

MC102 – Algoritmos e Programação de Computadores

Instituto de Computação

UNICAMP

Primeiro Semestre de 2016

Roteiro

- 1 Funções
- 2 A função `main`
- 3 O tipo `void`
- 4 Protótipos de funções
- 5 Exemplo de uso de funções
- 6 Escopo de variáveis
- 7 Vetores e funções
- 8 Matrizes e funções
- 9 Exercícios

Funções

- Um aspecto importante na resolução de um problema complexo é conseguir dividi-lo em subproblemas menores.
- Sendo assim, ao criarmos um programa para resolver um determinado problema, uma tarefa importante é dividir o código em partes menores, fáceis de serem compreendidas e mantidas.
- Funções são estruturas da linguagem que agrupam um conjunto de comandos, que são executados quando a função é chamada.
- Exemplo de uma chamada de função:

```
scanf ("%d", &x);
```

- Funções podem retornar um valor ao final de sua execução.
- Exemplo de uma função que retorna um valor:

```
x = sqrt(4);
```

Por que utilizar funções?

- Evitar que os blocos do programa fiquem grandes demais e, por consequência, difíceis de ler e entender.
- Separar o programa em partes que possam ser logicamente compreendidas de forma isolada.
- Permitir o reaproveitamento de códigos, implementados por você ou por outros programadores.
- Evitar que um trecho de código seja repetido várias vezes dentro de um mesmo programa, evitando inconsistências e facilitando alterações.

Definindo uma função

```
tipo nome(tipo parâmetro_1, ..., tipo parâmetro_N) {  
    comandos;  
    ...  
    comandos;  
  
    return valor_de_retorno;  
}
```

- Toda função deve ter um tipo que determina seu valor de retorno.
- Os parâmetros são variáveis que serão utilizadas pela função. Tais variáveis são inicializadas com valores na chamada da função.
- Para os nomes das funções e dos seus parâmetros valem as mesmas regras para os nomes de variáveis.
- Não é possível definir duas ou mais funções com o mesmo nome no mesmo programa.

Exemplo de função

A função abaixo soma dois valores, passados como parâmetros:

```
int soma(int a, int b) {  
    int c;  
    c = a + b;  
    return c;  
}
```

- O valor de retorno deve ser do mesmo tipo definido para a função.
- Quando o comando `return` é executado, a função termina sua execução e retorna o valor indicado para quem fez a chamada da função.
- Importante: a execução de um programa sempre começa pela função `main`.

Exemplo de função

```
#include <stdio.h>

int soma(int a, int b) {
    int c;
    c = a + b;
    return c;
}

int main() {
    int res, x1 = 4, x2 = -10;

    res = soma(5, 6);
    printf("Primeira soma: %d\n", res);

    printf("Segunda soma: %d\n", soma(x1, x2));

    return 0;
}
```

Definindo uma função

- Uma função pode não ter parâmetros, neste caso, basta não informá-los:

```
tipo nome() {  
    comandos;  
    ...  
    comandos;  
  
    return valor_de_retorno;  
}
```

- Em C, funções só podem ser definidas fora de outras funções.
- A expressão contida no comando `return` é chamada de valor de retorno. Após a execução do comando `return`, nenhum outro comando da função será executado.

Exemplo de função

```
#include <stdio.h>

int leNumero() {
    int n;
    printf("Digite um numero: ");
    scanf("%d", &n);
    return n;
}

int soma(int a, int b) {
    return (a + b);
}

int main() {
    int x1, x2;

    x1 = leNumero();
    x2 = leNumero();

    printf("Valor da soma: %d\n", soma(x1, x2));

    return 0;
}
```

Exemplo de função

```
#include <stdio.h>

int leNumero() {
    int n;
    printf("Digite um numero: ");
    scanf("%d", &n);
    return n;
    printf("bla bla bla bla!"); /* Nao imprime esta mensagem */
}

int main() {
    int x1, x2;

    x1 = leNumero();
    x2 = leNumero();

    printf("Valor da soma: %d\n", x1 + x2);

    return 0;
}
```

Chamada de uma função

- Para cada um dos parâmetros da função, devemos fornecer um valor, de mesmo tipo, na chamada da função.
- Ao chamar uma função passando variáveis como parâmetros, estamos usando apenas os seus valores que serão copiados para as variáveis parâmetros da função.
- Os valores das variáveis usadas na chamada da função não são afetados por alterações dentro da função.

Chamada de uma função

```
#include <stdio.h>

int somaEsquisita(int x, int y) {
    x = x + 1;
    y = y + 1;
    return (x + y);
}

int main() {
    int a = 10, b = 5;

    printf("Soma de a e b: %d\n", a + b);
    printf("Soma de x e y: %d\n", somaEsquisita(a, b));
    printf("a: %d\n", a);
    printf("b: %d\n", b);

    return 0;
}
```

Chamada de uma função

```
#include <stdio.h>

int somaEsquisita(int x, int y) {
    x = x + 1;
    y = y + 1;
    return (x + y);
}

int main() {
    int a = 10, b = 5;

    printf("Soma de a e b: %d\n", a + b); /* 15 */
    printf("Soma de x e y: %d\n", somaEsquisita(a, b)); /* 17 */
    printf("a: %d\n", a); /* 10 */
    printf("b: %d\n", b); /* 5 */

    return 0;
}
```

A função main

- O programa principal é uma função especial (`main`), que possui um tipo fixo (`int`) e é invocada automaticamente pelo sistema operacional quando este inicia a execução do programa.
- O comando `return` da função `main` informa ao sistema operacional se o programa funcionou corretamente ou não. O padrão é que um programa retorne zero, caso tenha funcionado corretamente, ou qualquer outro valor, caso contrário.
- Exemplo:

```
#include <stdio.h>

int main() {
    printf("Hello, World!\n");

    return 0;
}
```

O tipo void

- O tipo void é um tipo especial.
- Este tipo é utilizado para indicar que uma função não retorna nenhum valor.
- Por exemplo, a função abaixo verifica se um número inteiro fornecido como parâmetro é par ou ímpar, e imprime uma mensagem correspondente:

```
void parOuImpar(int numero) {  
    if (numero % 2)  
        printf("Impar\n");  
    else  
        printf("Par\n");  
}
```

O tipo void

```
#include <stdio.h>

void parOuImpar(int numero) {
    if (numero % 2)
        printf("Impar\n");
    else
        printf("Par\n");
}

int main() {
    parOuImpar(10);
    parOuImpar(21);

    return 0;
}
```


O tipo void

- Podemos usar o comando `return` (sem qualquer valor) para indicar explicitamente o fim de uma função do tipo `void`:

```
void parOuImpar(int numero) {  
    if (numero % 2)  
        printf("Impar\n");  
    else  
        printf("Par\n");  
  
    return;  
}
```

Exemplo de função do void

```
#include <stdio.h>

void parOuImpar(int numero) {
    if (numero % 2)
        printf("Impar\n");
    else
        printf("Par\n");

    return;
}

int main() {
    parOuImpar(10);
    parOuImpar(21);

    return 0;
}
```

Definindo funções depois da `main`

- Até o momento, aprendemos definir as funções antes do programa principal (função `main`).
- O que ocorreria se definíssemos uma função depois da função `main`?
 - ▶ Ocorreria um erro de compilação.

Definindo funções depois da main

```
#include <stdio.h>

int main() {
    float a = 8, b = 5;
    printf("%f\n", soma(a, b)); /* Erro funcao desconhecida */
    return 0;
}

float soma(float op1, float op2) {
    return (op1 + op2);
}
```

Declarando uma função sem defini-la

- Para organizar melhor um programa e podermos implementar funções em partes distintas do arquivo, *protótipos de funções* são utilizados.
- Protótipos de funções correspondem à primeira linha da definição de uma função contendo tipo de retorno, nome da função, parâmetros e um ponto e vírgula.

```
tipo nome(tipo parâmetro1, ..., tipo parâmetroN);
```

- O protótipo de uma função deve vir sempre antes do seu uso.
- É comum colocar os protótipos de funções no início do arquivo do programa.

Protótipos de funções

```
#include <stdio.h>

float soma(float op1, float op2);

int main() {
    float a = 8, b = 5;
    printf("%f\n", soma(a, b));
    return 0;
}

float soma(float op1, float op2) {
    return (op1 + op2);
}
```

Protótipos de funções

```
#include <stdio.h>

float soma(float op1, float op2);
float subt(float op1, float op2);

int main() {
    float a = 8, b = 5;
    printf("%f\n %f\n", soma(a, b), subt(a, b));
    return 0;
}

float soma(float op1, float op2) {
    return (op1 + op2);
}

float subt(float op1, float op2) {
    return (op1 - op2);
}
```

Números primos

- Em aulas anteriores, vimos como testar se um número é primo:

```
primo = 1;

for (divisor = 2; primo && (divisor <= candidato/2); divisor++)
    if ((candidato % divisor) == 0)
        primo = 0;

if (primo)
    printf("%d", candidato);
```

- Vimos também como escrever um programa para imprimir os n primeiros números primos.

Números primos

```
#include <stdio.h>

int main() {
    int divisor, candidato = 2, primos = 0, n, primo;

    printf("Numero de primos a serem calculados: ");
    scanf("%d",&n);

    while (primos < n) {
        primo = 1;

        for (divisor = 2; primo && (divisor <= candidato/2); divisor++)
            if ((candidato % divisor) == 0)
                primo = 0;

        if (primo) {
            printf("%d\n", candidato);
            primos++;
        }
        candidato++;
    }
    return 0;
}
```

Números primos

- Se o número de primos a serem calculados for negativo, usaremos o valor absoluto deste.
- Como refazer este código utilizando funções?
- Podemos criar uma função que testa se um número é primo ou não (note que este é um bloco logicamente bem definido).
- Vamos criar também uma função que retorna o valor absoluto de um número.
- Depois fazemos chamadas para estas funções.

Números primos

```
#include <stdio.h>

int testaPrimo(int candidato);
int valorAbs(int x);

int main() {
    int n, candidato = 2, primos = 0;

    printf("Numero de primos a serem calculados: ");
    scanf("%d", &n);
    n = valorAbs(n);

    while (primos < n) {
        if (testaPrimo(candidato)) {
            printf("%d\n", candidato);
            primos++;
        }
        candidato++;
    }

    return 0;
}
```

Números primos

```
/* Calcula o valor absoluto de x */  
int valorAbs(int x) {  
    if (x < 0)  
        return -x;  
    else  
        return x;  
}
```

Números primos

```
/* Verifica se um numero candidato eh primo */
int testaPrimo(int candidato) {
    int primo = 1, divisor;

    for (divisor = 2; primo && (divisor <= candidato/2); divisor++)
        if ((candidato % divisor) == 0)
            primo = 0;

    if (primo)
        return 1; /* Se for primo, retorna 1 (verdadeiro) */
    else
        return 0; /* Se nao for, retorna 0 (falso) */
}
```

Números primos

- As funções aumentam a clareza do código.
- Também tornam mais simples modificações no código.
- Exemplo: melhorar o teste de primalidade.
 - ▶ Testar se o candidato é um número par.
 - ▶ Se for ímpar, testar apenas divisores ímpares (3, 5, 7, etc).
- O uso de funções facilita a manutenção do código.
- Neste caso, basta alterar a função `testaPrimo`.

Números primos

```
int testaPrimo(int candidato) {
    int primo = 1, divisor;

    for (divisor = 2; primo && (divisor <= candidato/2); divisor++)
        if ((candidato % divisor) == 0)
            primo = 0;

    if (primo)
        return 1;
    else
        return 0;
}
```

Números primos

```
int testaPrimo(int candidato) {
    int primo = 1, divisor;

    if ((candidato % 2) == 0)
        if (candidato == 2)
            return 1;
        else
            return 0;

    for (divisor = 3; primo && (divisor <= candidato/2); divisor = divisor + 2)
        if ((candidato % divisor) == 0)
            primo = 0;

    if (primo)
        return 1;
    else
        return 0;
}
```


Números primos

```
int testaPrimo(int candidato) {
    int primo = 1, divisor;

    if ((candidato % 2) == 0)
        return (candidato == 2);

    for (divisor = 3; primo && (divisor <= candidato/2); divisor = divisor + 2)
        if ((candidato % divisor) == 0)
            primo = 0;

    return primo;
}
```

Números primos

```
int testaPrimo(int candidato) {
    int primo = 1, divisor;

    if ((candidato % 2) == 0)
        return (candidato == 2);

    for (divisor = 3; primo && (divisor <= candidato/2); divisor = divisor + 2)
        if ((candidato % divisor) == 0)
            primo = 0;

    return primo;
}
```

Variáveis locais e variáveis globais

- Uma variável é chamada local se ela foi declarada dentro de uma função. Nesse caso, ela existe somente dentro daquela função e, após o término da execução da mesma, a variável deixa de existir.
- Variáveis utilizadas como parâmetros de funções também são locais.
- Uma variável é chamada global se ela for declarada fora de qualquer função. Essa variável é visível em todas as funções. Qualquer função pode alterá-la e ela existe durante toda a execução do programa.
- Variáveis globais dificultam a legibilidade, manutenção e reuso de funções e, por estes motivos, devem ser evitadas.

Variáveis locais e variáveis globais

```
#Declaração de bibliotecas
```

```
#Definição de constantes
```

```
Protótipos de funções;
```

```
Declaração de variáveis globais;
```

```
int main() {  
    Declaração de variáveis locais;  
    Comandos;  
}
```

```
int funcao1(parâmetros) {  
    Declaração de variáveis locais;  
    Comandos;  
}
```

```
int funcao2(parâmetros) {  
    Declaração de variáveis locais;  
    Comandos;  
}
```

Escopo de variáveis

- O escopo de uma variável determina em quais partes do código pode-se ter acesso a ela.
- A regra de escopo em C é bem simples:
 - ▶ As variáveis globais são visíveis por todas as funções.
 - ▶ As variáveis locais são visíveis apenas na função onde foram declaradas.

Escopo de variáveis

```
#include <stdio.h>

void funcao1();
int funcao2(int local_b);

int global_a;

int main() {
    int local_main;
    /* Neste ponto sao visiveis global_a e local_main */
}

void funcao1() {
    int local_a;
    /* Neste ponto sao visiveis global_a e local_a */
}

int funcao2(int local_b) {
    int local_c;
    /* Neste ponto sao visiveis global_a, local_b e local_c */
}
```

Escopo de variáveis

- É possível declarar variáveis locais com o mesmo nome de variáveis globais.
- Nesta situação, a variável local “esconde” a variável global.

```
int nota = 10;

void a() {
    int nota;

    ...

    /* Altera o valor da variavel local,
       sem afetar a variavel global */
    nota = 5;
}
```

Escopo de variáveis

```
#include <stdio.h>

int x = 1;

void funcao1() {
    x = 3;
    printf("%d\n", x);
}

void funcao2() {
    int x = 4;
    printf("%d\n", x);
}

int main() {
    x = 2;
    funcao1();
    funcao2();
    printf("%d\n", x);
    return 0;
}
```


Escopo de variáveis

```
#include <stdio.h>

int x = 1;

void funcao1() {
    x = 3;
    printf("%d\n", x); /* 3 */
}

void funcao2() {
    int x = 4;
    printf("%d\n", x); /* 4 */
}

int main() {
    x = 2;
    funcao1();
    funcao2();
    printf("%d\n", x); /* 3 */
    return 0;
}
```

Vetores e funções

- Vetores também podem ser passados como parâmetros em funções.
- Ao contrário dos tipos simples, vetores têm um comportamento diferente quando usados como parâmetros de funções.
- Quando uma variável simples é passada como parâmetro, seu valor é atribuído para uma nova variável local da função.
- No caso de vetores, não é criado um novo vetor.
- Isto significa que os valores de um vetor podem ser alterados dentro de uma função.

Vetores e funções

- Para indicar que um parâmetro é um vetor, usamos [] na frente do nome do parâmetro. Exemplo:

```
int max(int vetor[], int tam) {  
    ...  
}
```

- Ao chamar uma função que possui um vetor como parâmetro, devemos apenas indicar o nome do vetor a ser fornecido para a função (sem usar [] na frente do nome do vetor). Exemplo:

```
int elementos[10], n = 10, x;  
...  
x = max(elementos, n);
```

Vetores e funções

```
#include <stdio.h>

void funcao(int vetor[], int tam, int valor) {
    int i;
    for (i = 0; i < tam; i++)
        vetor[i] = valor;
}

int main() {
    int v[10], i;

    for (i = 0; i < 10; i++)
        v[i] = 8;

    funcao(v, 6, 5);
    for (i = 0; i < 10; i++)
        printf("%d ", v[i]);

    return 0;
}
```

Vetores e funções

```
#include <stdio.h>

void funcao(int vetor[], int tam, int valor) {
    int i;
    for (i = 0; i < tam; i++)
        vetor[i] = valor;
}

int main() {
    int v[10], i;

    for (i = 0; i < 10; i++)
        v[i] = 8;

    funcao(v, 6, 5);
    for (i = 0; i < 10; i++)
        printf("%d ", v[i]); /* 5 5 5 5 5 5 8 8 8 8 */

    return 0;
}
```

Vetores e funções

- Vetores não podem ser usados como retorno de uma função.

```
int[] leVetor(int tam) { /* Erro */
    int i, vetor[100];

    for (i = 0; i < tam; i++) {
        printf("Digite um numero: ");
        scanf("%d", &vetor[i]);
    }

    return vetor;
}
```

- O código acima não compila, pois não podemos retornar um `int []`.
- Entretanto, podemos fazer algo semelhante usando o fato de que vetores são alterados dentro de funções.

Vetores e funções

```
#include <stdio.h>
```

```
void leVetor(int vetor[], int tam) {  
    int i;  
    for (i = 0; i < tam; i++) {  
        printf("Digite numero: ");  
        scanf("%d", &vetor[i]);  
    }  
}
```

```
void imprimeVetor(int vetor[], int tam) {  
    int i;  
    for (i = 0; i < tam; i++)  
        printf("vetor[%d] = %d\n", i, vetor[i]);  
}
```

Vetores e funções

```
int main() {
    int vetor1[10], vetor2[20];

    printf("----- Vetor 1 -----\\n");
    leVetor(vetor1, 10);
    printf("----- Vetor 2 -----\\n");
    leVetor(vetor2, 20);

    printf("----- Vetor 1 -----\\n");
    imprimeVetor(vetor1, 10);
    printf("----- Vetor 2 -----\\n");
    imprimeVetor(vetor2, 20);

    return 0;
}
```


Matrizes e funções

- Ao passar um vetor como parâmetro, não é necessário fornecer o seu tamanho na declaração da função.
- Quando usamos uma matriz, a possibilidade de não informar o tamanho na declaração se restringe à primeira dimensão.
- Matrizes são de fato alocadas pelo compilador como vetores, por isso é preciso informar todas as dimensões (com exceção, eventualmente, da primeira) para que o compilador seja capaz de determinar corretamente os elementos de uma matriz passada como parâmetro para uma função.
- Assim como vetores, matrizes passadas como parâmetros podem ser modificadas dentro das funções.

Matrizes e funções

- Pode-se criar uma função deixando de indicar a primeira dimensão:

```
void mostra_matriz(int mat[][10], int linhas) {  
    ...  
}
```

- Ou pode-se criar uma função indicando todas as dimensões:

```
void mostra_matriz(int mat[5][10], int linhas) {  
    ...  
}
```

- Não se pode deixar de indicar outras dimensões (exceto a primeira):

```
void mostra_matriz(int mat[5][], int linhas) {  
    /* Esta funcao nao compila */  
    ...  
}
```

Exemplo

```
#include <stdio.h>

void imprime_matriz(int linhas, int colunas, int matriz[][10]) {
    int i, j;
    for (i = 0; i < linhas; i++) {
        for (j = 0; j < colunas; j++)
            printf("%2d ", matriz[i][j]);
        printf("\n");
    }
}

int main() {
    int matriz[8][10] = { { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9},
                          {10, 11, 12, 13, 14, 15, 16, 17, 18, 19},
                          {20, 21, 22, 23, 24, 25, 26, 27, 28, 29},
                          {30, 31, 32, 33, 34, 35, 36, 37, 38, 39},
                          {40, 41, 42, 43, 44, 45, 46, 47, 48, 49},
                          {50, 51, 52, 53, 54, 55, 56, 57, 58, 59},
                          {60, 61, 62, 63, 64, 65, 66, 67, 68, 69},
                          {70, 71, 72, 73, 74, 75, 76, 77, 78, 79} };

    imprime_matriz(8, 10, matriz);
    return 0;
}
```

Exercícios

- Escreva uma função que, dados dois números inteiros positivos, calcule e retorne o Máximo Divisor Comum (MDC) entre eles.
- Escreva uma função que, dados dois números inteiros positivos, calcule e retorne o Mínimo Múltiplo Comum (MMC) entre eles.
- Escreva um função que, dados um número real x e um número inteiro y , calcule e retorne o valor de x^y .
- Escreva um função que, dado um número inteiro positivo, imprima o número invertido. Por exemplo, se o número 3248700 for fornecido para função, ela deve imprimir 0078423.
- Escreva um função que, dado um número inteiro $n > 1$, imprima a fatoração em números primos de n . Por exemplo, se o número 936936 for fornecido para função, ela deve imprimir:

$$936936 = 1 \times 2^3 \times 3^2 \times 7 \times 11 \times 13^2$$

Exercícios

- Escreva um função que, dado um número inteiro positivo n e um vetor de n números inteiros, verifique se o vetor está ordenado de forma crescente.
- Escreva um função que, dado um número inteiro positivo n e um vetor de n números inteiros, compute o número de inversões do vetor. Dado um vetor π , dizemos que o par de elementos (π_i, π_j) é uma inversão se $i < j$ e $\pi_i > \pi_j$. Por exemplo, o vetor `[2 3 5 1 4]` contém 4 inversões: $(2, 1)$, $(3, 1)$, $(5, 1)$ e $(5, 4)$.
- Escreva uma função que calcule o tamanho de uma string dada (equivalente a função `strlen` da biblioteca `string.h`).
- Escreva uma função que inverta a ordem dos caracteres de uma string dada.
- Escreva um função que, dado um número inteiro positivo n e uma matriz quadrada $n \times n$, verifique se a matriz é simétrica.

Mínimo Múltiplo Comum (MMC)

```
int mmc(int x, int y) {  
    int resultado = 1;  
  
    while ((resultado % x) || (resultado % y))  
        resultado++;  
  
    return resultado;  
}
```

Mínimo Múltiplo Comum (MMC)

```
int mmc(int x, int y) {  
    int resultado = x;  
  
    while (resultado % y)  
        resultado += x;  
  
    return resultado;  
}
```

Mínimo Múltiplo Comum (MMC)

```
int mmc(int x, int y) {  
    int aux, resultado;  
  
    if (x < y) {  
        aux = x;  
        x = y;  
        y = aux;  
    }  
  
    resultado = x;  
  
    while (resultado % y)  
        resultado += x;  
  
    return resultado;  
}
```


Mínimo Múltiplo Comum (MMC)

```
int mmc(int x, int y) {
    int resultado;

    if (x < y)
        return mmc(y, x);

    resultado = x;

    while (resultado % y)
        resultado += x;

    return resultado;
}
```

Mínimo Múltiplo Comum (MMC)

```
int mmc(int x, int y) {
    int resultado = 1, divisor = 2;

    while ((x != 1) || (y != 1))
        if ((x % divisor == 0) || (y % divisor == 0)) {
            resultado = resultado * divisor;

            if (x % divisor == 0)
                x = x / divisor;
            if (y % divisor == 0)
                y = y / divisor;
        } else
            divisor++;

    return resultado;
}
```

Mínimo Múltiplo Comum (MMC)

```
int mmc(int x, int y) {  
    return (x * y) / mdc(x, y);  
}
```

Vetor ordenado

```
int ordenado(int n, int vetor[]) {  
    int i, ok = 1;  
  
    for (i = 0; (i < n - 1) && ok; i++)  
        if (vetor[i] > vetor[i + 1])  
            ok = 0;  
  
    return ok;  
}
```

Vetor ordenado

```
int ordenado(int n, int vetor[]) {  
    int i;  
  
    for (i = 0; i < n - 1; i++)  
        if (vetor[i] > vetor[i + 1])  
            return 0;  
  
    return 1;  
}
```

Matriz simétrica

```
int simetrica(int n, int matriz[][MAX]) {
    int i, j;

    for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
            if (matriz[i][j] != matriz[j][i])
                return 0;

    return 1;
}
```

Matriz simétrica

```
int simetrica(int n, int matriz[][MAX]) {
    int i, j;

    for (i = 0; i < n; i++)
        for (j = 0; j < i; j++)
            if (matriz[i][j] != matriz[j][i])
                return 0;

    return 1;
}
```

Matriz simétrica

```
int simetrica(int n, int matriz[][MAX]) {
    int i, j;

    for (i = 0; i < n; i++)
        for (j = i + 1; j < n; j++)
            if (matriz[i][j] != matriz[j][i])
                return 0;

    return 1;
}
```