Ficha de Trabalho N.º 3 – Proposta de Resolução

Objectivos: Funções e *arrays*.

Soluções Propostas

1) Escreva as seguintes funções sobre o tipo *char*:

	Função	Devolve
a)	**int isdigit(char c)**	Verdade quando c é um dígito e Falso c.c.
b)	**int isalpha(char c)**	Verdade quando c é uma letra e Falso c.c.
c)	**int isalnum(char c)**	Verdade quando c é um carácter alfanumérico e Falso c.c.
d)	**char tolower(char c)**	Devolve c transformado na minúscula correspondente
e)	**char toupper(char c)**	Devolve c transformado na maiúscula correspondente

Nota: Obtém-se acesso a estas funções através da directiva
#include <ctype.h> // Funções sobre o tipo char (*ctype -> char type*)
Obs. Na resolução, não devem incluir a biblioteca, senão passaríamos a ter funções com o mesmo nome.

a)
```
int isdigit(char c)
        // função que devolve Verdade quando c é um dígito e Falso c.c.
{
     return (c>='0' && c=<'9');
}
```

b)
```
int isalpha(char c)
        // função que devolve Verdade quando c é uma letra e Falso c.c.
{
     return (c>='a' && c<= 'z' || c>='A' && c<='Z');
}
```

c)
```
int isalnum(char c)
  // função que devolve Verdade quando c é um carácter alfanumérico e Falso c.c.
{
     return isalpha(c) || isdigit(c);
}
```

d)
```
char tolower(char c)
        // função que devolve c transformado na minúscula correspondente
{
     if (c>='A' && c<='Z')
         return c+'a'-'A';
     else
         return c;
}
```
NOTA: representação em ASCII:
'A'-> 65
'a' -> 97
'A'+32 = 'a'
'a'-32 = 'A'

e)

```
char toupper(char c)
        // função que devolve c transformado na maiúscula correspondente
{
     if (c>='a' && c<='z')
         return c + 'A'-'a';
     else
         return c;
}
```

2) Escreva as funções a seguir indicadas de modo que devolvam os resultados descritos.

	Função	Devolve
a)	**int resto (int a, int b)**	O resto da divisão de a por b
b)	**int impar (int x)**	Verdade se x for impar e Falso c.c.
c)	**int perfeito (int n)**	Verdade se n for "perfeito" (igual à soma dos divisores de n, inferiores a n) e Falso c.c.
d)	**int primo (int n)**	Verdade se n for "primo" (apenas divisível por 1 e por n) e Falso c.c.

a)

```
int resto(int a, int b)
// Devolve o resto da divisão de a por b

{
     return a % b;
}
```

b)

```
int impar(int x)
// Devolve Verdade se x for impar e Falso c.c.

{
     return resto (x,2)!=0
}
```

c)

```
int perfeito (int n)
// Devolve Verdade se n for "perfeito" (igual à soma dos divisores de n, //inferiores a n)
e Falso c.c.
{
    int div, soma, meio;

     soma=0;
     meio=n/2;
     for (div=1; div<=meio; div++)
          if (resto(n,div)==0)
               soma +=div;
     return n==soma;
}
```

d)

VERSÃO 1

```
int primo (int n)
{
     int div, meio, sai;
     div=1;
     meio=n/2;
     do
     {
          div++; // equivalente a div=div+1;
          sai = (n % div) ==0;
     } while (sai==0 && div<meio);
     return sai==0; // devolve um valor LÓGICO
}
```

VERSÃO 2

```
int primo (int n)
{
      int div, meio, sai;
      meio=n/2;
      for (div=2; div<=meio; div++)
            if (n%div==0)
                  return 0;
      return 1;
}
```

Main para chamar as funções dos exercícios 1 e 2.

```
// Exercícios 1 e 2
// Main() para chamar as funções dos exercícios 1 e 2.
#include <stdio.h>
#include<locale.h>
#include<stdlib.h>
//int somaDig(int n);
// Protótipos das funções que devem estar num ficheiro de nome, p. ex., funcs.c, incluído no Projecto.
// Caso assim não seja, colocar aqui o código de todas estas funções.
// Ex. substituir o protótipo int isDigit(char c); por:
// ExerciÍio 1-a) função que devolve Verdade quando c é um dígito e Falso c.c.
/*int isDigit(char c)
{
    return (c >= '0' && c <= '9');
}*/
// Infra está o protótipo da função apresentada anteriormente (a utilizar se a função estiver abaixo
da invocação)
// ou em um outro ficheiro;
int isDigit(char c);
int isAlpha(char c);
int isAlphaNum(char c);
char toLower(char c);
char toUpper(char c);
int resto(int a, int b);
int impar(int x);
int perfeito (int n);
int primo (int n);

int main(void)
{
    setlocale(LC_ALL,"Portuguese");
//Exercício 1
    system("clear");
    char car;
    printf("*** Teste do exercício 1.............\n\n");
 // a)
    printf("---> alínea a) verificar se um caracter é um dígito\n");
    printf("Insira o caracter a testar: ");
    scanf("%c", &car);
    if (isDigit(car)) //verdade se for diferente de 0
        printf("%c é dígito!\n", car);
    else
        printf("%c não é dígito!\n", car);
// b)
    printf("\n---> alínea b) verificar se um caracter é uma letra\n");
    printf("Insira o caracter a testar: ");
    scanf(" %c", &car);
    if(isAlphaNum(car))
        printf("O caracter %c é uma letra!!\n", car);
    else
        printf("O caracter %c não é uma letra!!\n", car);
// c)
    printf("\n---> alínea c) verificar se um caracter é uma letra ou um número\n");
    printf("Insira o caracter a testar: ");
    scanf(" %c", &car);
    if(isAlphaNum(car))
        printf("O caracter %c é alfanumérico!!\n", car);
    else
        printf("O caracter %c não é alfanumérico!!\n", car);

 // d)
    printf("\n---> alínea d) converter uma letra maiúscula para minúscula (se for o caso)\n");
    printf("Insira o caracter a converter: ");
    scanf(" %c", &car);
    if(isAlpha(car))
```

```
        printf("A letra %c, convertida para minúscula: %c\n", car, toLower(car));
    else
        printf("O caracter %c não é uma letra!!\n", car);
    // e)
         printf("\n---> alínea e) converter uma letra minúscula para maiúscula (se for o caso)\n");
         printf("Insira o caracter a converter: ");
     scanf(" %c", &car);
     if(isAlpha(car))
         printf("A letra %c, convertida para maiúscula: %c\n", car, toUpper(car));
     else
         printf("O caracter %c não é uma letra!!\n", car);

//Exercício 2
    printf("\nDigite um caracter + Enter para continuar... ");
    scanf(" %c", &car);
    system("clear");
    printf("*** Teste do exercício 2..............\n\n");
    int a, b;
// a)
    printf("---> alínea a) devolver o resto da divisão inteira de a por b\n");
    printf("Insira o valor de a e b, separados por espaço: ");
    scanf("%d %d", &a, &b);
    printf("O resto da divisão inteira de %d por %d é: %d\n\n", a, b, resto(a, b));
// b)
    printf("---> alínea b) verificar se um número inteiro positivo é impar\n");
    printf("Insira o valor do número: ");
    do
    {
        printf("Insira o valor do numero inteiro positivo: ");
        scanf(" %d", &a);
    } while (a <= 0);
    if(impar(a)) //devolve diferente de 0: é verdade
        printf("O número %d é ímpar\n\n", a);
    else //devolve 0: é falso
        printf("O número %d não é ímpar\n\n", a);
// c)
    printf("---> alínea c) verificar se um número inteiro positivo é perfeito\n");
    do
    {
        printf("Insira o valor do numero inteiro positivo: ");
        scanf(" %d", &a);
    } while (a <= 0);
    if (perfeito(a)) //devolve diferente de 0: é verdade
         printf("\nO número %d é um número perfeito!!\n", a);
     else//devolve 0: é falso
         printf("\nO número %d não é um número perfeito!!\n", a);
// d)
    printf("---> alínea d) verificar se um número inteiro positivo é primo\n");
    do
    {
        printf("Insira o valor do numero inteiro positivo: ");
        scanf(" %d", &a);
    } while (a <= 0);
     if (primo(a))
         printf("\n%d é um número primo!!\n", a);
     else
         printf("\n%d não é um número primo!!\n\n", a);

    // Exercício adicional: aceitar um valor e calcular a soma dos dígitos
    /*    int num,total;
         num=leitura();
         total=somaDig(num);
         printf("A soma dos dígitos é %d",total);
    */

    return 0;
}
```

3) No século I d.C. os números naturais dividiam-se em três categorias:

REDUZIDOS: os superiores à soma dos seus divisores;
ABUNDANTES: os inferiores à soma dos seus divisores
PERFEITOS: os que são iguais à soma dos seus divisores.

NOTA: nesta definição exclui-se o próprio número, do conjunto dos seus divisores.

Escreva uma função que liste os inteiros entre a e b, a<b, classificando-os de acordo com esse critério, e que escreva também o total de cada uma das categorias.

Versão 1 (responde ao solicitado na questão 3)

```
// Exercício 3
// Programa para listar os inteiros entre a e b, a<b, classificando-os de acordo com os critérios:
// REDUZIDOS: os superiores à soma dos seus divisores;
// ABUNDANTES: os inferiores à soma dos seus divisores;
// PERFEITOS: os que são iguais à soma dos seus divisores;
// O programa deve tambem escrever o total de números de cada uma das categorias.
#include <stdio.h>
#include<locale.h>
#include<stdlib.h>
# define _CRT_SECURE_NO_WARNINGS
// Protótipo da função c_totais(...) que deve estar, tal como as funções dos exercícios 1 e 2,
// no ficheiro criado para o efeito, incluído no Projecto.
// Caso assim não seja, colocar aqui o código da função.
void c_totais (int na, int nb);
int main(void)
{
    setlocale(LC_ALL,"Portuguese");
    system("clear");
    int lim_inf, lim_sup;
    do
    {
        printf("\nDigite dois numeros inteiros positivos, entre [1...500],
                                                    separados por espaços, \n");
        printf("sendo estes o limite inf. e sup. dos números que pretende classificar: ");
        scanf("%d %d",&lim_inf, &lim_sup);
    } while (lim_inf<1 || lim_inf>lim_sup || lim_sup>500);
    c_totais (lim_inf, lim_sup);
    printf("\n\n");
}
```

Função c_Totais, a incluir no ficheiro funcs.c:

```
// Exercício 3)
// Função que liste os inteiros entre a e b, a>b, classificando-os de acordo com o critério:
// REDUZIDOS: os superiores à soma dos seus divisores
// ABUNDANTES: os inferiores à soma dos seus divisores
// PERFEITOS: os que são iguais à soma dos seus divisores
// No final deve escrever também o total de cada uma das categorias
//
void c_totais (int na, int nb)
{
    int n, div, soma, meio, nperfeitos, nreduzidos, nabundantes;
    n=na;
    nperfeitos=0;
    nreduzidos=0;
    nabundantes=0;
    do
    {
        soma=0;
        meio=n/2;
        for (div=1; div<=meio; div++)
            if (resto(n,div)==0)
                soma +=div;
        if (n==soma)S        {    printf("\n\t%d\t PERFEITO",n);
            nperfeitos ++;
        }
        else if (n>soma)
            {   printf("\n\t%d\t REDUZIDO",n);
                nreduzidos ++;
            }
```

```
            else
                {   printf("\n\t%d\t ABUNDANTE",n);
                    nabundantes ++;
                }
        n++;
    } while (n <= nb);
    printf("\n\n\n\tTotal de números PERFEITOS: %d",nperfeitos);
    printf("\n\tTotal de números REDUZIDOS: %d",nreduzidos);
    printf("\n\tTotal de números ABUNDANTES: %d",nabundantes);
}
```

Versão 2 (responde ao solicitado na questão 3, excepto quanto aos totais por cada categoria)

```
// Neste caso, as funções são colocadas antes do main()
#include <stdio.h>
int resto (int a, int b)
{
      return a % b;
}
void classifica (int na, int nb)
{
      int n, div, soma, meio;
      n=na;
      do
      {     soma=0;
            meio=n/2;
            for (div=1; div<=meio; div++)
                  if (resto(n,div)==0)
                        soma +=div;
            if (n==soma)
                  printf("\n\t%d\t PERFEITO",n);
            else if (n>soma)
                        printf("\n\t%d\t REDUZIDO",n);
                  else printf("\n\t%d\t ABUNDANTE",n);
            n++; } while (n <= nb);
}
void main()
{
      int lim_inf, lim_sup;
      do
      {
              printf("\nDigite dois numeros inteiros positivos (1..500): ");
              scanf("%d %d",&lim_inf, &lim_sup);
      } while (lim_inf<1 || lim_inf>lim_sup || lim_sup>500);
      classifica(lim_inf,lim_sup);
      printf("\n");
}
```

Versão 1, mas com as funções colocadas antes do main()

```
#include <stdio.h>
...
int resto (int a, int b); // função do exercício 2

int impar (int a); // função do exercício 2

void c_totais (int na, int nb)
{
    int n, div, soma, meio, nperfeitos, nreduzidos, nabundantes;
    n=na;
    nperfeitos=0;
    nreduzidos=0;
    nabundantes=0;
    do
    {
        soma=0;
        meio=n/2;
        for (div=1; div<=meio; div++)
            if (resto(n,div)==0)
                soma +=div;
        if (n==soma)
        {    printf("\n\t%d\t PERFEITO",n);
            nperfeitos ++;
        }
        else if (n>soma)
            {   printf("\n\t%d\t REDUZIDO",n);
                nreduzidos ++;
            }
            else
                {   printf("\n\t%d\t ABUNDANTE",n);
                    nabundantes ++;
                }
        n++;
    } while (n <= nb);
    printf("\n\n\n\tTotal de números PERFEITOS: %d",nperfeitos);
    printf("\n\tTotal de números REDUZIDOS: %d",nreduzidos);
    printf("\n\tTotal de números ABUNDANTES: %d",nabundantes);
}

void main()
{
     int lim_inf, lim_sup;
     do
     {
          printf("\nDigite dois numeros inteiros positivos (1..500): ");
          scanf("%d %d",&lim_inf, &lim_sup);
     } while (lim_inf<1 || lim_inf>lim_sup || lim_sup>500);
     c_totais (lim_inf, lim_sup);
     printf("\n");
}
```

4) Elabore funções que determinem:
   a. O cubo de um número inteiro **n**. O número **n** deve ser pedido ao utilizador através de uma função (denominada **leitura**) e o seu **cubo** deve ser calculado através de outra função (de nome **cubo**).
   b. Copiar a função cubo criada na alínea a., e alterá-la, criando a função **exponenciacao**, de forma a torná-la mais genérica: calcular **x^exp**. Os números **x** e **exp** devem ser solicitados ao utilizador através da função **leitura**. De igual modo, **x^exp** deve ser calculado através da função **exponenciacao**.

   Obs. No final, teste e corrija as funções, criando um main() para o efeito.

Proposta de resolução

a)

```
#include <stdio.h>

/* Função para leitura de um valor inteiro */
int leitura(int nValor)
{
    int n;
    printf("Qual o %d.º valor? ", nValor);
    scanf("%d",&n);
    return n;
}

/* Função que devolve o cubo de um inteiro */
int cubo(int x)
{
    return x*x*x;
}
```

b)

```
/* Função mais genérica que devolve o valor de um inteiro (x) levantado a outro
inteiro (exp)*/

// Função mais genérica que devolve o valor de um inteiro (x) elevado a outro inteiro (exp)
int exponenciacao(int x, int exp)
{
    int i=0;
    int result=x;
    for(i=1; i< exp; i++)
      result*=x;
    return result;
}
```

Main()

```
// Exercicio 4 - Main para testar as alíneas a 2 b do exercício 4
// a. O cubo de um número inteiro n. O número n deve ser pedido ao utilizador através
//    de uma função (denominada leitura)
//    e o seu cubo deve ser calculado através de outra função (de nome cubo).
// b. Copiar a função cubo criada na alínea a., e alterá-la, criando a função exponenciacao,
//    de forma a torná-la mais genérica:
//    calcular x^exp. Os números x e exp devem ser solicitados ao utilizador através
//    da função leitura.
//    De igual modo, x^exp deve ser calculado através da função exponenciacao.
// Obs. No final, teste e corrija as funções, criando um main() para o efeito.

#include <stdio.h>
#include<stdlib.h>
#include<locale.h>
# define _CRT_SECURE_NO_WARNINGS
int leitura(int nValor);
int cubo(int x);
int exponenciacao(int x, int exp);
int main(void)
{
    setlocale(LC_ALL,"Portuguese");
    system("clear");
    printf("*** Teste do exercício 4..............\n\n");
// a)
    printf("---> alínea a) calcular o cubo de um número inteiro n\n");
```

```
    int num, c;
    num=leitura(1);
    c=cubo(num);            // ou  c=cubo(leitura(1));
    printf("\tO cubo de %d é %d!\n",num, c);
    printf("\n\n");

// b)
    printf("---> alínea b) calcular x^exp, onde x e exp são inteiros n\n");
    int x, exp, res;
    x=leitura(1);
    exp=leitura(2);
    res=exponenciacao(x, exp);
    printf("\tO resultado de %d^%d = %d\n",x, exp, res);
    printf("\n\n");
    return 0;
      }
```

5) Escreva uma função que determine o maior de dois números dados. Teste a função num pequeno programa.

```
#include <stdio.h>
int maximo(int x, int y)
{
      if(x>y)
                  return x;
      else
                  return y;
}

void main()
{
       int a, b;
       printf("Insira dois números inteiros, separados por espaço: ");
       scanf("%d %d",&a, &b);
       printf("\nMax{%d, %d}=%d\n", a,b,maximo(a,b));
}
```

6) Elabore um programa que:
   a) Leia as *n* componentes de um vector;
   b) Escreva as n componentes de um vector;
   c) Determine a posição em que se encontra a maior componente.

   Cada uma destas tarefas deve ser realizada por uma função.

a)

```
#include <stdio.h>


void le_vector(int x[], int tam)
// Funçao para leitura de um vector com tam inteiros
{
     int i;
     for (i = 0; i<tam; i++)
     {
          printf("Qual o %d° elemento? ",i+1);
          scanf("%d",&x[i]);
     }
}
```

b)

```
void escreve_vector(int x[], int tam)
// Função que mostra os elementos de um vector com tam inteiros
{
     int i;
     for (i = 0; i<tam;i++)
          printf("x[%d]=%d   ",i+1, x[i]);
     printf("\n");
}
```

c)

```
int pos_max (int x[], int tam)
//Função para determinar qual a MAIOR componente dum vector
{
     int p,i;
     p=0;
     for (i = 1; i<tam; i++)
     if (x[i]>x[p])
          p=i;
     return p;
}

void main()
{
 int v[20], n, p_max;
 do
 {
     printf("\nIndique o numero de componentes do vector, sem exceder 20: ");
     scanf("%d",&n);
 } while (n<1 || n>20);
 le_vector(v, n);
 printf("\nA maior componente do vector");
 escreve_vector(v, n);
 p_max = pos_max (v,n) ;
 printf("\né v[%d]=%d", p_max+1, v[p_max]);
}
```

7) Escreva uma função que determine o produto interno de dois vectores x e y de n componentes inteiras.

```
// Função do Exercício 7
// Função que determina o produto interno de dois vectores (x e y)
int prod_int(int x[], int y[], int n)
{
  int soma, i;
  soma=0;
  for (i=0;i<n;i++)
      soma+=x[i]*y[i];
  return(soma);
}
```

8) Relativamente ao vector x do exercício anterior, escreva funções que permitam realizar as seguintes operações:
   a) Trocar a componente da posição p com a da posição q;
   b) Efectuar a permutação circular do vetor dado.

```
void troca_2componentes(int x[], int p1, int p2)
// Função para troca das componentes nas posicoes p1 e p2
{
      int aux;
      aux=x[p1];
      x[p1]=x[p2];
      x[p2]=aux;
}

void permutacao_circular(int x[], int tam)
//Permuta circularmente as componentes de um vector
{
      int i, aux;
      aux= x[0];
      for (i=0; i<tam-1 ; i++)
         x[i]= x[i + 1];
      x[tam-1]= aux;
}
```

Main para testar as funções dos exercícios 7 e 8:

```
// Exercício 7 e 8 – Main para testar as funções respectivas
// 7) Escreva uma função que determine o produto interno de dois vectores x e y
// com n componentes inteiras.
// 8) Relativamente ao vector x do exercício anterior, escreva funções que
// permitam realizar as seguintes operações:
//    a)Trocar a componente da posição p com a da posição q;
//    b) Efectuar a permutação circular do vetor dado.
#include <stdio.h>
#include<stdlib.h>
#include<locale.h>
#define _CRT_SECURE_NO_WARNINGS
#define MAX 20
void escreveVector(int x[], int tam);
void leVector(int x[], int tam);
int prodInterno(int x[], int y[], int n);
void trocaPosicao(int x[], int p1, int p2);
void permutacaoCircular(int x[], int tam);
int main(void)
{
    setlocale(LC_ALL,"Portuguese");
    system("clear");
    printf("*** Teste do exercício 7 (cálculo do prod. interno de 2 vectores.............\n\n");
    int tam, p, q, pI;
    char c;
    int vect1[MAX],vect2[MAX];
    do
    {
        printf("Qual o tamanho dos 2 vectores (máximo admissível: %d)? ", MAX);
        scanf("%d", &tam);
    } while ((tam > MAX) || (tam <= 0));
```

```
    //tam é o número de elementos utilizados no vector
    printf("Insira os valores para os elementos do 1.º vector....\n");
    leVector(vect1, tam);
    printf("Insira os valores para os elementos do 2.º vector....\n");
    leVector(vect2, tam);
    printf("\n\nO produto interno do vector:\n");
    escreveVector(vect1, tam);
    printf("e do vector:\n");
    escreveVector(vect2, tam);
    pI=prodInterno(vect1, vect2,tam);
    printf("--> é: %d\n\n",pI);

    printf("\n Digite <Enter> para continuar...");
    c=getchar();
    c=getchar();

    system("clear");
    printf("\n\n*** Teste do exercício 8, usando o vector vect1..............\n\n");
    printf("---> alínea a) tocar a componente p com a da posição q \n");
    printf("\nIndique quais as posições entre 0 e %d do vector vect1 a trocar,
                            separadas por espaços? ", tam-1);
    do{
        scanf("%d %d",&p,&q);
    }while ((p<0 && p>tam) || (q<0 && q>tam));

    printf("\nO Vect1 original era:\n");
    escreveVector(vect1, tam);
    trocaPosicao(vect1,p,q);
    printf("\nO vector Vect 1 depois da troca fica: \n");
    escreveVector(vect1, tam);

    printf("\n\n---> alínea b) efectuar a permutação circular do vector vect1\n");
    printf("O Vect1 original era:\n");
    escreveVector(vect1, tam);
    permutacaoCircular(vect1,tam);
    printf("O vetor vect1 depois da realizada a permutação circular é: \n");
    escreveVector(vect1, tam);
    printf("\n\n");
    return 0;
}
```

9) Considere uma matriz quadrada com nxn elementos inteiros.
Elabore um programa que lhe permita:

- Ler os *nxn* elementos da matriz;
- Mostrar no monitor os *nxn* elementos da matriz;
- Determinar o valor mínimo de uma matriz lida;
- Verificar se a matriz é ou não simétrica;
- Determinar a transposta da matriz;
- Calcular a soma de duas matrizes dadas.

Cada tarefa deve ser realizada por uma função.

```
#include <stdio.h>
#define MAX 10
void le_mat(int m[MAX][MAX], int tam)
// Funçao para leitura de uma matriz
{
      int i,j;
      for (i = 0;i < tam;i++)
            for (j = 0;j < tam;j++)
            {
                  printf("\n[%d,%d]= ",i+1,j+1);
                  scanf("%d",&m[i][j]);
            }
}

void escreve_mat(int m[MAX][MAX], int tam)
// Funçao para escrita de uma matriz
{
      int i,j;
      for (i = 0;i < tam;i++)
      {
            for (j = 0;j < tam;j++)
                  printf("%5c[%d,%d]=%d",' ',i+1,j+1,m[i][j]);
            printf("\n");
      }
}

int menor_elemento(int m[MAX][MAX], int nl, int nc)
// Determina a menor componente da matriz m com tam por tam elementos

{
      int i, j;
      int menor=m[0][0];
      for (i = 0;i < nl;i++)
      {
            for (j = 0;j < nc; j++)
                  if (m[i][j] < menor)
                        menor = m[i][j];
      }
      return menor;
}

int MatrizSimetrica(int m[MAX][MAX], int nl, int nc)
{
      int i, j, teste;
      teste= (nl==nc);
      if (teste)
            for (i = 1;i < nl;i++)
            {
                  for (j = 0;j < i; j++)
                        if (m[i][j]!=m[j][i])
                              return false;
            }
      return teste;
}
```

```
void transpor (int m[MAX][MAX], int tam)
// Transpõe a matriz m com tam por tam elementos
{
      int i, j, tmp;
      for (i=1; i<tam; i++)
            for (j=0; j<i; j++)
                  {tmp=m[i][j];
                   m[i][j]=m[j][i];
                   m[j][i]=tmp;
                  }
}


void soma_matrizes(int ma[MAX][MAX],int mb[MAX][MAX],int mc[MAX][MAX],int nl,int nc) //
determina a matriz mc, soma de duas matrizes dadas, ma e mb
{
      int i, j;
      for (i=0; i<nl; i++)
            for (j=0; j<nc; j++)
                  mc[i][j]=ma[i][j]+ mb[i][j];
}

void main()
{
      int mat1[MAX][MAX],mat2[MAX][MAX],mat3[MAX][MAX];
      int dim=5;
      le_mat(mat1, dim);
      printf("\n MATRIZ LIDA\n");
      escreve_mat(mat1, dim);
      printf("\nValor mínimo= %d\n", menor_elemento(mat1,dim,dim));
      if (MatrizSimetrica(mat1, dim, dim))
            printf("\n Matriz simetrica\n");
      else printf("\n Matriz NAO simetrica\n");
      printf("\n SEGUNDA MATRIZ:\n");
      le_mat(mat2, dim);
      printf("\n SEGUNDA MATRIZ LIDA\n");
      escreve_mat(mat2, dim);
      soma_matrizes (mat1,mat2,mat3,dim,dim);
      printf("\n  MATRIZ SOMA\n");
      escreve_mat(mat3, dim);
      transpor (mat3, dim);
      printf("\n  MATRIZ SOMA TRANSPOSTA\n");
      escreve_mat(mat3, dim);
}
```