27 à 29. O programa, então, é finalizado.

8.3.7 Sub-rotinas em arquivos separados (bibliotecas)

Na linguagem C/C++ existem algumas bibliotecas já implementadas e dentro destas existem inúmeras funções. Por exemplo, na biblioteca `stdio.h`, existem as funções `scanf`, `getchar`, `printf`, entre outras. Assim, nesta seção, discutiremos a possibilidade de criar bibliotecas próprias com uma ou várias funções.

Para criar uma biblioteca é necessário gerar um novo arquivo e dentro dele pôr o código de todas as funções que farão parte dessa biblioteca. Esse arquivo deve ser salvo com extensão `.h` e deve ser compilado normalmente.

O exemplo a seguir é uma biblioteca denominada `rotinas.h` e possui três sub-rotinas.

```
#include <stdio.h>
void sub_rotina1()
{
    printf("mostrando uma mensagem");
}
void sub_rotina2()
{
    int a,b,c;
    printf("Digite o valor de a: ");
    scanf("%d%c",&a);
    printf("Digite o valor de b: ");
    scanf("%d%c",&b);
    c = a - b;
    printf("Resultado = %d",c);
}
int sub_rotina3(int x,int y)
{
    int res;
    res = x * y;
    return res;
}
```

O código a seguir mostra um programa que faz uso da biblioteca criada anteriormente.

```
1.  #include <stdio.h>
2.  #include "c:\teste\rotinas.h"
3.  int main()
4.  {
5.      int num1, num2, res;
6.      sub_rotina1();
7.      sub_rotina2();
8.      printf("Digite um número: ");
9.      scanf("%d%c",&num1);
10.     printf("Digite outro número: ");
11.     scanf("%d%c",&num2);
12.     res = sub_rotina3(num1, num2);
13.     printf("resultado = %d",res);
14.     getchar();
15.     return 0;
16. }
```

Quando uma nova biblioteca for utilizada em um programa e seu arquivo estiver salvo no diretório-padrão das bibliotecas da linguagem C/C++, basta chamá-la por meio da diretiva `#include <biblioteca.h>`. Quando a nova biblioteca está em outro diretório, sua chamada deve obedecer à seguinte sintaxe: `#include "caminho\nome_da_biblioteca.h"`, como mostra a linha 2 do código dado anteriormente.

8.4 Sub-rotinas em JAVA (métodos)

JAVA é uma linguagem que dá suporte ao paradigma orientado a objeto. Assim, todos os programas devem fazer uso de classes. Uma classe pode ser entendida como um tipo de dado capaz de armazenar diversas informações e também várias funções para manipular adequadamente essas informações. Seguindo o paradigma orientado a objetos, essas informações são chamadas **atributos**, e as funções, **métodos**.

Como acontece com todas as linguagens de programação, o usuário poderá utilizar métodos de classes já existentes (o Capítulo 9 abordará, por exemplo, vários métodos especificamente criados para trabalhar com cadeias de caracteres), como também poderá criar quantos métodos forem necessários para a resolução do problema. O capítulo 12 apresenta uma introdução à programação orientada a objetos.

Cada método pode receber diversos valores, os parâmetros, e pode devolver um valor, o retorno. Dessa maneira, quando se especifica um método, deve-se deixar claro qual será o tipo de retorno e quais são os parâmetros necessários para a sua execução. A chamada a um método normalmente requer que seja informado o objeto ou a classe que o executará. A sintaxe para chamada de um método é apresentada a seguir.

```
[retorno = ]nomeDoObjeto.nomeDoMétodo([ listaDeParametros ]);
```

ou

```
[retorno = ]nomeDaClasse.nomeDoMétodo([ listaDeParametros ]);
```

Observe que o retorno e a listaDeParametros aparecem entre colchetes, indicando que são opcionais.

A seguir, são apresentados vários exemplos de métodos. É importante observar que, em todos eles, aparecerão as palavras `public` e `static`. Essas palavras são chamadas modificadores, ou seja, definem características complementares aos métodos. Outros modificadores são: `private`, `protected`, `abstract` e `final`.

O modificador `public` quer dizer que o método poderá ser chamado por qualquer classe, e o modificador `static` indica que esse método existirá e poderá ser executado mesmo se nenhum objeto da classe onde estiver inserido for criado.

Os parâmetros são representados por uma lista de variáveis colocadas dentro de parênteses, logo após o nome do método. Caso haja retorno, a última linha do método deverá incluir o comando `return`, seguido do valor ou variável que será devolvido a quem chamou o método. O tipo do valor retornado deverá ser exatamente igual ao tipo informado antes do nome do método. Caso não haja retorno, deverá ser digitada a palavra `void`. Os tipos de métodos são apresentados em detalhes a seguir.

As variáveis globais, ou seja, aquelas que são reconhecidas por todas as partes do programa, devem ser declaradas fora de todos os métodos, inclusive fora do método `main`. As variáveis locais devem ser declaradas dentro dos métodos e são reconhecidas apenas no método onde foram declaradas.

A seguir, um programa exemplo que soma dois números digitados pelo usuário será utilizado para demonstrar os diferentes tipos de métodos.

8.4.1 Métodos sem passagem de parâmetros e sem retorno

O tipo mais simples de método é aquele que não recebe nenhuma informação no momento de sua chamada e também não repassa nenhum valor para quem o chamou. A seguir, é apresentado um exemplo de método sem parâmetros e sem retorno (a numeração das linhas não faz parte do programa) com a utilização de variáveis globais.

```
1.    import java.util.Scanner;
2.    public class Exemplo
3.    {
4.        static int a, b, s; // variáveis globais
5.        public static void main(String args[])
6.        {
7.            soma();           // chamada do método soma()
8.        }
9.        public static void soma()
10.       {
11.           Scanner e = new Scanner(System.in);
12.           System.out.println("Digite o primeiro número: ");
```

```

13.      a = e.nextInt();
14.      System.out.println("Digite o segundo número: ");
15.      b = e.nextInt();
16.      s = a + b;
17.      System.out.println("Soma = " + s);
18.  }
19.  }

```

Como acontece nos programas escritos na linguagem C/C++, a execução de um programa JAVA também começa pelo método `main`. No exemplo, a execução iniciou na linha 5. Na linha 7, existe uma chamada ao método `soma`. Nesse ponto, o fluxo da execução é desviado para a linha 9. Depois, são executadas as linhas 10 a 18. Quando a execução atingir a linha 18, a marca de final de método será encontrada. Nesse momento, o fluxo da execução retorna para a linha 8, exatamente abaixo de onde ocorreu o desvio para o método. Nessa linha está a marca de finalização do método `main`. Desse modo, a execução do programa é concluída.

Devemos destacar que, no momento em que o método `soma` foi chamado, na linha 7, nenhum valor ou variável foi colocado entre parênteses, o que indica que não houve passagem de parâmetros. Além disso, dentro do método `soma` não foi utilizado o comando `return`, sinalizando que ele não retornou valor para quem o chamou. Por essa razão, seu tipo é `void`.

A seguir, é apresentado um exemplo de método sem parâmetros e sem retorno (a numeração das linhas não faz parte do programa) com a utilização de variáveis locais.

```

1. import java.util.Scanner;
2. public class Exemplo
3. {
4.     public static void main(String args[])
5.     {
6.         soma();                // chamada do método soma()
7.     }
8.     public static void soma()
9.     {
10.        int a, b, s;           // variáveis locais
11.        Scanner e = new Scanner(System.in);
12.        System.out.println("Digite o primeiro número: ");
13.        a = e.nextInt();
14.        System.out.println("Digite o segundo número: ");
15.        b = e.nextInt();
16.        s = a + b;
17.        System.out.println("Soma = " + s);
18.    }
19. }

```

No exemplo anterior, a execução iniciou na linha 4. Na linha 6, existe uma chamada ao método `soma`. Nesse ponto, o fluxo da execução é desviado para a linha 8. Depois, são executadas as linhas 9 a 17. Quando a execução atingir a linha 18, a marca de final de método será encontrada. Nesse momento, o fluxo da execução retorna para a linha 7, exatamente abaixo de onde ocorreu o desvio para o método. Nessa linha está a marca de finalização do método `main`. Desse modo, a execução do programa é concluída.

Devemos destacar que, no momento em que o método `soma` foi chamado, na linha 6, nenhum valor ou variável foi colocado entre parênteses, o que indica que não houve passagem de parâmetros. Além disso, dentro do método `soma` não foi utilizado o comando `return`, sinalizando que ele não retornou valor para quem o chamou. Por essa razão, seu tipo é `void`.

8.4.2 Métodos com passagem de parâmetros e sem retorno

O segundo tipo de método é representado por aqueles que recebem valores no momento em que são chamados (parâmetros), mas que, no final, não devolvem valor para quem os chamou (retorno). A seguir, é apresentado um exemplo (a numeração das linhas não faz parte do programa, servindo apenas para facilitar a explicação).

```

1. import java.util.Scanner;
2. public class Exemplo
3. {
4.     public static void main(String args[])
5.     {
6.         int a, b;           // variáveis locais do método main
7.         Scanner e = new Scanner(System.in);
8.         System.out.println("Digite o primeiro número: ");
9.         a = e.nextInt();
10.        System.out.println("Digite o segundo número: ");
11.        b = e.nextInt();
12.        soma(a,b);           // chamada do método soma()
13.    }
14.    public static void soma(int a,int b)
15.    {
16.        int s;               // variável local do método soma()
17.        s = a + b;
18.        System.out.println("Soma = " + s);
19.    }
20. }
```

Como acontece nos programas escritos na linguagem C/C++, a execução de um programa JAVA também começa pelo método main. No exemplo, a execução teve início na linha 4. A partir daí, são executadas sequencialmente as linhas 5 a 12. Nas linhas 9 e 11, dois valores são recebidos e armazenados nas variáveis *a* e *b*. Chegando à linha 12, o fluxo de execução é desviado para o método *soma*, na linha 14, levando para lá os valores das variáveis *a* e *b*. Serão, então, executadas as linhas 14 a 19, onde está a marca de encerramento do método. O fluxo de execução retorna ao método main, na linha 13, imediatamente abaixo do ponto de chamada ao método *soma*. Desse modo, a execução do programa é concluída.

Devemos destacar que, no momento em que o método *soma* foi chamado, na linha 12, duas variáveis foram colocadas entre parênteses, o que significa que houve passagem de parâmetros. Os valores dessas variáveis são copiados para as variáveis *a* e *b*, descritas no cabeçalho do método, na linha 14. Além disso, dentro do método *soma* não foi utilizado o comando *return*, indicando que ele não retornou valor para quem o chamou. Por essa razão, seu tipo foi definido como *void*.

8.4.3 Métodos sem passagem de parâmetros e com retorno

O terceiro tipo de método é representado por aqueles que não recebem valores no momento em que são chamados (parâmetros), mas que, no final, devolvem um valor para quem os chamou (retorno). A seguir, é apresentado um exemplo (a numeração das linhas não faz parte do programa, servindo apenas para facilitar a explicação).

```

1.     import java.util.Scanner;
2.     public class Exemplo
3.     {
4.         public static void main(String args[])
5.         {
6.             int s;           // variável local do método main
7.             s = soma();       // chamada do método soma()
```

```

8.      System.out.println("Soma = " + s);
9.    }
10.   public static int soma()
11.   {
12.     int a, b, s;           // variáveis locais do método soma()
13.     Scanner e = new Scanner(System.in);
14.     System.out.println("Digite o primeiro número: ");
15.     a = e.nextInt();
16.     System.out.println("Digite o segundo número: ");
17.     b = e.nextInt();
18.     s = a + b;
19.     return s;
20.   }
21. }

```

No exemplo anterior, a execução iniciou na linha 4 e, sequencialmente, as linhas 5, 6 e 7 foram executadas. Na linha 7, existe uma chamada ao método `soma`. Nesse ponto, o fluxo da execução é desviado para a linha 10. Depois, são executadas as linhas 11 a 18. Quando a execução atingir a linha 19, o comando `return` é encontrado. Isso significa que a execução do método chegou ao fim e que o conteúdo da variável `s` será devolvido para quem o chamou. O fluxo de execução retorna ao método `main`, na linha 7, e o valor retornado é atribuído à variável `s`. Depois disso, as linhas 8 e 9 são executadas e o programa chega ao fim.

Devemos destacar que, no momento em que o método `soma` foi chamado, na linha 7, nenhum valor ou variável foi colocado entre parênteses, sinalizando que não houve passagem de parâmetros. Além disso, dentro do método `soma` foi utilizado o comando `return s`, o que indica que o valor da variável `s` será devolvido a quem o chamou. Por essa razão, o tipo do método é `int`, exatamente igual ao tipo do valor retornado.

8.4.4 Métodos com passagem de parâmetros e com retorno

O quarto tipo de método é representado por aqueles que recebem valores no momento em que são chamados (parâmetros) e que, no final, devolvem um valor para quem os chamou (retorno). A seguir, é apresentado um exemplo (a numeração das linhas não faz parte do programa, servindo apenas para facilitar a explicação).

```

1. import java.util.Scanner;
2. public class Exemplo
3. {
4.   public static void main(String args[])
5.   {
6.     int a, b, s;           // variáveis locais do método main
7.     Scanner e = new Scanner(System.in);
8.     System.out.println("Digite o primeiro número: ");
9.     a = e.nextInt();
10.    System.out.println("Digite o segundo número: ");
11.    b = e.nextInt();
12.    s = soma(a,b);          // chamada do método soma()
13.    System.out.println("Soma = " + s);
14.  }
15.  public static int soma(int a,int b)
16.  {
17.    return a + b;
18.  }
19. }

```

No exemplo anterior, a execução teve início na linha 4. A partir daí, são executadas, sequencialmente, as linhas 5, 6, 7, 8, 9, 10 e 11. Nas linhas 9 e 11, dois valores são recebidos e armazenados nas variáveis *a* e *b*. Chegando à linha 12, o fluxo de execução é desviado para o método *soma*, levando para lá os valores das variáveis *a* e *b*. Serão, então, executadas as linhas 15, 16 e 17. Ao chegar à linha 17, o comando *return* é encontrado. Isso significa que a execução do método chegou ao fim e que o valor da operação *a + b* será devolvido para quem o chamou. O fluxo de execução retorna ao método *main*, na linha 12, e o valor retornado é atribuído à variável *s*. Depois disso, as linhas 13 e 14 são executadas e o programa chega ao fim.

Devemos destacar que, no momento em que o método *soma* foi chamado, na linha 12, duas variáveis foram colocadas entre parênteses, indicando que houve passagem de parâmetros. Assim, os valores dessas variáveis são copiados, respectivamente, para as variáveis *a* e *b*, descritas no cabeçalho do método, na linha 15. Além disso, dentro do método *soma* foi utilizado o comando *return a + b*, o que significa que um valor foi devolvido a quem o chamou. Por essa razão, o tipo do método é *int*, exatamente igual ao tipo do valor retornado.

8.4.5 Passagem de parâmetros por valor e por referência

Uma peculiaridade da linguagem JAVA é que todas as variáveis que não forem de tipos primitivos serão consideradas referência. Isso quer dizer que a variável contém apenas o endereço de memória onde a informação completa foi gravada.

Assim, toda vez que um tipo primitivo for passado como parâmetro, essa passagem será feita por valor, ou seja, será criada uma cópia completa da informação dentro do método. Toda vez que um tipo não primitivo for passado como parâmetro, a passagem será por referência, isto é, será feita uma cópia apenas do endereço onde a informação está gravada. As seções 8.3.5 e 8.3.6, apresentadas anteriormente, descrevem em detalhe esses dois tipos de passagem de parâmetros. Os princípios lá descritos são os mesmos em todas as linguagens de programação.



Observação

Na linguagem JAVA, vetores e matrizes não são considerados tipos primitivos e, assim, são passados como parâmetros por meio de referência. Observe o exemplo (a numeração das linhas não faz parte do programa, servindo apenas para facilitar a explicação).

```

1. import java.util.Scanner;
2.     public class Exemplo
3.     {
4.         public static void main(String args[])
5.         { Scanner e;
6.             int i, j;
7.             float mat[][],vet[];
8.             mat = new float[3][5];
9.             vet = new float[3];
10.            e = new Scanner(System.in);
11.            for (i=0;i<3;i++)
12.            { vet[i] = 0;
13.                for (j=0;j<5;j++)
14.                { System.out.println("Digite o elemento " + i + "-" + j + " : ");
15.                    mat[i][j] = e.nextFloat();
16.                }
17.            }
18.            soma_linhas(mat, vet);
19.            for (i=0; i<3; i++)
20.            { System.out.println ("Soma da linha " + i + " = " + vet[i]);
21.            }
22.        }

```



```

23.     public static void soma_linhas(float m[][], float v[])
24.     { int i, j;
25.         for (i=0;i<3;i++)
26.         { for (j=0;j<5;j++)
27.             { v[i] = v[i] + m[i][j];
28.             }
29.         }
30.     }
31. }

```

A execução desse programa começa na linha 4 com o método `main`. Nesse método, são declaradas algumas variáveis, dentre elas, `mat` e `vet`. A variável `mat` representa uma matriz bidimensional com 3 linhas e 5 colunas para armazenar números reais. A variável `vet` representa um vetor com 3 posições para armazenar a soma dos números de cada linha da matriz `mat`. Da linha 11 à 17, a matriz `mat` é preenchida com números digitados pelo usuário. Aproveitando essas estruturas de repetição, o vetor `vet` tem todas as suas posições inicializadas com zero, na linha 12.

Duas linhas merecem atenção especial, a 18 e a 23. A linha 18 está chamando o método `soma_linhas`, passando como parâmetros a matriz `mat` e o vetor `vet`. Observe, entretanto, que essas duas variáveis não estão acompanhadas de colchetes. Isso significa que, quando o nome de uma matriz ou vetor for usado sem apresentar colchetes contendo um índice, estamos usando o endereço de memória ocupado pela posição 0 do vetor ou pela posição 0X0 da matriz.

Como endereços de memória só podem ser atribuídos a ponteiros, observe a linha 23, onde está o cabeçalho do método `soma_linhas`. Nela, pode-se ver que o método está preparado para receber dois valores que serão atribuídos a `m[][]` e `v[]`. Assim, toda vez que encontrar um vetor e matriz com colchetes vazios, entenda que eles são variáveis ponteiros que guardam os endereços iniciais das variáveis.

A partir daí, o programa consegue percorrer o vetor e a matriz normalmente, da linha 25 à 29. Quando o método `soma_linhas` chegar ao fim, o fluxo de execução retornará à linha 19, e o vetor `vet`, que entrou no método `soma_linhas` contendo zero em todas as suas posições, voltará com o somatório dos números de cada linha da matriz `mat`. Esses valores serão mostrados nas linhas 19 à 21. O programa, então, é finalizado na linha 22, quando o método `main` chega ao fim.

8.4.6 Métodos em arquivos separados

Na linguagem JAVA existe a possibilidade de criar métodos em arquivos separados do método `main`, o que torna possível a reutilização dos códigos. O exemplo a seguir é um arquivo salvo como `rotinas.java` e este foi compilado normalmente.

```

import java.util.Scanner;
public class rotinas
{
    public static void mensagem()
    {
        System.out.println("Mostra uma mensagem");
    }
    public static int soma()
    {
        int a, b, c;
        Scanner dado;
        dado = new Scanner(System.in);
        System.out.println("Digite o valor de a: ");
        a = dado.nextInt();
        System.out.println("Digite o valor de b: ");
    }
}

```

```

        b = dado.nextInt();
        c = a + b;
        return c;
    }
}

```

O exemplo a seguir é um programa JAVA que utiliza os métodos implementados em `rotinas.java`, por isso, na primeira linha de código, existe a importação do arquivo que tem os métodos implementados.

```

import sub_rotinas.rotinas;
public class principal
{
    public static void main(String args[])
    {
        int r;
        rotinas.mensagem();
        r = rotinas.soma();
        System.out.println("soma = "+r);
    }
}

```

EXERCÍCIOS RESOLVIDOS

1. Faça um programa contendo uma sub-rotina que retorne 1 se o número digitado for positivo ou 0 se for negativo.

ALGORITMO SOLUÇÃO:

```

ALGORITMO
DECLARE num, x NUMÉRICO
LEIA num
x ← verifica(num)
SE x = 0
ENTÃO ESCREVA "Número positivo"
SENÃO ESCREVA "Número negativo"
FIM_ALGORITMO.

SUB-ROTINA verifica(num NUMÉRICO)
DECLARE res NUMÉRICO
SE num >= 0
ENTÃO res ← 1
SENÃO res ← 0
RETORNE res
FIM_SUB_ROTINA verifica

```

PASCAL SOLUÇÃO:

\EXERC\CAP8\PASCAL\EX1.PAS e \EXERC\CAP8\PASCAL\EX1.EXE

C/C++ 1ª SOLUÇÃO: FUNÇÃO ANTES DA `main()`

\EXERC\CAP8\C++\EX1_A.CPP e \EXERC\CAP8\C++\EX1_A.EXE

2ª SOLUÇÃO: FUNÇÃO DEPOIS DA `main()`

\EXERC\CAP8\C++\EX1_B.CPP e \EXERC\CAP8\C++\EX1_B.EXE

JAVA

SOLUÇÃO:

\EXERC\CAP8\JAVA\EX1.java e \EXERC\CAP8\JAVA\EX1.class

- 2.** Faça um programa contendo uma sub-rotina que receba dois números positivos por parâmetro e retorne a soma dos N números inteiros existentes entre eles.

ALGORITMO SOLUÇÃO:

```

ALGORITMO
DECLARE num1, num2, s NUMÉRICO
LEIA num1, num2
s ← somar(num1, num2)
ESCREVA "soma = ", s
FIM_ALGORITMO.

SUB-ROTINA somar(num1, num2 NUMÉRICO)
DECLARE i, s NUMÉRICO
s ← 0
PARA i ← num1+1 ATÉ num2-1 FAÇA
    INÍCIO
        s ← s + i
    FIM
RETORNE s
FIM_SUB_ROTINA somar

```

PASCAL

SOLUÇÃO:

\EXERC\CAP8\PASCAL\EX2.PAS e \EXERC\CAP8\PASCAL\EX2.EXE

C/C++

1ª SOLUÇÃO: FUNÇÃO ANTES DA main()

\EXERC\CAP8\C++\EX2_A.CPP e \EXERC\CAP8\C++\EX2_A.EXE

2ª SOLUÇÃO: FUNÇÃO DEPOIS DA main()

\EXERC\CAP8\C++\EX2_B.CPP e \EXERC\CAP8\C++\EX2_B.EXE

JAVA

SOLUÇÃO:

\EXERC\CAP8\java\EX2.java e \EXERC\CAP8\JAVA\EX2.class

- 3.** Faça um programa contendo uma sub-rotina que receba três números inteiros a , b e c , sendo a maior que 1. A sub-rotina deverá somar todos os inteiros entre b e c que sejam divisíveis por a (inclusive b e c) e retornar o resultado para ser impresso.

ALGORITMO SOLUÇÃO:

```

ALGORITMO
DECLARE a, b, c, result NUMÉRICO
REPITA
LEIA a
ATÉ a>1
LEIA b,c
result ← divisores(a, b, c)
ESCREVA "Soma dos inteiros = ", result
FIM_ALGORITMO.

SUB-ROTINA divisores(a, b, c NUMÉRICO)
DECLARE i, s, r NUMÉRICO
s ← 0

```

```

PARA i ← b ATÉ c FAÇA
  INÍCIO
    r ← RESTO (i / a)
    SE r = 0
      ENTÃO s ← s + i
    FIM
  RETORNE s
FIM_SUB_ROTINA divisores

```

PASCAL SOLUÇÃO:

\EXERC\CAP8\PASCAL\EX3.PAS e \EXERC\CAP8\PASCAL\EX3.EXE

C/C++ 1ª SOLUÇÃO: FUNÇÃO ANTES DA main()

\EXERC\CAP8\C++\EX3_A.CPP e \EXERC\CAP8\C++\EX3_A.EXE

2ª SOLUÇÃO: FUNÇÃO DEPOIS DA main()

\EXERC\CAP8\C++\EX3_B.CPP e \EXERC\CAP8\C++\EX3_B.EXE

JAVA SOLUÇÃO:

\EXERC\CAP8\JAVA\EX3.java e \EXERC\CAP8\JAVA\EX3.class

- 4.** Faça uma sub-rotina que receba um único valor representando segundos. Essa sub-rotina deverá convertê-lo para horas, minutos e segundos. Todas as variáveis devem ser passadas como parâmetro, não havendo variáveis globais.

ALGORITMO SOLUÇÃO:

```

ALGORITMO
DECLARE seg NUMÉRICO
LEIA seg
transformacao(seg);
FIM_ALGORITMO.
SUB-ROTINA transformacao(segundos NUMÉRICO)
  DECLARE h, m, s, r NUMÉRICO
  h ← segundos / 3600
  r ← RESTO(segundos / 3600)
  m ← r / 60
  s = RESTO(r / 60)
  ESCREVA h, m, s
FIM_SUB_ROTINA transformacao

```

PASCAL SOLUÇÃO:

\EXERC\CAP8\PASCAL\EX4.PAS e \EXERC\CAP8\PASCAL\EX4.EXE

C/C++ 1ª SOLUÇÃO: FUNÇÃO ANTES DA main()

\EXERC\CAP8\C++\EX4_A.CPP e \EXERC\CAP8\C++\EX4_A.EXE

2ª SOLUÇÃO: FUNÇÃO DEPOIS DA main()

\EXERC\CAP8\C++\EX4_B.CPP e \EXERC\CAP8\C++\EX4_B.EXE

JAVA SOLUÇÃO:

\EXERC\CAP8\JAVA\EX4.java e \EXERC\CAP8\JAVA\EX4.class

- 5.** Crie um programa que receba os valores antigo e atual de um produto. Chame uma sub-rotina que determine o percentual de acréscimo entre esses valores. O resultado deverá ser mostrado no programa principal.

ALGORITMO Solução:

```

ALGORITMO
DECLARE precoantigo, precoatual, acrescimo NUMÉRICO
LEIA precoantigo
LEIA precoatual
acrescimo ← calculo_reajuste(precoantigo, precoatual)
ESCREVA acrescimo
FIM_ALGORITMO.

SUB-ROTINA calculo_reajuste(PA, PN NUMÉRICO)
DECLARE result NUMÉRICO
result ← (100 * PN - 100 * PA) / PA
RETORNE result
FIM_SUB_ROTINA calculo_reajuste

```

PASCAL

Solução:

```
\EXERC\CAP8\PASCAL\EX5.PAS e \EXERC\CAP8\PASCAL\EX5.EXE
```

C/C++

Solução:

```
\EXERC\CAP8\C++\EX5.CPP e \EXERC\CAP8\C++\EX5.EXE
```

JAVA

Solução:

```
\EXERC\CAP8\JAVA\EX5.java e \EXERC\CAP8\JAVA\EX5.class
```

- 6.** Faça uma sub-rotina que receba como parâmetro um inteiro no intervalo de 1 a 9 e mostre a seguinte tabela de multiplicação (no exemplo, n = 9):

1								
2	4							
3	6	9						
4	8	12	16					
5	10	15	20	25				
6	12	18	24	30	36			
7	14	21	28	35	42	49		
8	16	24	32	40	48	56	64	
9	18	27	36	45	54	63	72	81

ALGORITMO Solução:

```

ALGORITMO
DECLARE num NUMÉRICO
REPITA
  LEIA num
ATÉ (num >= 1) E (num <= 9)
multiplicacao(num)
FIM_ALGORITMO.

SUB-ROTINA multiplicacao(n NUMÉRICO)
DECLARE i, j NUMÉRICO
PARA i ← 1 ATÉ n FAÇA
  INÍCIO
    PARA j ← 1 ATÉ i FAÇA
      INÍCIO
        ESCREVA i * j

```

```

FIM
FIM
FIM_SUB_ROTINA multiplicacao

```

PASCAL SOLUÇÃO:

```
\EXERC\CAP8\PASCAL\EX6.PAS e \EXERC\CAP8\PASCAL\EX6.EXE
```

C/C++

1ª SOLUÇÃO: FUNÇÃO ANTES DA main()

```
\EXERC\CAP8\C++\EX6_A.CPP e \EXERC\CAP8\C++\EX6_A.EXE
```

2ª SOLUÇÃO: FUNÇÃO DEPOIS DA main()

```
\EXERC\CAP8\C++\EX6_B.CPP e \EXERC\CAP8\C++\EX6_B.EXE
```

JAVA

SOLUÇÃO:

```
\EXERC\CAP8\JAVA\EX6.java e \EXERC\CAP8\JAVA\EX6.class
```

- 7.** Elabore um programa contendo uma sub-rotina que receba as três notas de um aluno como parâmetros e uma letra. Se a letra for A, a sub-rotina deverá calcular a média aritmética das notas do aluno; se for P, deverá calcular a média ponderada, com pesos 5, 3 e 2. A média calculada deverá ser devolvida ao programa principal para, então, ser mostrada.

ALGORITMO SOLUÇÃO:

```

ALGORITMO
DECLARE nota1, nota2, nota3, m NUMÉRICO
      letra LITERAL
LEIA nota1
LEIA nota2
LEIA nota3
REPITA
    LEIA letra
ATÉ (letra = "A") OU (letra = "P")
m ← calcula_media(nota1, nota2, nota3, letra)
SE letra = "A"
    ENTÃO ESCREVA "A média aritmética " , m
    SENÃO ESCREVA "A média ponderada " , m
FIM_ALGORITMO.

SUB-ROTINA calcula_media(n1, n2, n3 NUMÉRICO, l LITERAL)
    DECLARE media NUMÉRICO
    SE l = "A"
        ENTÃO media ← (n1+n2+n3)/3
        SENÃO media ← (n1*5+n2*3+n3*2)/(5+3+2)
    RETORNE media
FIM_SUB_ROTINA calcula_media

```

PASCAL SOLUÇÃO:

```
\EXERC\CAP8\PASCAL\EX7.PAS e \EXERC\CAP8\PASCAL\EX7.EXE
```

C/C++

1ª SOLUÇÃO – FUNÇÃO ANTES DA main()

```
\EXERC\CAP8\C++\EX7_A.CPP e \EXERC\CAP8\C++\EX7_A.EXE
```

2ª SOLUÇÃO – FUNÇÃO DEPOIS DA main()

```
\EXERC\CAP8\C++\EX7_B.CPP e \EXERC\CAP8\C++\EX7_B.EXE
```

JAVA

SOLUÇÃO:

```
\EXERC\CAP8\JAVA\EX7.java e \EXERC\CAP8\JAVA\EX7.class
```

- 8.** Crie uma sub-rotina que receba como parâmetro a hora de início e a hora de término de um jogo, ambas subdivididas em dois valores distintos: horas e minutos. A sub-rotina deverá retornar a duração expressa em minutos, considerando que o tempo máximo de duração de um jogo é de 24 horas e que ele pode começar em um dia e terminar no outro.

ALGORITMO SOLUÇÃO:

```
ALGORITMO
DECLARE hora_i, min_i, hora_f, min_f, minutos NUMÉRICO
LEIA hora_i, min_i
LEIA hora_f, min_f
minutos ← calculo(hora_i, min_i, hora_f, min_f)
ESCREVA minutos
FIM_ALGORITMO.

SUB-ROTINA calculo(h_i, m_i, h_f, m_f NUMÉRICO)
DECLARE tot_h, tot_m, total NUMÉRICO
SE m_f < m_i
ENTÃO INÍCIO
    m_f ← m_f + 60
    h_f ← h_f - 1
FIM
SE h_f < h_i
ENTÃO INÍCIO
    h_f ← h_f + 24
FIM
tot_m ← m_f - m_i
tot_h ← h_f - h_i
total ← tot_h * 60 + tot_m
RETORNE total
FIM_SUB_ROTINA calculo
```

PASCAL

SOLUÇÃO:

\EXERC\CAP8\PASCAL\EX8.PAS e \EXERC\CAP8\PASCAL\EX8.EXE

C/C++

1ª SOLUÇÃO – FUNÇÃO ANTES DA main()

\EXERC\CAP8\C++\EX8_A.CPP e \EXERC\CAP8\C++\EX8_A.EXE

2ª SOLUÇÃO – FUNÇÃO DEPOIS DA main()

\EXERC\CAP8\C++\EX8_B.CPP e \EXERC\CAP8\C++\EX8_B.EXE

JAVA

SOLUÇÃO:

\EXERC\CAP8\JAVA\EX8.java e \EXERC\CAP8\JAVA\EX8.class

- 9.** Faça uma sub-rotina que leia cinco valores inteiros, determine e mostre o maior e o menor deles.

ALGORITMO SOLUÇÃO:

```
ALGORITMO
maior_menor;
FIM_ALGORITMO.

SUB-ROTINA maior_menor
DECLARE i, num, maior, menor NUMÉRICO
PARA i ← 1 ATÉ 5 FAÇA
```

```

INÍCIO
  ESCREVA "Digite o ", i, "º número: "
  LEIA num
  SE i = 1
  ENTÃO INÍCIO
    maior ← num
    menor ← num
    FIM
  SENÃO INÍCIO
    SE num > maior
    ENTÃO maior ← num
    SE num < menor
    ENTÃO menor ← num
    FIM
  FIM
  ESCREVA "O maior número digitado foi: ", maior
  ESCREVA "O menor número digitado foi: ", menor
  FIM_SUB_ROTINA maior_menor

```

PASCAL SOLUÇÃO:

```
\EXERC\CAP8\PASCAL\EX9.PAS e \EXERC\CAP8\PASCAL\EX9.EXE
```

C/C++ 1ª SOLUÇÃO – FUNÇÃO ANTES DA `main()`

```
\EXERC\CAP8\C++\EX9_A.CPP e \EXERC\CAP8\C++\EX9_A.EXE
```

2ª SOLUÇÃO – FUNÇÃO DEPOIS DA `main()`

```
\EXERC\CAP8\C++\EX9_B.CPP e \EXERC\CAP8\C++\EX9_B.EXE
```

JAVA SOLUÇÃO:

```
\EXERC\CAP8\JAVA\EX9.java e \EXERC\CAP8\JAVA\EX9.class
```

- 10.** Crie uma sub-rotina que receba como parâmetro um valor inteiro e positivo N e retorne o valor de S , obtido pelo seguinte cálculo:

$$S = 1 + 1/1! + 1/2! + 1/3! + \dots + 1/N!$$

ALGORITMO SOLUÇÃO:

```

ALGORITMO
DECLARE num, s NUMÉRICO
LEIA num
s ← sequencia(num)
ESCREVA s
FIM_ALGORITMO.

SUB-ROTINA sequencia(n NUMÉRICO)
DECLARE a, b, f, seq NUMÉRICO
seq ← 1
PARA a ← 1 ATÉ n FAÇA
  INÍCIO
    f ← 1
    PARA b ← 1 ATÉ a FAÇA
      INÍCIO
        f ← f * b
      FIM
    seq ← seq + 1/f
  FIM
FIM


```





```

    seq ← seq + 1 / f
FIM
RETORNE seq
FIM_SUB_ROTINA sequencia

```

PASCAL SOLUÇÃO:
 \EXERC\CAP8\PASCAL\EX10.PAS e \EXERC\CAP8\PASCAL\EX10.EXE

C/C++ 1ª SOLUÇÃO – FUNÇÃO ANTES DA main()
 \EXERC\CAP8\C++\EX10_A.CPP e \EXERC\CAP8\C++\EX10_A.EXE
2ª SOLUÇÃO – FUNÇÃO DEPOIS DA main()
 \EXERC\CAP8\C++\EX10_B.CPP e \EXERC\CAP8\C++\EX10_B.EXE

JAVA SOLUÇÃO:
 \EXERC\CAP8\JAVA\EX10.java e \EXERC\CAP8\JAVA\EX10.class

11. Foi realizada uma pesquisa sobre algumas características físicas de cinco habitantes de uma região. Foram coletados os seguintes dados de cada habitante: sexo, cor dos olhos (A — azuis; ou C — castanhos), cor dos cabelos (L — louros; P — pretos; ou C — castanhos) e idade. Faça um programa que apresente as sub-rotinas a seguir:

- Que leia esses dados, armazenando-os em vetores.
- Que determine e devolva ao programa principal a média de idade das pessoas com olhos castanhos e cabelos pretos.
- Que determine e devolva ao programa principal a maior idade entre os habitantes.
- Que determine e devolva ao programa principal a quantidade de indivíduos do sexo feminino com idade entre 18 e 35 anos (inclusive) e que tenham olhos azuis e cabelos louros.

ALGORITMO SOLUÇÃO:

```

ALGORITMO
DECLARE sexo[5], olhos[5], cabelos[5] LITERAL
        idade[5], x, i, q, m NUMÉRICO
leitura (sexo, olhos, cabelos, idade)
m ← media_idade(olhos, cabelos, idade)
ESCREVA m
i ← maior_idade(sexo, olhos, cabelos, idade)
ESCREVA i
q ← qt_individuos(sexo, olhos, cabelos, idade)
ESCREVA q
FIM_ALGORITMO.

SUB-ROTINA leitura(sexo[5], olhos[5], cabelos[5] LITERAL,
        idade[5] NUMÉRICO)
  DECLARE x NUMÉRICO
  PARA x ← 1 ATÉ 5 FAÇA
    INÍCIO
      REPITA
        LEIA sexo[x]
      ATÉ (sexo[x] = "F") OU (sexo[x] = "M")
      REPITA
        LEIA olhos[x]
      ATÉ (olhos[x] = "C") OU (olhos[x] = "A")

```

```

    REPITA
    LEIA cabelos[x]
    ATÉ (cabelos[x] = "C" OU cabelos[x] = "L" OU cabelos[x] = "P")
    LEIA idade[x]
    FIM
FIM_SUB_ROTINA leitura

SUB-ROTINA media_idade (olhos[5], cabelos[5] LITERAL,
                       idade[5] NUMÉRICO)
    DECLARE i, cont, soma, media NUMÉRICO
    soma ← 0
    cont ← 0
    PARA i ← 1 ATÉ 5 FAÇA
    INÍCIO
        SE (olhos[i] = "C") E (cabelos[i] = "P")
        ENTÃO INÍCIO
            soma ← soma + idade[i]
            cont ← cont + 1
        FIM
    FIM
    SE cont = 0
    ENTÃO media ← 0
    SENÃO media ← soma / cont
    RETORNE media
FIM_SUB-ROTINA media_idade

SUB-ROTINA maior_idade (idade[5] NUMÉRICO)
    DECLARE i, maior NUMÉRICO
    PARA i ← 1 ATÉ 5 FAÇA
    INÍCIO
    SE i = 1
        ENTÃO maior ← idade[i]
    SENÃO INÍCIO
        SE (idade[i] > maior)
        ENTÃO maior ← idade[i]
    FIM
    FIM
    RETORNE maior
FIM_SUB_ROTINA maior_idade

SUB-ROTINA qt_individuos (sexo[5], olhos[5], cabelos[5] LITERAL,
                        idade[5] NUMÉRICO)
    DECLARE i, qtd NUMÉRICO
    qtd ← 0
    PARA i ← 1 ATÉ 5 FAÇA
    INÍCIO
    SE sexo[i] = "F" E idade[i] >= 18 E idade[i] <= 35 E
        olhos[i] = "A" E cabelos[i] = "L"
    ENTÃO qtd ← qtd + 1
    FIM
    RETORNE qtd
FIM_SUB_ROTINA qt_individuos

```

PASCALSOLUÇÃO:

\EXERC\CAP8\PASCAL\EX11.PAS e \EXERC\CAP8\PASCAL\EX11.EXE

C/C++SOLUÇÃO:

\EXERC\CAP8\C++\EX11.CPP e \EXERC\CAP8\C++\EX11.EXE

JAVASOLUÇÃO:

\EXERC\CAP8\JAVA\EX11.java e \EXERC\CAP8\JAVA\EX11.class

- 12.** Elabore uma sub-rotina que retorne um vetor com os três primeiros números perfeitos. Sabe-se que um número é perfeito quando é igual à soma de seus divisores (exceto ele mesmo). Exemplo: os divisores de 6 são 1, 2 e 3, e $1 + 2 + 3 = 6$, logo 6 é perfeito.

ALGORITMO SOLUÇÃO:

```

ALGORITMO
DECLARE vet[3], i NUMÉRICO
perfeitos(vet)
PARA i ← 1 ATÉ 3 FAÇA
  INÍCIO
    ESCREVA vet[i]
  FIM
FIM_ALGORITMO.
SUB-ROTINA perfeitos(v[3] NUMÉRICO)
DECLARE a, r, num, soma, cont NUMÉRICO
cont ← 0
num ← 1
ENQUANTO (cont < 3) FAÇA
  INÍCIO
    soma ← 0
    PARA a ← 1 ATÉ num-1 FAÇA
      INÍCIO
        r ← RESTO(num / a)
        SE r = 0
          ENTÃO soma ← soma + a
      FIM
    SE soma = num
      ENTÃO INÍCIO
        v[cont + 1] ← num
        cont ← cont + 1
      FIM
    num ← num + 1
  FIM
FIM_SUB_ROTINA perfeitos

```

PASCALSOLUÇÃO:

\EXERC\CAP8\PASCAL\EX12.PAS e \EXERC\CAP8\PASCAL\EX12.EXE

C/C++SOLUÇÃO:

\EXERC\CAP8\C++\EX12.CPP e \EXERC\CAP8\C++\EX12.EXE

JAVASOLUÇÃO:

\EXERC\CAP8\JAVA\EX12.java e \EXERC\CAP8\JAVA\EX12.class

- 13.** Faça uma sub-rotina que receba um vetor A de dez elementos inteiros como parâmetro. Ao final dessa função, deverá ter sido gerado um vetor B contendo o fatorial de cada elemento de A. O vetor B deverá ser mostrado no programa principal.

ALGORITMO SOLUÇÃO:

```

ALGORITMO
DECLARE x, vet1[10], vet2[10] NUMÉRICO
PARA x ← 1 ATÉ 10 FAÇA

```

```

INÍCIO
  LEIA vet1[x]
FIM
fatoriais(vet1, vet2)
PARA x ← 1 ATÉ 10 FAÇA
  INÍCIO
    ESCREVA vet2[x]
  FIM
FIM_ALGORITMO.

SUB-ROTINA fatoriais(a[10], b[10] NUMÉRICO)
  DECLARE i, j , f NUMÉRICO
  PARA i ← 1 ATÉ 10 FAÇA
    INÍCIO
      SE (a[i] = 0) OU (a[i] = 1)
        ENTÃO b[i] ← 1
      SENÃO INÍCIO
        b[i] ← 1
        PARA j ← 1 ATÉ a[i] FAÇA
          INÍCIO
            b[i] ← b[i] * j
          FIM
        FIM
      FIM
    FIM
  FIM
FIM SUB_ROTINA fatoriais

```

PASCAL SOLUÇÃO:

```
\EXERC\CAP8\PASCAL\EX13.PAS e \EXERC\CAP8\PASCAL\EX13.EXE
```

C/C++ SOLUÇÃO:

```
\EXERC\CAP8\C++\EX13.CPP e \EXERC\CAP8\C++\EX13.EXE
```

JAVA SOLUÇÃO:

```
\EXERC\CAP8\JAVA\EX13.java e \EXERC\CAP8\JAVA\EX13.class
```

- 14.** Crie uma sub-rotina que receba como parâmetro dois vetores de dez elementos inteiros positivos e mostre o vetor união dos dois primeiros.

ALGORITMO SOLUÇÃO:

```

ALGORITMO
  DECLARE x, vet1[10], vet2[10], vet3[20], cont NUMÉRICO
  PARA x ← 1 ATÉ 10 FAÇA
    INÍCIO
      REPITA
        LEIA vet1[x]
        ATÉ vet1[x] >= 0
      FIM
  PARA x ← 1 ATÉ 10 FAÇA
    INÍCIO
      REPITA
        LEIA vet2[x]
        ATÉ vet2[x] >= 0
      FIM
  cont ← uniao(vet1,vet2,vet3)
  x ← 1

```

```

ENQUANTO x < cont FAÇA
INÍCIO
    ESCREVA vet3[x]
FIM
FIM_ALGORITMO.

SUB-ROTINA uniao(a[10], b[10], u[20] NUMÉRICO)
DECLARE i, j, k, cont NUMÉRICO
k = 1
    PARA i ← 1 ATÉ 10 FAÇA
        INÍCIO
            cont ← 1
            ENQUANTO cont < k E a[i] ≠ u[cont] FAÇA
                INÍCIO
                    cont ← cont + 1
                FIM
            SE cont = k
                ENTÃO INÍCIO
                    u[k] ← a[i]
                    k ← k + 1
                FIM
        FIM
    PARA i ← 1 ATÉ 10 FAÇA INÍCIO
        cont ← 1
        ENQUANTO cont < k E b[i] ≠ u[cont] FAÇA
            INÍCIO
                cont ← cont + 1
            FIM
        SE cont = k
            ENTÃO INÍCIO
                u[k] ← b[i]
                k ← k + 1
            FIM
        FIM
    RETORNE k
FIM_SUB_ROTINA uniao

```

PASCALSOLUÇÃO:

\EXERC\CAP8\PASCAL\EX14.PAS e \EXERC\CAP8\PASCAL\EX14.EXE

C/C++SOLUÇÃO:

\EXERC\CAP8\C++\EX14.CPP e \EXERC\CAP8\C++\EX14.EXE

JAVASOLUÇÃO:

\EXERC\CAP8\JAVA\EX14.java e \EXERC\CAP8\JAVA\EX14.class

- 15.** Faça uma sub-rotina que receba como parâmetro um vetor A com cinco números reais e retorne esses números ordenados de forma crescente.

ALGORITMO SOLUÇÃO:

```

ALGORITMO
DECLARE x, vet[5] NUMÉRICO
PARA x ← 1 ATÉ 5 FAÇA
    INÍCIO
        LEIA vet[x]
    FIM
ordena(vet)

```

```

PARA x ← 1 ATÉ 5 FAÇA
  INÍCIO
    ESCRIVA vet[x]
  FIM
FIM_ALGORITMO.

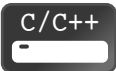
SUB-ROTINA ordena(v[5] NUMÉRICO)
  DECLARE i, j, aux NUMÉRICO
  PARA i ← 1 ATÉ 5 FAÇA
    INÍCIO
      PARA j ← 1 ATÉ 4 FAÇA
        INÍCIO
          SE (v[j] > v[j+1])
            ENTÃO INÍCIO
              aux ← v[j]
              v[j] ← v[j+1]
              v[j+1] ← aux
            FIM
          FIM
        FIM
      FIM
    FIM
  FIM
FIM SUB_ROTINA ordena

```



SOLUÇÃO:

\EXERC\CAP8\PASCAL\EX15.PAS e \EXERC\CAP8\PASCAL\EX15.EXE



SOLUÇÃO:

\EXERC\CAP8\C++\EX15.CPP e \EXERC\CAP8\C++\EX15.EXE



SOLUÇÃO:

\EXERC\CAP8\JAVA\EX15.java e \EXERC\CAP8\JAVA\EX15.class

- 16.** Crie uma sub-rotina que receba dois vetores A e B de dez elementos inteiros como parâmetro. A sub-rotina deverá determinar e mostrar um vetor C que contenha os elementos de A e B em ordem decrescente. O vetor C deverá ser mostrado no programa principal.

ALGORITMO SOLUÇÃO:

```




ALGORITMO
  DECLARE x, vet1[10], vet2[10], vet3[20] NUMÉRICO
  PARA x ← 1 ATÉ 10 FAÇA
    INÍCIO
      LEIA vet1[x]
    FIM
  PARA x ← 1 ATÉ 10 FAÇA
    INÍCIO
      LEIA vet2[x]
    FIM
  ordena_todos(vet1,vet2,vet3)
  PARA x ← 1 ATÉ 20 FAÇA
    INÍCIO
      ESCRIVA vet3[x]
    FIM
  FIM
FIM_ALGORITMO.

```

```

SUB-ROTINA ordena_todos(a[10], b[10], c[20] NUMÉRICO)
  DECLARE i, j, k, cont NUMÉRICO
  k ← 1
  PARA i ← 1 ATÉ 10 FAÇA
  INÍCIO
    cont ← 1
    ENQUANTO cont < k E a[i] < c[cont] FAÇA
    INÍCIO
      cont ← cont + 1
    FIM
  SE cont = k
  ENTÃO INÍCIO
    c[k] ← a[i]
    k ← k + 1
  FIM
  SENÃO INÍCIO
    PARA j ← k-1 ATÉ cont PASSO -1 FAÇA
    INÍCIO
      c[j+1] ← c[j]
    FIM
    c[cont] ← a[i]
    k ← k + 1
  FIM
  FIM
  PARA i ← 1 ATÉ 10 FAÇA
  INÍCIO
    cont ← 1
    ENQUANTO cont < k E b[i] < c[cont] FAÇA
    INÍCIO
      cont ← cont + 1
    FIM
  SE cont = k
  ENTÃO INÍCIO
    c[k] ← b[i]
    k ← k + 1
  FIM
  SENÃO INÍCIO
    PARA j ← k-1 ATÉ cont PASSO -1 FAÇA
    INÍCIO
      c[j+1] ← c[j]
    FIM
    c[cont] ← b[i]
    k ← k + 1
  FIM
  FIM
  FIM_SUB_ROTINA ordena_todos




```

PASCAL SOLUÇÃO: \EXERC\CAP8\PASCAL\EX16.PAS e \EXERC\CAP8\PASCAL\EX16.EXE**C/C++** SOLUÇÃO: \EXERC\CAP8\C++\EX16.CPP e \EXERC\CAP8\C++\EX16.EXE**JAVA** SOLUÇÃO: \EXERC\CAP8\JAVA\EX16.java e \EXERC\CAP8\JAVA\EX16.class**17.** Faça uma sub-rotina que receba como parâmetro uma matriz A(5,5) e retorne a soma de seus elementos.**ALGORITMO** SOLUÇÃO:

```

ALGORITMO
DECLARE x, y, s, matriz[5,5] NUMÉRICO
PARA x ← 1 ATÉ 5 FAÇA
  INÍCIO
    PARA y ← 1 ATÉ 5 FAÇA
      INÍCIO
        LEIA matriz[x,y]
      FIM
    FIM
  FIM
s ← soma_matriz(matriz)
ESCREVA s
FIM_ALGORITMO.
SUB-ROTINA soma_matriz(m[5,5] NUMÉRICO)
  DECLARE i, j, soma NUMÉRICO
  soma ← 0
  PARA i ← 1 ATÉ 5 FAÇA
    INÍCIO
      PARA j ← 1 ATÉ 5 FAÇA
        INÍCIO
          soma ← soma + m[i, j]
        FIM
      FIM
    FIM
  RETORNE soma
FIM SUB-ROTINA soma_matriz

```

PASCAL SOLUÇÃO: \EXERC\CAP8\PASCAL\EX17.PAS e \EXERC\CAP8\PASCAL\EX17.EXE**C/C++** SOLUÇÃO: \EXERC\CAP8\C++\EX17.CPP e \EXERC\CAP8\C++\EX17.EXE**JAVA** SOLUÇÃO: \EXERC\CAP8\JAVA\EX17.java e \EXERC\CAP8\JAVA\EX17.class**18.** Crie uma sub-rotina que receba como parâmetro uma matriz A(6,6) e retorne o menor elemento de sua diagonal secundária.**ALGORITMO** SOLUÇÃO:

```

ALGORITMO
DECLARE x, y, menor, matriz[6,6] NUMÉRICO
PARA x ← 1 ATÉ 6 FAÇA
  INÍCIO

```



```

        PARA y ← 1 ATÉ 6 FAÇA
            INÍCIO
                LEIA matriz[x,y]
            FIM
        FIM
menor ← menor_elemento(matriz)
ESCREVA menor
FIM_ALGORITMO.

SUB-ROTINA menor_elemento(m[6,6] NUMÉRICO)
    DECLARE i, j, me NUMÉRICO
    me ← m[1,6]
    j ← 5
    PARA i ← 2 ATÉ 6 FAÇA
        INÍCIO
            SE m[i,j] < me
                ENTÃO me ← m[i,j]
            j ← j - 1
        FIM
    RETORNE me
FIM_SUB_ROTINA menor_elemento

```

PASCALSOLUÇÃO:

```
\EXERC\CAP8\PASCAL\EX18.PAS e \EXERC\CAP8\PASCAL\EX18.EXE
```

C/C++SOLUÇÃO:

```
\EXERC\CAP8\C++\EX18.CPP e \EXERC\CAP8\C++\EX18.EXE
```

JAVASOLUÇÃO:

```
\EXERC\CAP8\JAVA\EX18.java e \EXERC\CAP8\JAVA\EX18.class
```

- 19.** Elabore uma sub-rotina que receba como parâmetro uma matriz A(6,6) e multiplique cada linha pelo elemento da diagonal principal da linha. A sub-rotina deverá retornar a matriz alterada para ser mostrada no programa principal.

ALGORITMO SOLUÇÃO:

```

ALGORITMO
DECLARE x, y, matriz[6,6] NUMÉRICO
PARA x ← 1 ATÉ 6 FAÇA
    INÍCIO
        PARA y ← 1 ATÉ 6 FAÇA
            INÍCIO
                LEIA matriz[x,y]
            FIM
        FIM
    multiplica_matriz(matriz)
PARA x ← 1 ATÉ 6 FAÇA
    INÍCIO
        PARA y ← 1 ATÉ 6 FAÇA
            INÍCIO
                ESCREVA matriz[x,y]
            FIM
        FIM
    FIM

```

```

        FIM
    FIM
FIM_ALGORITMO.

SUB-ROTINA multiplica_matriz(m[6,6] NUMÉRICO)
    DECLARE i, j, v NUMÉRICO
    PARA i ← 1 ATÉ 6 FAÇA
        INÍCIO
            v ← m[i,i]
            PARA j ← 1 ATÉ 6 FAÇA
                INÍCIO
                    m[i][j] ← m[i][j] * v
                FIM
            FIM
        FIM
    FIM
FIM_SUB_ROTINA multiplica_matriz

```

PASCALSOLUÇÃO:

\EXERC\CAP8\PASCAL\EX19.PAS e \EXERC\CAP8\PASCAL\EX19.EXE

C/C++SOLUÇÃO:

\EXERC\CAP8\C++\EX19.CPP e \EXERC\CAP8\C++\EX19.EXE

JAVASOLUÇÃO:

\EXERC\CAP8\JAVA\EX19.java e \EXERC\CAP8\JAVA\EX19.class

- 20.** Crie uma sub-rotina que receba como parâmetro uma matriz A(12,12) e retorne a média aritmética dos elementos abaixo da diagonal principal.

ALGORITMO SOLUÇÃO:

```

ALGORITMO
DECLARE x, y, matriz[12,12], m NUMÉRICO
PARA x ← 1 ATÉ 12 FAÇA
    INÍCIO
        PARA y ← 1 ATÉ 12 FAÇA
            INÍCIO
                LEIA matriz[x,y]
            FIM
        FIM
    FIM
m ← media_aritmetica(matriz)
ESCREVA m
FIM_ALGORITMO.

SUB-ROTINA media_aritmetica(m[12,12] NUMÉRICO)
    DECLARE i, j, cont, soma, media NUMÉRICO
    soma ← 0
    cont ← 0
    PARA i ← 2 ATÉ 12 FAÇA
        INÍCIO
            PARA j ← 12 ATÉ (14 - i) PASSO -1 FAÇA
                INÍCIO
                    soma ← soma + m[i,j]
                    cont ← cont + 1
                FIM
            FIM
        FIM
    FIM
    media ← soma / cont
    FIM
FIM_SUB_ROTINA media_aritmetica

```

```

        FIM
    FIM
    media ← soma/cont
    RETORNE media
FIM_SUB_ROTINA media_aritmetica

```

PASCALSOLUÇÃO:

```
\EXERC\CAP8\PASCAL\EX20.PAS e \EXERC\CAP8\PASCAL\EX20.EXE
```

C/C++SOLUÇÃO:

```
\EXERC\CAP8\C++\EX20.CPP e \EXERC\CAP8\C++\EX20.EXE
```

JAVASOLUÇÃO:

```
\EXERC\CAP8\JAVA\EX20.java e \EXERC\CAP8\JAVA\EX20.class
```

- 21.** Escreva um algoritmo que leia um número não determinado de pares de valores x,y (x obrigatoriamente deve ser menor que y), todos inteiros e positivos, um par de cada vez. Para cada par, chame uma sub-rotina que determine a soma dos números primos entre x e y (inclusive). O algoritmo deverá mostrar os valores de x e de y, seguidos pelo somatório calculado. A leitura dos pares terminará quando os valores digitados para x e y forem iguais.

ALGORITMO SOLUÇÃO:

```

ALGORITMO
DECLARE x, y, soma NUMÉRICO
LEIA x
LEIA y
ENQUANTO x ≠ y FAÇA
INÍCIO
    soma ← primos(x, y)
    ESCREVA soma
    LEIA x
    LEIA y
FIM
FIM_ALGORITMO.

SUB-ROTINA primos(x, y NUMÉRICO)
DECLARE i, j, r, cont, somatorio NUMÉRICO
somatorio ← 0
PARA i ← x ATÉ y FAÇA
    INÍCIO
        cont ← 0
        PARA j ← 1 ATÉ i FAÇA
            INÍCIO
                r ← RESTO(i / j)
                SE r = 0
                    ENTÃO cont ← cont + 1
            FIM
        SE cont <= 2
            somatorio ← somatorio + i
    FIM
RETORNE somatorio
FIM_SUB_ROTINA primos

```

PASCAL SOLUÇÃO:

\EXERC\CAP8\PASCAL\EX21.PAS e \EXERC\CAP8\PASCAL\EX21.EXE

C/C++ SOLUÇÃO:

\EXERC\CAP8\C++\EX21.CPP e \EXERC\CAP8\C++\EX21.EXE

JAVA SOLUÇÃO:

\EXERC\CAP8\JAVA\EX21.java e \EXERC\CAP8\JAVA\EX21.class

- 22.** Crie um programa que carregue um vetor com oito números inteiros, calcule e mostre dois vetores resultantes. O primeiro vetor resultante deverá conter os números positivos e o segundo, os números negativos. Cada vetor resultante terá **no máximo** oito posições, e nem todas serão obrigatoriamente utilizadas.

ALGORITMO SOLUÇÃO:

```

ALGORITMO
DECLARE vet1[8], vet2[8], vet3[8], i, cont1, cont2 NUMÉRICO
PARA i ← 1 ATÉ 8 FAÇA
  INÍCIO
    LEIA vet1[i]
  FIM
cont1 ← pares(vet1, vet2)
SE cont1 = 1
  ENTÃO ESCREVA "NENHUM PAR FOI DIGITADO"
SENÃO
  PARA i ← 1 ATÉ cont1 - 1 FAÇA
    INÍCIO
      ESCREVA vet2[i]
    FIM
cont2 ← impares(vet1, vet3)
SE cont2 = 1
  ENTÃO ESCREVA "NENHUM ÍMPAR FOI DIGITADO"
SENÃO
  PARA i ← 1 ATÉ cont2 - 1 FAÇA
    INÍCIO
      ESCREVA vet3[i]
    FIM
FIM_ALGORITMO.

SUB-ROTINA pares(v1[8], v2[8] NUMÉRICO)
DECLARE i, r, cont NUMÉRICO
cont ← 1
PARA i ← 1 ATÉ 8 FAÇA
  INÍCIO
    r ← RESTO(v1[i] / 2)
    SE r = 0
      ENTÃO INÍCIO
        v2[cont] ← v1[i]
        cont ← cont + 1
      FIM
  FIM
RETORNE cont
FIM_SUB_ROTINA pares

```

```

SUB-ROTINA impares(v1[8], v2[8] NUMÉRICO)
  DECLARE i, r, cont NUMÉRICO
  cont ← 1
  PARA i ← 1 ATÉ 8 FAÇA
    INÍCIO
      r ← RESTO(v1[i] / 2)
      SE r = 1
        ENTÃO INÍCIO
          v2[cont] ← v1[i]
          cont ← cont + 1
        FIM
    FIM
  FIM
  RETORNE cont
FIM_SUB_ROTINA impares

```

PASCAL SOLUÇÃO:
 \EXERC\CAP8\PASCAL\EX22.PAS e \EXERC\CAP8\PASCAL\EX22.EXE

C/C++ SOLUÇÃO:
 \EXERC\CAP8\C++\EX22.CPP e \EXERC\CAP8\C++\EX22.EXE

JAVA SOLUÇÃO:
 \EXERC\CAP8\JAVA\EX22.java e \EXERC\CAP8\JAVA\EX22.class

- 23.** Crie um programa que carregue uma matriz 3X4 com números reais. Utilize uma função para copiar todos os valores da matriz para um vetor de 12 posições. Esse vetor deverá ser mostrado no programa principal.

ALGORITMO SOLUÇÃO:

```

ALGORITMO
  DECLARE mat[3,4], vet[12], i, j NUMÉRICO
  PARA i ← 1 ATÉ 3 FAÇA
    INÍCIO
      PARA j ← 1 ATÉ 4 FAÇA
        INÍCIO
          LEIA mat[i,j]
        FIM
      FIM
    FIM
  gera_vetor(mat, vet)
  PARA i ← 1 ATÉ 12 FAÇA
    INÍCIO
      ESCREVA vet[i]
    FIM
  FIM_ALGORITMO.

SUB_ROTINA gera_vetor(m[3,4], v[12] NUMÉRICO)
  DECLARE i, j, k NUMÉRICO
  k ← 1
  PARA i ← 1 ATÉ 3 FAÇA
    INÍCIO
      PARA j ← 1 ATÉ 4 FAÇA
        INÍCIO
          v[k] ← m[i,j] k ← k+1
        FIM
      FIM
    FIM
  FIM
  FIM_SUB_ROTINA gera_vetor

```

PASCAL SOLUÇÃO:

\EXERC\CAP8\PASCAL\EX23.PAS e \EXERC\CAP8\PASCAL\EX23.EXE

C/C++ SOLUÇÃO:

\EXERC\CAP8\C++\EX23.CPP e \EXERC\CAP8\C++\EX23.EXE

JAVA SOLUÇÃO:

\EXERC\CAP8\JAVA\EX23.java e \EXERC\CAP8\JAVA\EX23.class

- 24.** Faça um programa contendo uma sub-rotina que receba dois valores numéricos e um símbolo. Esse símbolo representará a operação que se deseja efetuar com os números. Se o símbolo for +, deverá ser realizada uma adição, e, se for *, deverá ser efetuada uma multiplicação. O resultado deverá ser mostrado no programa principal.

ALGORITMO SOLUÇÃO:

```

ALGORITMO
DECLARE num1, num2, res NUMÉRICO
        op LITERAL
LEIA num1
LEIA num2
REPITA
    LEIA op
ATÉ op = '+' OU op = '*'
res ← calculo(num1, num2, op)
ESCREVA res
FIM_ALGORITMO.

SUB-ROTINA calculo(n1, n2 NUMÉRICO, simbolo LITERAL)
    DECLARE result NUMÉRICO
    SE simbolo = '+'
        ENTÃO result ← n1 + n2
    SENAÓ result ← n1 * n2
    RETORNE result
FIM_SUB_ROTINA calculo

```

PASCAL SOLUÇÃO:

\EXERC\CAP8\PASCAL\EX24.PAS e \EXERC\CAP8\PASCAL\EX24.EXE

C/C++ SOLUÇÃO:

\EXERC\CAP8\C++\EX24.CPP e \EXERC\CAP8\C++\EX24.EXE

JAVA SOLUÇÃO:

\EXERC\CAP8\JAVA\EX24.java e \EXERC\CAP8\JAVA\EX24.class

- 25.** Crie uma sub-rotina que receba como parâmetro um vetor A de 25 números inteiros e substitua todos os valores negativos de A por zero. O vetor resultante deverá ser mostrado no programa principal.

ALGORITMO SOLUÇÃO:

```

ALGORITMO
DECLARE vetor[25], i NUMÉRICO
PARA i ← 1 ATÉ 25 FAÇA
    INÍCIO

```

```

        LEIA vetor[i]
    FIM
substituicao(vetor)
PARA i ← 1 ATÉ 25 FAÇA
    INÍCIO
        ESCREVA vetor[i]
    FIM
FIM_ALGORITMO.

SUB-ROTINA substituicao(vet[25] NUMÉRICO)
DECLARE z NUMÉRICO
PARA z ← 1 ATÉ 25 FAÇA
    INÍCIO
        SE vet[z] < 0
            ENTÃO vet[z] ← 0
        FIM
FIM_SUB_ROTINA substituicao

```

PASCAL

SOLUÇÃO:

\EXERC\CAP8\PASCAL\EX25.PAS e \EXERC\CAP8\PASCAL\EX25.EXE

C/C++

SOLUÇÃO:

\EXERC\CAP8\C++\EX25.CPP e \EXERC\CAP8\C++\EX25.EXE

JAVA

SOLUÇÃO:

\EXERC\CAP8\JAVA\EX25.java e \EXERC\CAP8\JAVA\EX25.class

EXERCÍCIOS PROPOSTOS

1. Faça uma sub-rotina que receba um número inteiro e positivo N como parâmetro e retorne a soma dos números inteiros existentes entre o número 1 e N (inclusive).
2. Crie uma sub-rotina que receba três números inteiros como parâmetros, representando horas, minutos e segundos, e os converta em segundos. Exemplo: 2h, 40min e 10s correspondem a 9.610 segundos.
3. Elabore uma sub-rotina que receba dois números como parâmetros e retorne 0, se o primeiro número for divisível pelo segundo número. Caso contrário, deverá retornar o próximo divisor.
4. Faça uma sub-rotina que receba como parâmetro o raio de uma esfera, calcule e mostre no programa principal o seu volume: $v = 4/3 * R^3$.
5. Faça uma sub-rotina que receba um valor inteiro e verifique se ele é positivo ou negativo.
6. Crie uma sub-rotina que receba como parâmetro a altura (alt) e o sexo de uma pessoa e retorne seu peso ideal. Para homens, deverá calcular o peso ideal usando a fórmula: peso ideal = $72.7 * alt - 58$; para mulheres: peso ideal = $62.1 * alt - 44.7$.
7. Elabore uma sub-rotina que leia um número não determinado de valores positivos e retorne a média aritmética desses valores. Terminar a entrada de dados com o valor zero.
8. Faça uma sub-rotina que receba um valor inteiro e positivo, calcule e mostre seu fatorial.
9. Crie uma sub-rotina que receba como parâmetro um valor inteiro e positivo e retorne a soma dos divisores desse valor.
10. Elabore uma sub-rotina que receba como parâmetro um valor N (inteiro e maior ou igual a 1) e determine o valor da sequência S, descrita a seguir:

$$S = 1 + 1/2 + 1/3 \dots$$

**Observação**

A quantidade de parcelas que compõe S é igual a N.

- 11.** Faça uma sub-rotina que receba como parâmetro um valor inteiro e positivo N, indicando a quantidade de parcelas de uma soma S, calculada pela fórmula:

$$S = 2/4 + 5/5 + 10/6 + 17/7 + 26/8 + \dots + (n^2 + 1)/(n + 3)$$

- 12.** Crie uma sub-rotina que receba como parâmetro dois valores X e Z, calcule e retorne X^Z sem utilizar funções ou operadores de potência prontos.

- 13.** Foi realizada uma pesquisa entre 15 habitantes de uma região. Os dados coletados de cada habitante foram: idade, sexo, salário e número de filhos.

Faça uma sub-rotina que leia esses dados armazenando-os em vetores. Depois, crie sub-rotinas que recebam esses vetores como parâmetro e retornem a média de salário entre os habitantes, a menor e a maior idade do grupo e a quantidade de mulheres com três filhos que recebem até R\$ 500,00 (utilize uma sub-rotina para cada cálculo).

- 14.** Faça uma sub-rotina que receba um vetor X de 30 elementos inteiros como parâmetro e retorne dois vetores A e B. O vetor A deve conter os elementos de X que sejam maiores que zero e o vetor B, os elementos menores ou iguais a zero.

- 15.** Elabore uma sub-rotina que receba um vetor X de 15 números inteiros como parâmetro e retorne a quantidade de valores pares em X.

- 16.** Faça uma sub-rotina que receba um vetor X de 20 de números reais como parâmetro e retorne a soma dos elementos de X.

- 17.** Elabore uma sub-rotina que calcule o máximo divisor comum (MDC) de dois números recebidos como parâmetros.

- 18.** Crie uma sub-rotina que gere e mostre os três primeiros números primos acima de 100.

- 19.** Faça uma sub-rotina que receba como parâmetro dois vetores de dez números inteiros, determine e mostre o vetor intersecção entre eles.

- 20.** A prefeitura de uma cidade fez uma pesquisa entre seus habitantes, coletando dados sobre o salário e o número de filhos. Faça uma sub-rotina que leia esses dados para um número não determinado de pessoas e retorne a média de salário da população, a média do número de filhos, o maior salário e o percentual de pessoas com salário inferior a R\$ 380,00.

- 21.** Faça uma sub-rotina que receba uma matriz 10X10 e determine o maior elemento acima da diagonal principal. Esse valor deverá ser mostrado no programa principal.

- 22.** Criar um programa que:

- utilize uma sub-rotina para receber os elementos de uma matriz 10X5 de números reais;
- utilize uma sub-rotina para calcular a soma de todos os elementos localizados abaixo da sexta linha dessa matriz;

Os resultados deverão ser mostrados no programa principal.

- 23.** Crie um programa que receba três valores (obrigatoriamente maiores que zero), representando as medidas dos três lados de um triângulo.

Elabore sub-rotinas para:

- determinar se esses lados formam um triângulo (sabe-se que, para ser triângulo, a medida de um lado qualquer deve ser inferior ou igual à soma das medidas dos outros dois).
- determinar e mostrar o tipo de triângulo (equilátero, isósceles ou escaleno), caso as medidas formem um triângulo.

Todas as mensagens deverão ser mostradas no programa principal.

- 24.** Faça um programa que receba a temperatura média de cada mês do ano e armazene-as em um vetor. O programa deverá calcular e mostrar a maior e a menor temperatura do ano, junto com o mês em que elas ocorreram (o mês deverá ser mostrado por extenso: 1 = janeiro; 2 = fevereiro; ...).

**Observação**

Não se preocupe com empates. Cada cálculo deve ser realizado e mostrado em uma sub-rotina.

- 25.** Crie um programa que receba o número dos 10 alunos de uma sala, armazenando-os em um vetor, junto com as notas obtidas ao longo do semestre (foram realizadas quatro avaliações). Elabore sub-rotinas para:
- determinar e mostrar a média aritmética de todos os alunos;
 - indicar os números dos alunos que deverão fazer recuperação, ou seja, aqueles com média inferior a 6.

**Observação**

Todas as mensagens deverão ser mostradas no programa principal.