

# Linguagem de Programação II

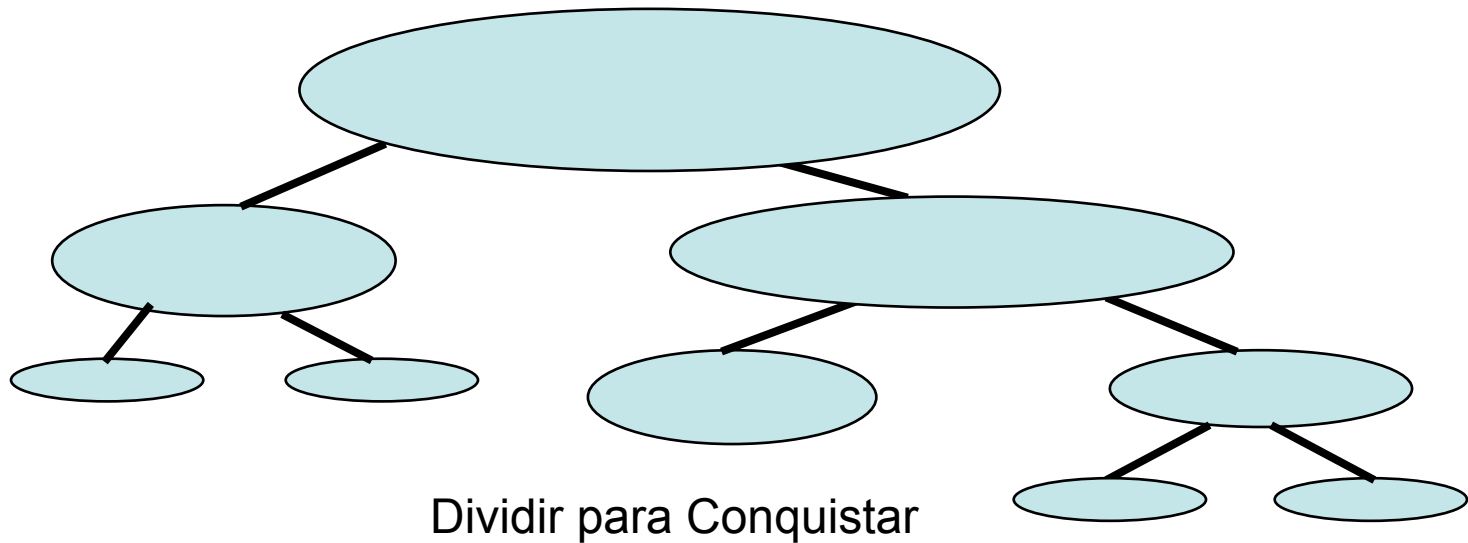
Prof. Mario Bessa

Aula 14

<http://mariobessa.info>

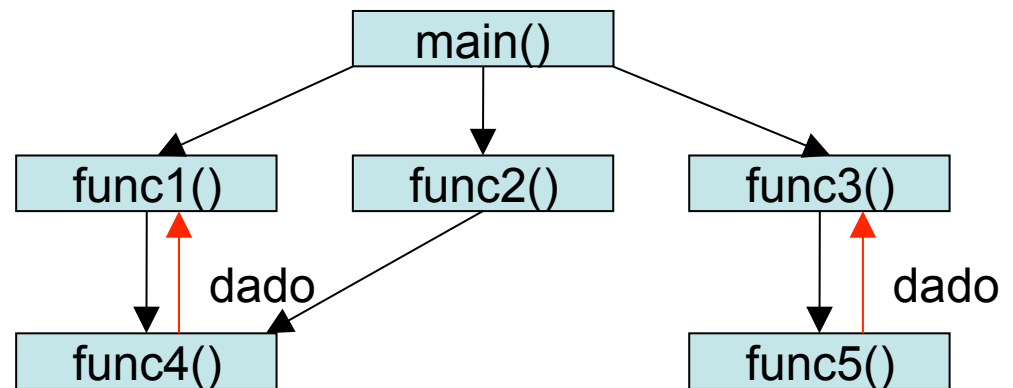
# Programação Modular com Funções

- Como resolver um problema grande/complexo?
- Dividindo-o em pequenos módulos (funções).



# Programação Modular com Funções

- Em C usa-se **funções** (**módulos**) para executar uma tarefa específica de uma solução.



Mostra um programa separado em tarefas (funções).

NOTA: não indica a sequência de passos no programa!

# Vantagem do uso de módulos

- Módulos podem ser escritos e testados separadamente.
- Módulos podem ser reusados.
- Grandes projetos podem ser desenvolvidos paralelamente.
- Diminui o tamanho dos programas, tornando-os mais legíveis.
- Promove o conceito de **abstração**
  - Um módulo esconde os detalhes de uma tarefa.
  - É necessário apenas saber o quê esse módulo faz e NÃO com ele faz uma tarefa.

# Refinamentos Sucessivos

- **Motivação:**
  - Dividir tarefas grandes em tarefas menores:
    - Facilita manutenção do código.
    - As partes podem ser testadas separadamente.
  - Reaproveitar código existente:
    - Evitar a reescrita de um mesmo código repetidas vezes.
    - Integração com códigos de outros programadores.
  - Esconder detalhes:
    - Permite uma visão de mais alto nível.
    - Viabiliza programas mais complexos.

# Funções

- **Definição:**

- Uma função é uma unidade de código de programa autônoma desenhada para cumprir uma tarefa particular (ex: `printf`, `sqrt`, `strcmp`).

- **Sintaxe:**

```
tipo nome_da_função(lista de argumentos) {  
    Declarações de variáveis locais;  
    Bloco de comandos;  
}
```

# Funções

- Exemplo:

```
#include <stdio.h>

int resto(int a, int b){
    int q;
    q = a/b;
    return (a-q*b);
}

int main(){
    int r;

    r = resto(5,3);
    printf("Resto: %d\n",r);

    return 0;
}
```

# Funções

- Exemplo:

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

```
int resto(int a, int b) {
```

```
    int q;
```

```
    q = a/b;
```

```
    return (a-q*b);
```

```
}
```

Corpo da função

```
int main() {
```

```
    int r;
```

```
    r = resto(5, 3);
```

```
    printf("Resto: %d\n", r);
```

```
    return 0;
```

```
}
```



# Funções

- Exemplo:

```
#include <stdio.h>

int resto(int a, int b){
    int q;
    q = a/b;
    return (a-q*b);
}

int main(){
    int r;

    r = resto(5,3);
    printf("Resto: %d\n",r);

    return 0;
}
```

Lista de argumentos  
(parâmetros)

# Funções

- Exemplo:


```
#include <stdio.h>

int resto(int a, int b){
    int q;
    q = a/b;
    return (a-q*b);
}

int main(){
    int r;

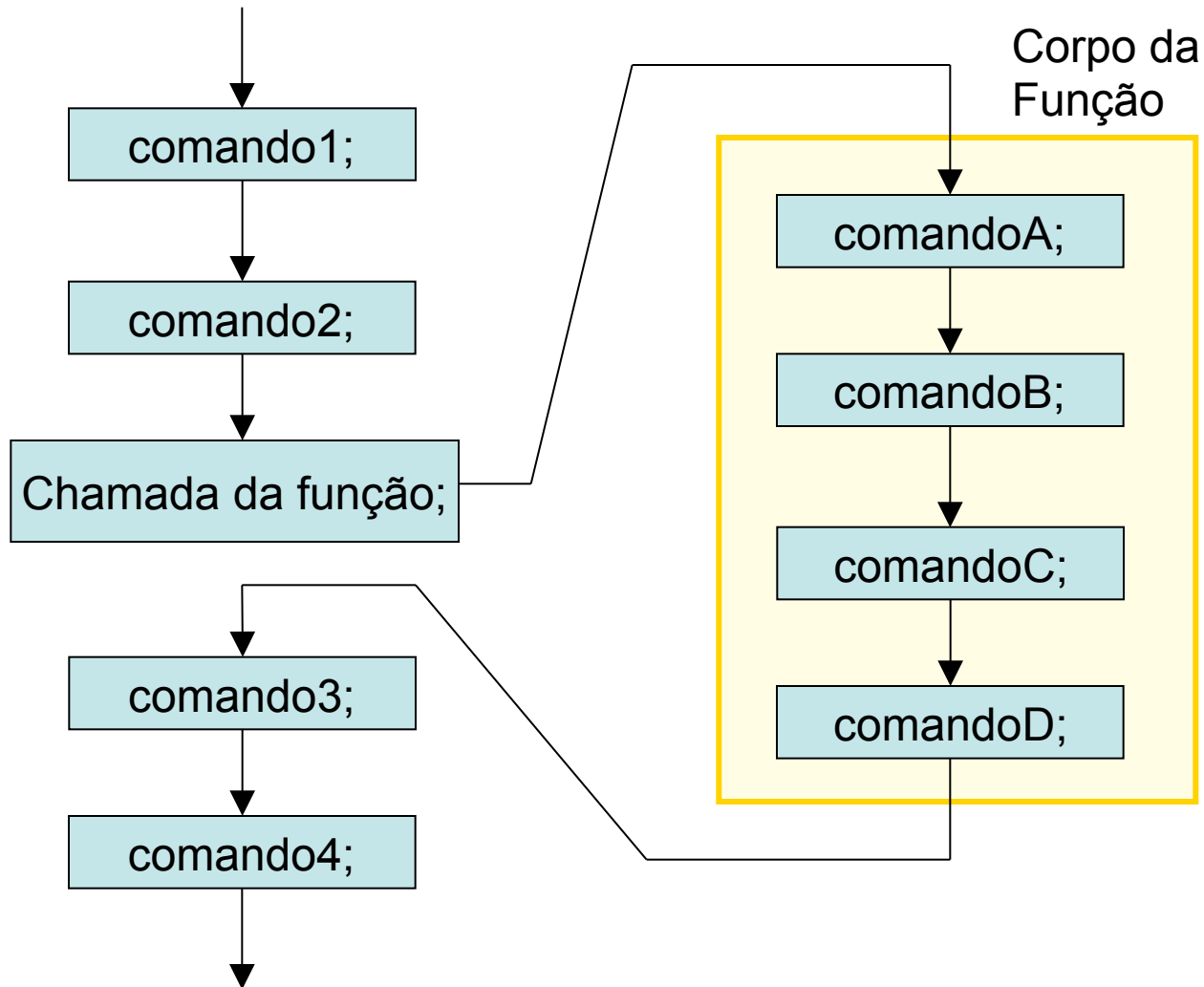
    r = resto(5, 3);
    printf("Resto: %d\n", r);

    return 0;
}
```



Chamada da função

# Chamando Funções



# Chamando Funções

- **Chamada a uma função:**
  - Aloca espaço para todas variáveis locais declaradas na função.
  - Inicializa os parâmetros com os valores passados para a função.
  - Desvia a execução do código para o corpo da função.
- **Após execução:**
  - Desaloca memória das variáveis locais.
  - Retorna para a próxima instrução após a chamada.

# Variáveis locais

- Variáveis declaradas dentro de uma função e argumentos são chamados de variáveis locais.
  - Existem somente durante a execução da função.
  - Podem ser acessadas apenas no corpo da função onde foram declaradas.
- **OBS:** Variáveis locais com mesmo nome declaradas em funções diferentes são variáveis distintas.

# Variáveis locais

- Variáveis declaradas dentro de uma função e argumentos são chamados de variáveis locais.
  - Existem somente durante a execução da função.
  - Podem ser acessadas apenas no corpo da função onde foram declaradas.
- **OBS:** Variáveis locais com mesmo nome declaradas em funções diferentes são variáveis distintas.

```
#include <stdio.h>

int resto(int a, int b) {
    int q;
    q = a/b;
    return (a-q*b);
}
```

Possui 3 variáveis locais:

`int a, int b, int q`

# Valor de Retorno

- Funções podem retornar um valor (ex: **sqrt** retorna a raiz quadrada de um número).
- O valor retornado deve ser **compatível** com o tipo da função.
- O valor é retornado através do comando **return** que causa a **saída imediata** da função.
- Funções que não retornam nada são chamadas de procedimentos e em C são implementadas como sendo do tipo **void**.

# Exemplo: função retornando valor

```
#include <stdio.h>

int resto(int a, int b) {
    int q;
    q = a/b;
    return (a-q*b);
}

• → int main() {
    int r;

    r = resto(5, 3);
    printf("Resto: %d\n", r);

    return 0;
}
```

- O programa inicia a execução na função principal (**main**).



# Exemplo: função retornando valor

```
#include <stdio.h>

int resto(int a, int b) {
    int q;
    q = a/b;
    return (a-q*b);
}

int main() {
    • → int r;

    r = resto(5, 3);
    printf("Resto: %d\n", r);

    return 0;
}
```

- É alocado espaço para as variáveis locais da função principal (**main**).

int r



# Exemplo: função retornando valor

```
#include <stdio.h>

int resto(int a, int b){
    int q;
    q = a/b;
    return (a-q*b);
}


int main(){
    int r;

    → r = resto(5,3);
    printf("Resto: %d\n",r);

    return 0;
}
```

- A função **resto** é chamada.

int r



# Exemplo: função retornando valor

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

```
int resto(int a, int b){  
    int q;  
    q = a/b;  
    return (a-q*b);  
}
```

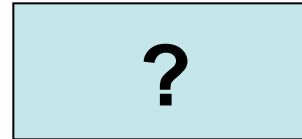
```
int main(){  
    int r;
```

```
    r = resto(5,3);  
    printf("Resto: %d\n",r);
```

```
    return 0;  
}
```

- É alocado espaço para as variáveis locais da função.

int r



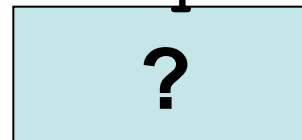
int a



int b



int q



# Exemplo: função retornando valor

```
#include <stdio.h>

int resto(int a, int b) {
    int q;
    q = a/b;
    return (a-q*b);
}

int main() {
    int r;

    r = resto(5, 3);
    printf("Resto: %d\n", r);

    return 0;
}
```

- Os parâmetros são inicializados com os valores passados para a função.

int r  
?

int a  
5

int b  
3

int q  
?

# Exemplo: função retornando valor

```
#include <stdio.h>

int resto(int a, int b){
    int q;
    → q = a/b;
    return (a-q*b);
}


int main(){
    int r;

    → r = resto(5,3);
    printf("Resto: %d\n",r);


    return 0;
}
```

- A variável **q** recebe o resultado da divisão inteira.


int r

A light blue rectangular box representing memory for variable 'r', containing a large black question mark.


int a

A light blue rectangular box representing memory for variable 'a', containing the red number 5.

int b

A light blue rectangular box representing memory for variable 'b', containing the red number 3.

int q

A light blue rectangular box representing memory for variable 'q', containing the red number 1.

# Exemplo: função retornando valor

```
#include <stdio.h>

int resto(int a, int b){
    int q;
    q = a/b;
    → return (a-q*b);
}

int main(){
    int r;

    → r = resto(5,3);
    printf("Resto: %d\n",r);

    return 0;
}
```

- O resultado **2** da expressão **(a-q\*b)** é calculado e retornado pela função.

int r



int a



int b



int q



# Exemplo: função retornando valor

```
#include <stdio.h>

int resto(int a, int b){
    int q;
    q = a/b;
    return (a-q*b);
}

int main(){
    int r;

    → r = resto(5,3);
    printf("Resto: %d\n",r);

    return 0;
}
```

- A variável **r** recebe o valor de retorno da função e as variáveis locais da função são destruídas.

int r

2

int a

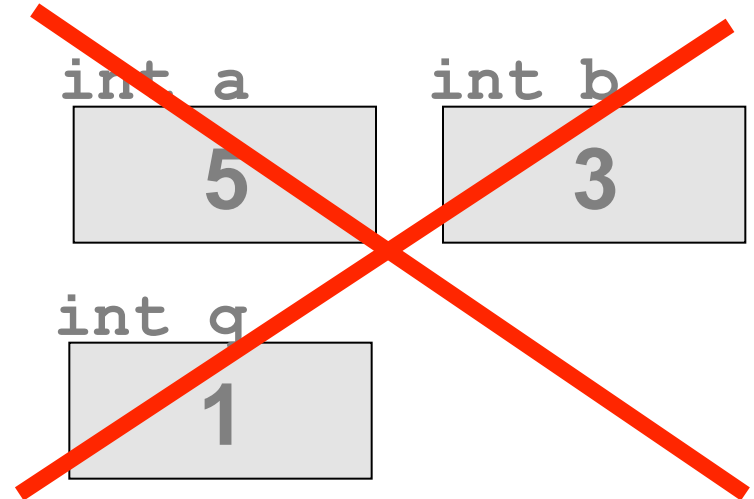
5

int b

3

int q

1



# Exemplo: função retornando valor

```
#include <stdio.h>

int resto(int a, int b){
    int q;
    q = a/b;
    return (a-q*b);
}


int main(){
    int r;

    r = resto(5,3);
    • → printf("Resto: %d\n",r);

    return 0;
}
```

- O conteúdo da variável **r** é impresso na saída padrão.

int r



2



# Exemplo: função retornando valor

```
#include <stdio.h>

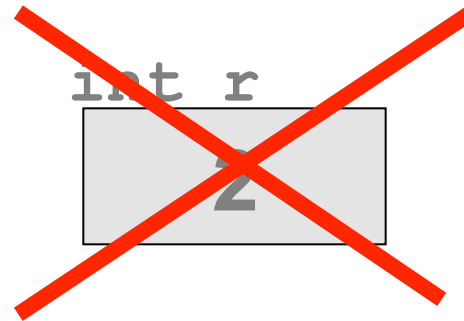
int resto(int a, int b){
    int q;
    q = a/b;
    return (a-q*b);
}

int main(){
    int r;

    r = resto(5,3);
    printf("Resto: %d\n",r);

    return 0;
}
```

- A função principal retorna **0** e a execução do programa é encerrada.



- As variáveis locais da função principal são destruídas.

# Exemplo: função sem retorno (void)

- Escreva um programa para gerar a saída abaixo:

```
*  
**  
***  
****  
*****
```

```
for (i=1; i<=5; i++) {  
    for (j=1; j<=i; j++)  
        printf("*");  
    printf("\n");  
}
```

```
#include <stdio.h>  
void print_i_star(int i) {  
    int j;  
    for (j=1; j<=i; j++)  
        printf("*");  
    printf("\n");  
    return;  
}  
int main() {  
    int i;  
    for (i=1; i<=5; i++) {  
        print_i_star( i );  
    }  
    return 0;  
}
```

# Exemplo: função sem retorno (void)

```
#include <stdio.h>
void print_i_star(int i){
    int j;
    for (j=1; j<=i; j++)
        printf("*");
    printf("\n");
    return;
}
• → int main(){
    int i;
    for (i=1; i<=5; i++) {
        print_i_star( i );
    }
    return 0;
}
```

- O programa inicia a execução na função principal (**main**).

# Exemplo: função sem retorno (void)

```
#include <stdio.h>
void print_i_star(int i){
    int j;
    for (j=1; j<=i; j++)
        printf("*");
    printf("\n");
    return;
}
int main(){
    int i;
    for (i=1; i<=5; i++) {
        print_i_star( i );
    }
    return 0;
}
```

- É alocado espaço para as variáveis locais da função principal (**main**).

int i


A light blue rectangular box with a black border, containing a large black question mark. This represents the memory space allocated for the local variable 'i' in the main function, which is currently unknown or uninitialized.

# Exemplo: função sem retorno (void)

```
#include <stdio.h>
void print_i_star(int i){
    int j;
    for (j=1; j<=i; j++)
        printf("*");
    printf("\n");
    return;
}
int main(){
    int i;
    • → for (i=1; i<=5; i++) {
        print_i_star( i );
    }
    return 0;
}
```

- A variável local recebe um valor.

int i




1

# Exemplo: função sem retorno (void)

```
#include <stdio.h>
void print_i_star(int i){
    int j;
    for (j=1; j<=i; j++)
        printf("*");
    printf("\n");
    return;
}
int main(){
    int i;
    for (i=1; i<=5; i++) {
        • → print_i_star( i );
    }
    return 0;
}
```

- A função `print_i_star` é chamada.

`int i`



1

# Exemplo: função sem retorno (void)

```
#include <stdio.h>
void print_i_star(int k) {
    int j;
    for (j=1; j<=k; j++)
        printf("*");
    printf("\n");
    return;
}
int main() {
    int i;
    for (i=1; i<=5; i++) {
        print_i_star( i );
    }
    return 0;
}
```

- É alocado espaço para as variáveis locais da função.

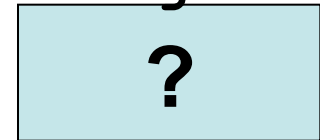
int i



int k



int j



# Exemplo: função sem retorno (void)

```
#include <stdio.h>
void print_i_star(int k){
    int j;
    for (j=1; j<=k; j++)
        printf("*");
    printf("\n");
    return;
}
int main(){
    int i;
    for (i=1; i<=5; i++) {
        print_i_star( i );
    }
    return 0;
}
```

- Os parâmetros são inicializados com os valores passados para a função.

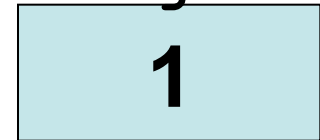
int i



int k



int j





# Exemplo: função sem retorno (void)

```
#include <stdio.h>
void print_i_star(int k){
    int j;
    for (j=1; j<=k; j++)
        •→ printf("*");
    printf("\n");
    return;
}
int main(){
    int i;
    for (i=1; i<=5; i++) {
        •→ print_i_star( i );
    }
    return 0;
}
```

- Saída: \*

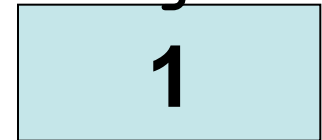
int i



int k



int j



# Exemplo: função sem retorno (void)

```
#include <stdio.h>
void print_i_star(int k){
    int j;
    for (j=1; j<=k; j++)
        printf("*");
    printf("\n");
    return;
}
int main(){
    int i;
    for (i=1; i<=5; i++) {
        print_i_star( i );
    }
    return 0;
}
```

- Saída: \* (salto de linha)

int i

1

int k

1

int j

1

# Exemplo: função sem retorno (void)

```
#include <stdio.h>
void print_i_star(int k){
    int j;
    for (j=1; j<=k; j++)
        printf("*");
    printf("\n");
    return;
}
int main(){
    int i;
    for (i=1; i<=5; i++) {
        print_i_star( i );
    }
    return 0;
}
```

- Retorno da função.

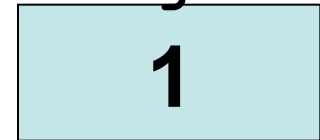
int i



int k



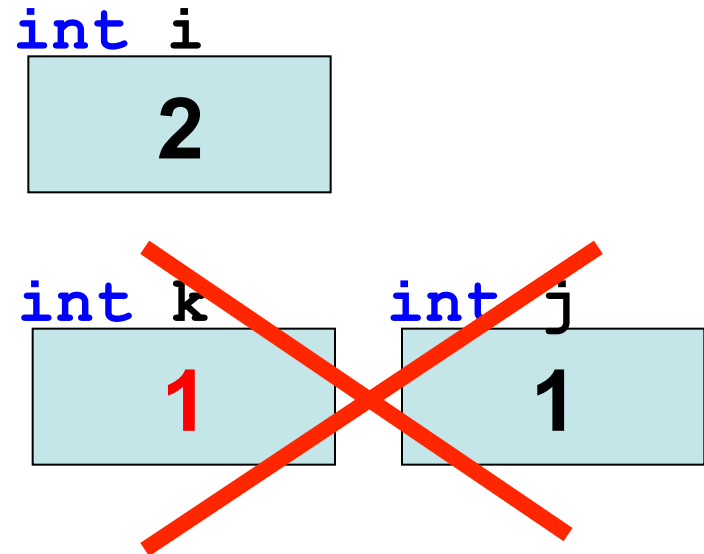
int j



# Exemplo: função sem retorno (void)

```
#include <stdio.h>
void print_i_star(int k){
    int j;
    for (j=1; j<=k; j++)
        printf("*");
    printf("\n");
    return;
}
int main(){
    int i;
    • → for (i=1; i<=5; i++) {
        print_i_star( i );
    }
    return 0;
}
```

- A variável local é incrementada.



# Variáveis globais

- Variáveis globais são declaradas fora de todas as funções do programa.
- Elas são conhecidas e podem ser alteradas por todas as funções do programa.
- Quando uma função tem uma variável local com o mesmo nome de uma variável global a função dará preferência à variável local.
- As variáveis globais devem ser evitadas.

# Exemplo de variáveis globais

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

```
int a = 10;
```

```
void imprimir () {  
    printf("%d\n", a++);  
}
```

```
int main () {  
    imprimir();  
    printf("%d\n", a++);  
    printf("%d\n", a);  
    return 0;  
}
```

# Exemplo de variáveis globais

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

```
int a = 10;
```

```
void imprimir (int a) {  
    printf("%d\n", a++);  
}
```

```
int main () {  
    imprimir(a);  
    printf("%d\n", a++);  
    return 0;  
}
```

# Passagem de parâmetro

- Por valor
  - Passamos o valor da variável.
  - Esses valores são ***copiados*** (atribuídos) para variáveis locais (parâmetros) da função.
  - Alterações nos parâmetros dentro da função **NÃO** alteram os valores das variáveis que foram passados



# Passagem de parâmetro por valor

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

```
void troca (int x, int y) {
```

```
    int temp;
```

```
    temp = x;
```

```
    x = y;
```

```
    y = temp;
```

```
}
```

```
int main() {
```

```
    int i,j;
```

```
    i=10;
```

```
    j=20;
```

```
    troca(i,j);
```

```
    printf("i:%d - j:%d\n", i, j);
```

```
    return 0;
```

```
}
```

Os valores de **i** e **j** são  
**copiados**  
para a função **troca**

0022FF74

**int x**

**10**

0022FF70

**int i**

**10**



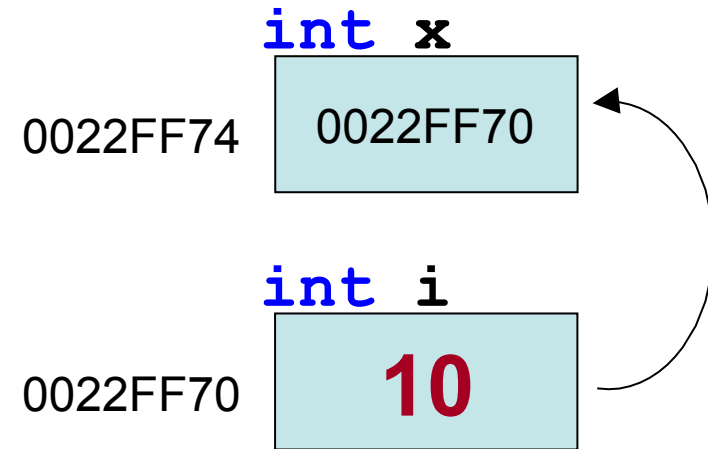
# Passagem de parâmetro

- Por referência (ponteiros)
  - Passamos o endereço da variável.
  - Esses endereços são ***copiados*** (atribuídos) para variáveis locais (parâmetros) da função.
  - Alterações nos parâmetros dentro da função **ALTERAM** os valores das variáveis que foram passados

# Passagem de parâmetro por referência

```
#include <stdio.h>
void troca(int *x, int *y) {
    int temp;
    temp = *x;
    *x = *y;
    *y = temp;
}
int main() {
    int i,j;
    i=10;
    j=20;
    troca(&i, &j);
    printf("i:%d - j:%d", i, j);
    return 0;
}
```

Os endereços de **i** e **j** são  
**copiados**  
para a função **troca**



# Problema

- Calcular o número de combinações de  $m$  elementos tomados  $p$  a  $p$  ( $p \leq m$ ), não importando a **ordem** dos elementos.

$$C(m, p) = \binom{m}{p} = \frac{m!}{p!(m-p)!}$$

# Solução Monolítica

```
#include <stdio.h>
int main(){
    int m,p,fatm=1,fatp=1,fatmp=1,mp;
    printf("Este programa calcula a combinação de m elementos tomados p a p (p<=m)\n");
    printf("Entre com o valor para m: ");
    scanf("%i",&m);
    printf("\nEntre com o valor para p: ");
    scanf("%i",&p);
    mp=m-p;
    while(m>1){                //fatorial de m
        fatm *= m;
        m--;
    }
    while(p>1){                //fatorial de p
        fatp *= p;
        p--;
    }
    while(mp>1){                //fatorial de m-p
        fatmp *= mp;
        mp--;
    }
    printf("Combinação= %i",fatm/(fatp*fatmp));
    return 0;
}
```

# Solução Modular

- **Função Fatorial:**

```
int fat(int x) {  
    int f = 1;  
  
    while(x>1) {  
        f *= x;  
        x--;  
    }  
    return f;  
}
```

# Solução Modular

```
#include <stdio.h>
int fat(int x){
    int f ;
    while(x>1){
        f *= x;
        x--;
    }
    return f;
}
int main(){
    int m,p;
    printf("Este programa calcula a combinação de m elementos tomados p a p (p<=m)\n");
    printf("Entre com o valor para m: ");
    scanf("%i",&m);
    printf("Entre com o valor para p: ");
    scanf("%i",&p);
    printf("Combinação de %i, %i a %i",p,p,fat(m)/(fat(p)*fat(m-p)));
    return 0;
}
```

# Problema

- Escreva um programa em C que leia um número e verifique se é primo. Crie uma função chamada *primo* para verificar essa condição.



# Solução Modular

```
#include <stdio.h>

int primo(int n){
    for (int i=2; i<n; i++) {
        if (n%i==0) {
            return 0;
        }
    }
    return 1;
}

int main(){
    int m;
    printf("Entre com um número ");
    scanf("%i",&m);
    if (m<1)
        printf("Número inválido");
    else if (primo(m))
        printf("%i é primo",m);
    else
        printf("%i não é primo",m);
    return 0;
}
```

# Arrays como argumentos de funções

- Quando o nome de uma variável simples (ex: `int`) é passado na chamada da função, é gerada uma cópia do conteúdo da variável fornecida em uma variável local da função. Esse processo é conhecido como **chamada por valor**.
  - **Ex:** Todos exemplos até agora usaram chamadas por valor.
- Vetores podem conter grandes quantidades de dados, logo copiar todos dados de um vetor em uma chamada por valor é ineficiente. Em C, vetores são passados **por referência** de modo que existe uma única cópia do vetor (a original). É passado apenas o endereço do vetor para a função.

# Arrays como argumentos de funções

- Quando uma matriz/vetor é passado como argumento de uma função, apenas o endereço da matriz é passado, não uma cópia da matriz/vetor inteiro.
- Existem três formas de se passar uma matriz ou vetor para uma função:

`void imprimir (int vet [10]);`

`void imprimir (int vet[]);`

`void imprimir (int *vet);`

# Exemplo arrays como argumentos de funções

```
#include <stdio.h>
#include <stdlib.h>
void imprimir1 (int vet[10]) {
    int i;
    for (i=0;i<10;i++) {
        printf("%d ", vet[i]);
    }
}
void imprimir2 (int vet[]) {
    int i;
    for (i=0;i<10;i++) {
        printf("%d ", vet[i]);
    }
}
void imprimir3 (int *vet) {
    int i;
    for (i=0;i<10;i++) {
        printf("%d ", vet[i]);
    }
}
```

```
int main() {
    int i, vetor[10];
    for (i=0;i<10;i++) {
        vetor[i]= rand()%10;
    }
    imprimir1(vetor);
    printf("\n");
    imprimir2(&vetor[0]);
    printf("\n");
    imprimir3(vetor);
    return 0;
}
```

# Exemplo com Vetores:

- Faça uma função que copia um vetor em outro.

```
#include <stdio.h>
#define LIM 5
void copiaVetor(int dest[], int orig[]){
    for (int i=0;i<LIM;i++){
        dest[i] = orig[i];
    }
}
void imprimeVetor(int vetor[]){
    for (int i=0;i<LIM;i++)
        printf("\n%i",vetor[i]);
}

int main(){
    int A[LIM]= {1,2,3,4,5},B[LIM];
    copiaVetor(B, A);
    imprimeVetor(A);
    imprimeVetor(B);
    return 0;
}
```

Os **tamanhos** dos vetores  
podem ser **omitidos**

# Exemplo com Vetores:

- Faça uma função que copia um vetor em outro.

```
#include <stdio.h>
#define LIM 5
void copiaVetor(int dest[], int orig[]){
    for (int i=0;i<LIM;i++)
        dest[i] = orig[i];
}
void imprimeVetor(int vetor[]){
    for (int i=0;i<LIM;i++)
        printf("\n%i",vetor[i]);
}

int main(){
    int A[LIM]= {1,2,3,4,5},B[LIM];
    copiaVetor(B, A);
    imprimeVetor(A);
    imprimeVetor(B);
    return 0;
}
```

O **nome** de um vetor sem o colchetes representa o **endereço** do vetor que é passado para a função.

# Exemplo com Vetores:

- Faça uma função que copia um vetor em outro.

```
#include <stdio.h>
#define LIM 5
void copiaVetor(int dest[], int orig[]){
    for (int i=0;i<LIM;i++)
        dest[i] = orig[i];
}
void imprimeVetor(int vetor[]){
    for (int i=0;i<LIM;i++)
        printf("\n%i",vetor[i]);
}

int main(){
    int A[LIM]= {1,2,3,4,5},B[LIM];
    copiaVetor(B, A);
    imprimeVetor(A);
    imprimeVetor(B);
    return 0;
}
```

**Alterações nos elementos dos vetores possuem efeito externo a função, pois atuam sobre o vetor original (passagem por referência).**

# Problema

- Escreva um programa em C que leia um vetor com 10 posições de números inteiros e verifique se um determinado valor, também digitado pelo usuário, está no vetor.



# Solução Monolítica

```
#include <stdio.h>
#define tam 10
int main() {
    int vetor[tam], valor, i, achou=0;
    for (i=0;i<tam;i++)
        scanf("%d", &vetor[i]);
    scanf("%d", &valor);
    for (i=0;i<tam;i++){
        if (valor==vetor[i])
            achou=1;
    }
    if (achou)
        printf("O valor %d está no vetor.", valor);
    else
        printf("O valor %d não está no vetor.", valor);
    return 0;
}
```

# Solução Modular

```
int achouVetor(int vetorA[],int n){
    for (int i=0;i<tam;i++)
        if (n==vetorA[i])
            return 1;
    return 0;
}
```

```
void leiaVetor(int vetor[]){
    printf("Leitura do vetor\n");
    for (int i=0;i<tam;i++){
        printf("Entre com vetor[%i] ",i);
        scanf("%d", &vetor[i]);
    }
}
```

```
#include <stdio.h>
#define tam 5

int main() {
    int vetor[tam], valor;
    leiaVetor(vetor);
    printf("Entre com um valor a ser pesquisado ");
    scanf("%d", &valor);
    if (achouVetor(vetor,valor))
        printf("O valor %d está no vetor.", valor);
    else
        printf("O valor %d não está no vetor.", valor);
    return 0;
}
```

# Problema

- Escreva um programa em C que leia dois vetores de 10 posições e faça a multiplicação dos elementos de mesmo índice, colocando o resultado em um terceiro vetor. Mostre o vetor resultante.

# Solução Monolítica

```
#include <stdio.h>
int main(){
    int v1[10],v2[10],v3[10];
    printf("Leitura do primeiro vetor\n");
    for (int i=0; i<10; i++) {
        printf("Entre com um valor ");
        scanf("%i",&v1[i]);
    }
    printf("Leitura do segundo vetor\n");
    for (int i=0; i<10; i++) {
        printf("Entre com um valor ");
        scanf("%i",&v2[i]);
    }
    // multiplicação dos vetores
    for (int i=0; i<10; i++)
        v3[i]=v1[i]*v2[i];
    // impressão do terceiro vetor
    printf("Terceiro vetor\n");
    for (int i=0; i<10; i++) {
        printf("v3[%i]=%i\n",i,v3[i]);
    }
    return 0;
}
```

# Solução Modular

```
#include <stdio.h>
#define LIM 5
void leituraVetor(int vetor[]){
    for (int i=0; i<LIM; i++) {
        printf("Entre com um valor ");
        scanf("%i",&vetor[i]);
    }
}
void multiplicaVetor(int mult[], int v1[], int v2[]){
    for (int i=0; i<LIM; i++)
        mult[i]=v1[i]*v2[i];
}
void imprimeVetor(int vetor[]){
    for (int i=0; i<LIM; i++)
        printf("\n%i",vetor[i]);
}
```

```
int main(){
    int v1[LIM],v2[LIM],v3[LIM];
    printf("Leitura do primeiro vetor\n");
    leituraVetor(v1);
    printf("Leitura do segundo vetor\n");
    leituraVetor(v2);
    multiplicaVetor(v3, v1, v2);
    printf("Vetor Multiplicação\n");
    imprimeVetor(v3);
    return 0;
}
```

# Exemplo com Matrizes:

- Matrizes são vetores de vetores e logo também são passadas por referência.
- A função abaixo embaralha os dados de uma matriz:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define LIM 100
void EmbaralhaMatriz(int M[][LIM], int m, int n){
    int i,j,x,y,tmp;
    srand(time(NULL));
    for(i=0; i<m; i++){
        for(j=0; j<n; j++){
            x = rand()%n; //valor de 0 a n-1.
            y = rand()%m; //valor de 0 a m-1.
            tmp = M[i][j];
            M[i][j] = M[y][x];
            M[y][x] = tmp;
        }
    }
}
```

# Exemplo com Matrizes:

- Matrizes são vetores de vetores e logo também são passadas por referência.
- A função abaixo embaralha os dados de uma matriz:

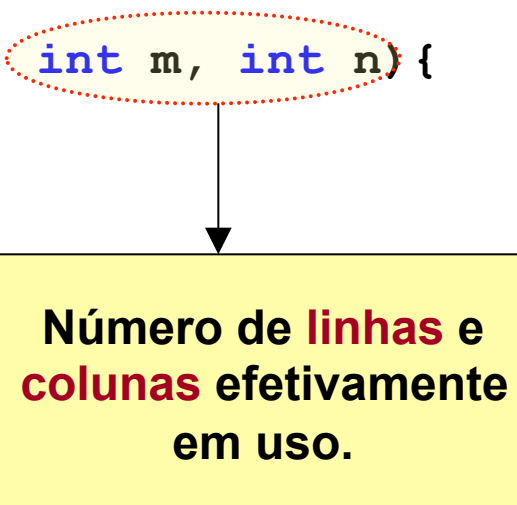
```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define LIM 100
void EmbaralhaMatriz(int M[][LIM], int m, int n){
    int i,j,x,y,tmp;
    srand(time(NULL));
    for(i=0; i<m; i++){
        for(j=0; j<n; j++){
            x = rand()%n; //valor de 0 a
            y = rand()%m; //valor de 0 a
            tmp = M[i][j];
            M[i][j] = M[y][x];
            M[y][x] = tmp;
        }
    }
}
```

Somente a primeira  
**dimensão** pode  
ser omitida.

# Exemplo com Matrizes:

- Matrizes são vetores de vetores e logo também são passadas por referência.
- A função abaixo embaralha os dados de uma matriz:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define LIM 100
void EmbaralhaMatriz(int M[][LIM], int m, int n) {
    int i, j, x, y, tmp;
    srand(time(NULL));
    for(i=0; i<m; i++){
        for(j=0; j<n; j++){
            x = rand()%n; //valor de 0
            y = rand()%m; //valor de 0
            tmp = M[i][j];
            M[i][j] = M[y][x];
            M[y][x] = tmp;
        }
    }
}
```



Número de **linhas** e **colunas** efetivamente em uso.



# Exemplo com Matrizes:

- Matrizes são vetores de vetores e logo também são passadas por referência.
- A função abaixo embaralha os dados de uma matriz:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define LIM 100
void EmbaralhaMatriz(int M[][LIM]
    int i,j,x,y,tmp;
    srand(time(NULL));
    for(i=0; i<m; i++){
        for(j=0; j<n; j++){
            x = rand()%n; //valor de 0 a n-1.
            y = rand()%m; //valor de 0 a m-1.
            tmp = M[i][j];
            M[i][j] = M[y][x];
            M[y][x] = tmp;
        }
    }
}
```

Troca elementos de cada posição **i, j** por elementos de posição aleatória **y, x**.

# Exemplo com Matrizes:

- A função geraMatriz abaixo pode ser usada para criar uma matriz:

```
void geraMatriz(int M[LIM][LIM], int m, int n){
    int i,j;
    srand(time(NULL));
    for(i=0; i<m; i++){
        for(j=0; j<n; j++){
            M[i][j] = rand()%100;
        }
    }
}
```

# Exemplo com Matrizes:

- A função `imprimeMatriz` abaixo pode ser usada para imprimir uma matriz:

```
void imprimeMatriz(int M[LIM][LIM], int m, int n){
    int i,j;
    for(i=0; i<m; i++){
        for(j=0; j<n; j++){
            printf("%02d ",M[i][j]);
        }
        printf("\n");
    }
    printf("\n");
}
```

# Exemplo com Matrizes:

- A função principal abaixo pode ser usada para testar a função que embaralha os dados de uma matriz:

```
int main(){  
    int M[LIM][LIM];  
    geraMatriz(M,4,4);  
    imprimeMatriz(M,4,4);  
    EmbaralhaMatriz(M,4,4);  
    imprimeMatriz(M,4,4);  
    return 0;  
}
```

# Problema

- Faça um programa que encontre a transposta da matriz.
- Faça um programa que encontre o maior e o menor elemento de uma matriz.

# Protótipo de funções

- **Definição:**
  - Protótipos de funções são declarações de funções, sem que haja uma implementação.
- **Sintaxe:**

```
tipo nome_da_função(lista de argumentos) ;
```

# Exemplo de protótipo de funções

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

```
float quadrado (float a);
```

```
int main () {  
    float num;  
    printf ("Digite um número: ");  
    scanf ("%f",&num);  
    num = quadrado(num);  
    printf ("O seu quadrado vale: %f\n",num);  
    return 0;  
}
```

```
float quadrado (float a) {  
    return (a*a);  
}
```