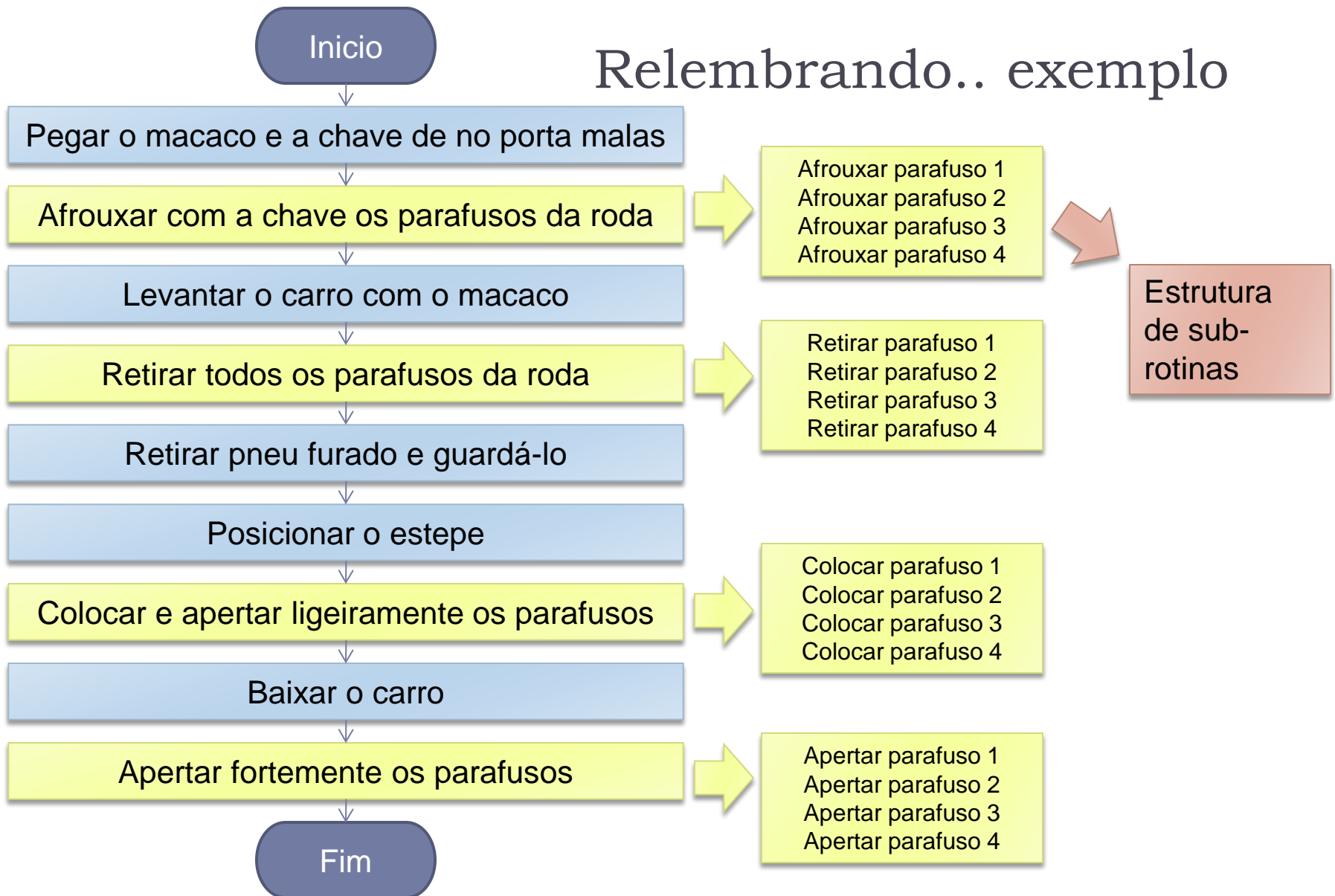


Funções

Prof. Bruno Travençolo
Baseado em slides do Prof. André Backes

Relembrando.. exemplo



Subprogramas

- ▶ Subprograma é um programa que auxilia o programa principal através da realização de uma determinada subtarefa.
- ▶ Também costuma receber os nomes de *sub-rotina*, *procedimento*, *função*, *método* ou *módulo*.
- ▶ Os subprogramas são chamados dentro do corpo do programa principal como se fossem *comandos*. *Após seu término, a execução continua a partir do ponto onde foi chamado.*
- ▶ É importante compreender que a chamada de um subprograma simplesmente gera um **desvio provisório no fluxo de execução**.



Função

- ▶ Funções são blocos de código que podem ser nomeados e chamados de dentro de um programa.
 - ▶ **printf()**: função que escreve na tela
 - ▶ **scanf()**: função que lê o teclado



Função

- ▶ **Facilitam a estruturação e reutilização do código.**
 - ▶ Estruturação: programas grandes e complexos são construídos bloco a bloco.
 - ▶ Reutilização: o uso de funções evita a cópia desnecessária de trechos de código que realizam a mesma tarefa, diminuindo assim o tamanho do programa e a ocorrência de erros



Função - Estrutura

- ▶ Forma geral de uma função:

```
tipo_retornado nome_função(parâmetros){  
    conjunto de declarações e comandos  
}
```



Função - Parâmetros

- ▶ A declaração de parâmetros é uma lista de variáveis juntamente com seus tipos:
 - ▶ tipo nome1, tipo nome2, ... , tipoN nomeN

```
01 //Declaração CORRETA de parâmetros
02 int soma(int x, int y){
03     return x + y;
04 }
05
06 //Declaração ERRADA de parâmetros
07 int soma(int x, y){
08     return x + y;
09 }
```

- ▶ Pode-se colocar **void** entre os parênteses se a função não recebe nenhum parâmetro de entrada



Função - Corpo

- ▶ O corpo da função é a sua alma.
 - ▶ É formado pelas “declarações” e “comandos” que a função executa.
 - ▶ Processa os parâmetros, realiza outras tarefas e gera saídas se necessário.
 - ▶ Similar a cláusula **main()**

```
int main(){  
    conjunto de declarações e comandos  
    return 0;  
}
```



Função - Corpo

- ▶ De modo geral, evita-se fazer operações de leitura e escrita dentro de uma função.
- ▶ Uma função é construída com o intuito de realizar uma tarefa específica e bem-definida.
- ▶ As operações de entrada e saída de dados (funções **scanf()** e **printf()**) devem ser feitas em quem chamou a função (por exemplo, na **main()**).
- ▶ Isso assegura que a função construída possa ser utilizada nas mais diversas aplicações, garantindo a sua generalidade.



Função - Retorno

- ▶ Uma função pode retornar qualquer valor válido em C
 - ▶ tipos pré-definidos (int, char, float e double);
 - ▶ tipos definidos pelo usuário (struct).
- ▶ Uma função que retorna nada é definida colocando-se o tipo **void** como valor retornado



Comando return

- ▶ O valor retornado pela função é dado pelo comando **return**. Forma geral:

return *valor ou expressão*;

Ou

return;

- ▶ É importante lembrar que o valor de retorno fornecido tem que ser compatível com o tipo de retorno declarado para a função.



Comando return

- ▶ Uma função pode ter mais de uma declaração **return**.

```
01  int maior(int x, int y){  
02      if(x > y)  
03          return x;  
04      else  
05          return y;  
06  }
```

- ▶ Quando o comando **return** é executado, a função termina imediatamente. Todos os comandos restantes são ignorados.



Chamada de Função

- ▶ Para usar uma função, devemos chamá-la dentro da função principal (main) ou dentro de outra função
- ▶ Para chamar a função, basta escrever seu nome e colocar os parâmetros necessários
- ▶ Se a função retorna algum valor, pode-se copiar este valor para um variável ou usá-lo em alguma expressão



Exemplos - Função sem retorno e sem parâmetros

```
// função sem retorno e sem parâmetros de entrada
void MensagemBoasVindas() {
    printf("\n");
    printf("=====\n");
    printf("  Seja Bem-Vindo  \n");
    printf("=====\n");
    printf("\n");
}
```

```
int main()
{
    // Chamando a função
    MensagemBoasVindas();
    . . . .
}
```



Exemplos - Função sem retorno e sem parâmetros

Sem parâmetros.
Coloque parênteses na
frente do nome da função



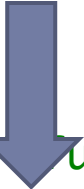
```
// função sem retorno e sem parâmetros de entrada
void MensagemBoasVindas() {
    printf("\n");
    printf("=====\n");
    printf("  Seja Bem-Vindo  \n");
    printf("=====\n");
    printf("\n");
}
```

```
int main()
{
    // Chamando a função
    MensagemBoasVindas();
    ....
}
```



Exemplos - Função sem retorno e sem parâmetros

Sem retorno. Escreva **void** antes do nome da função



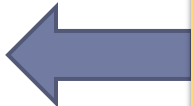
```
// função sem retorno e sem parâmetros de entrada
void MensagemBoasVindas() {
    printf("\n");
    printf("=====\n");
    printf("  Seja Bem-Vindo  \n");
    printf("=====\n");
    printf("\n");
}
```

```
int main()
{
    // Chamando a função
    MensagemBoasVindas();
    ....
}
```


Exemplos - Função sem retorno e sem parâmetros

```
// função sem retorno e sem parâmetros de entrada
void MensagemBoasVindas() {
    printf("\n");
    printf("=====\n");
    printf("  Seja Bem-Vindo  \n");
    printf("=====\n");
    printf("\n");
}
```

```
int main()
{
    // Chamando a função
    MensagemBoasVindas();
    ....
}
```



Chamada da função. Basta escrever seu nome. Coloque parênteses na frente do nome da função

Exemplos - Função sem retorno e sem parâmetros

```
// função sem retorno e sem parâmetros de entrada
void MensagemBoasVindas() {
    printf("\n");
    printf("=====\n");
    printf("  Seja Bem-Vindo  \n");
    printf("=====\n");
    printf("\n");
}
```

Função sem
parâmetros

```
int main()
{
    // Chamando a função
    MensagemBoasVindas();
    ....
}
```



Exemplos - Função sem retorno e com parâmetros

```
// função sem parâmetros de entrada, mas com retorno
char MenuPrincipal(){


    char op;
    printf("Escolha uma opção: \n\n");
    printf("1 - Novo Jogo\n");
    printf("2 - Carregar Jogo\n");
    printf("3 - Sair\n");

    setbuf(stdin, NULL);
    scanf("%c", &op);

    return op;
}

int main()
{
    char escolha;
    MensagemBoasVindas();

    escolha = MenuPrincipal();
```



Exemplos - parâmetros

Sem parâmetros.
Coloque parênteses na
frente do nome da função

retorno e com

// função sem parâmetros de entrada, mas com retorno
`char` MenuPrincipal(){

```
    char op;  
    printf("Escolha uma opção: \n\n");  
    printf("1 - Novo Jogo\n");  
    printf("2 - Carregar Jogo\n");  
    printf("3 - Sair\n");
```

```
    setbuf(stdin, NULL);  
    scanf("%c", &op);
```

```
    return op;  
}
```

```
int main()  
{  
    char escolha;  
    MensagemBoasVindas();
```

```
    escolha = MenuPrincipal();
```

Com retorno. Coloque o tipo de dado que será retornado. Neste exemplo, é um char

o sem retorno e com

// função sem parâmetros de entrada, mas com retorno
char MenuPrincipal(){

```
    char op;  
    printf("Escolha uma opção: \n\n");  
    printf("1 - Novo Jogo\n");  
    printf("2 - Carregar Jogo\n");  
    printf("3 - Sair\n");
```

```
    setbuf(stdin, NULL);  
    scanf("%c", &op);
```

```
    return op;
```

```
}
```

```
int main()
```

```
{
```

```
    char escolha;  
    MensagemBoasVindas();
```

```
    escolha = MenuPrincipal();
```

Exemplos - Função sem retorno e com parâmetros

```
// função sem parâmetros de entrada, mas com retorno  
char MenuPrincipal(){
```

```
    char op;  
    printf("Escolha uma opção: \n\n");  
    printf("1 - Novo Jogo\n");  
    printf("2 - Carregar Jogo\n");  
    printf("3 - Sair\n");
```

```
    setbuf(stdin, NULL);  
    scanf("%c", &op);
```


```
    return op;
```

```
}
```

```
int main()  
{
```

```
    char escolha;  
    MensagemBoasVindas();
```

```
    escolha = MenuPrincipal();
```



Como há retorno, devemos usar o comando 'return' para indicar o retornar e finalizar a função

Exemplos - Função sem retorno e com parâmetros

```
// função sem parâmetros de entrada, mas com retorno
```

```
char MenuPrincipal(){  
    char op;  
    printf("Escolha uma opção: \n\n");  
    printf("1 - Novo Jogo\n");  
    printf("2 - Carregar Jogo\n");  
    printf("3 - Sair\n");
```

```
    setbuf(stdin, NULL);  
    scanf("%c", &op);
```

```
    return op;  
}
```

Observe que o tipo retornado deve ser do mesmo especificado no cabeçalho da função

```
int main()  
{  
    char escolha;  
    MensagemBoasVindas();
```

```
    escolha = MenuPrincipal();
```

Exemplos - Função sem retorno e com parâmetros

```
// função sem parâmetros de entrada, mas com retorno  
char MenuPrincipal(){
```

```
    char op;  
    printf("Escolha uma opção: \n\n");  
    printf("1 - Novo Jogo\n");  
    printf("2 - Carregar Jogo\n");  
    printf("3 - Sair\n");
```

```
    setbuf(stdin, NULL);  
    scanf("%c", &op);
```


```
    return op;
```

```
}
```

```
int main()  
{
```

```
    char escolha;  
    MensagemBoasVindas();
```

```
    escolha = MenuPrincipal();
```



Chamando a
função

Exemplos - Função sem retorno e com parâmetros

```
// função sem parâmetros de entrada, mas com retorno  
char MenuPrincipal(){
```

```
    char op;  
    printf("Escolha uma opção: \n\n");  
    printf("1 - Novo Jogo\n");  
    printf("2 - Carregar Jogo\n");  
    printf("3 - Sair\n");
```

```
    setbuf(stdin, NULL);  
    scanf("%c", &op);
```

```
    return op;
```

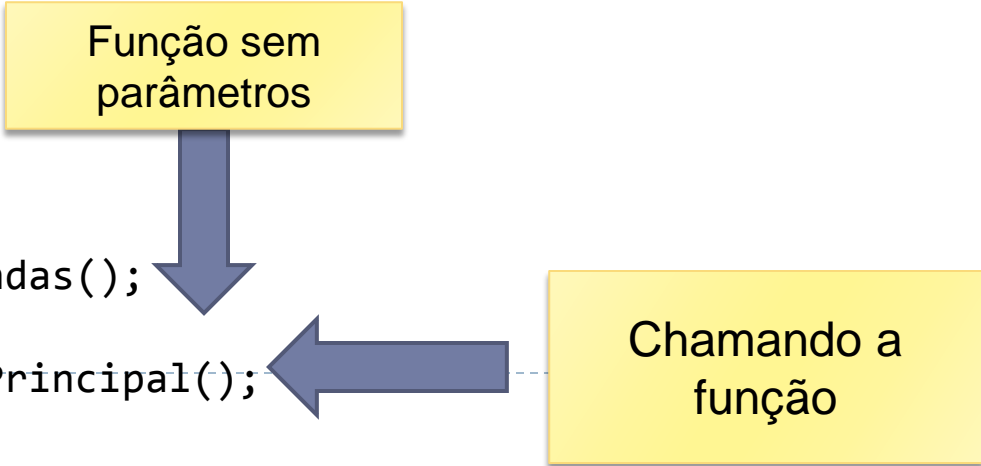
```
}
```

```
int main()  
{
```

```
    char escolha;  
    MensagemBoasVindas();
```

```
    escolha = MenuPrincipal();
```

Função sem
parâmetros



Chamando a
função

Exemplos - Função sem retorno e com parâmetros

```
// função sem parâmetros de entrada, mas com retorno  
char MenuPrincipal(){
```

```
    char op;  
    printf("Escolha uma opção: \n\n");  
    printf("1 - Novo Jogo\n");  
    printf("2 - Carregar Jogo\n");  
    printf("3 - Sair\n");
```

```
    setbuf(stdin, NULL);
```

A função retorna um valor. Esse valor deve ser armazenado em algum local, do mesmo tipo do retorno

```
int main()
```

```
{
```

```
    char escolha;
```

```
    MensagemBoasVindas();
```

```
    escolha = MenuPrincipal();
```

Chamando a
função

Exemplos - Função sem retorno e com parâmetros

```
// função sem parâmetros de entrada, mas com retorno  
char MenuPrincipal(){
```

```
    char op;  
    printf("Escolha uma opção: \n\n");  
    printf("1 - Novo Jogo\n");  
    printf("2 - Carregar Jogo\n");  
    printf("3 - Sair\n");
```

```
    setbuf(stdin, NULL);  
    scanf("%c", &op);
```

```
    return op;
```

```
}
```

```
int main()
```

```
{ ...
```

```
    while (escolha = MenuPrincipal() != '3') {
```

```
        switch(escolha){
```

```
            //...
```

```
        };
```

Outro exemplo de
chamada da
função



Exemplos - Função sem retorno e com parâmetros

```
// função sem parâmetros de entrada, mas com retorno  
char MenuPrincipal(){
```

```
    char op;  
    printf("Escolha uma opção: \n\n");  
    printf("1 - Novo Jogo\n");  
    printf("2 - Carregar Jogo\n");  
    printf("3 - Sair\n");
```

Armazenado o
retorno

```
        scanf(stdin, NULL);  
        f("%c",&
```

Chamando a
função

```
        return op;  
    }  
  
int main()  
{  
    while (escolha = MenuPrincipal() != '3') {  
        switch(escolha){  
            //...  
        }  
    }  
};
```

Exemplos - Função com retorno e com parâmetros


```
// função com retorno e com parâmetros
int VerificaAprovacao(double nota, int faltas) {
    int aprovado = 1;
    if ( (faltas > 18) || (nota < 60.0) ) {
        aprovado = 0;
    }

    return aprovado;
}
```



Exemplos - parâmetros

Com parâmetros. Coloque parênteses na frente do nome da função e, dentro dos parênteses os parâmetros. Neste caso temos dois parâmetros: um double e um int




```
// função com retorno e 2 parâmetros
int VerificaAprovacao(double nota, int faltas) {
    int aprovado = 1;
    if ( (faltas > 18) || (nota < 60.0) ) {
        aprovado = 0;
    }

    return aprovado;
}
```



Com retorno. Coloque o tipo de dado que será retornado. Neste exemplo, é um int

o com retorno e com




```
// função com retorno e com parâmetros
int VerificaAprovacao(double nota, int faltas) {
    int aprovado = 1;
    if ( (faltas > 18) || (nota < 60.0) ) {
        aprovado = 0;
    }

    return aprovado;
}
```


Com retorno. Coloque o tipo de dado que será retornado. Neste exemplo, é um int

o com retorno e com



```
// função com retorno e com parâmetros
int VerificaAprovacao(double nota, int faltas) {
    int aprovado = 1;
    if ( (faltas > 18) || (nota < 60.0) ) {
        aprovado = 0;
    }

    return aprovado;
}
```



Observe que o tipo retornado deve ser do mesmo especificado no cabeçalho da função

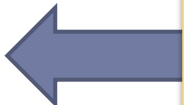
Exemplos - Função com retorno e com parâmetros

```
// função com retorno e com parâmetros
int VerificaAprovacao(double nota, int faltas) {
    int aprovado = 1;
    if ( (faltas > 18) || (nota < 60.0) ) {
        aprovado = 0;
    }

    return aprovado;
}
```

```
int main()
{
    int ap;

    ap = VerificaAprovacao(4.0, 5);
    if (ap) {
        printf("Aprovado!!!");
    }
}
```



Chamando a
função

Exemplos - Função com retorno e com parâmetros


```
// função com retorno e com parâmetros
int VerificaAprovacao(double nota, int faltas) {
    int aprovado = 1;
    if ( (faltas > 18) || (nota < 60.0) ) {
        aprovado = 0;
    }

    return aprovado;
}
```

```
int main()
{
    int ap;

    ap = VerificaAprovacao(4.0, 5);
    if (ap) {
        printf("Aprovado!!!");
    }
}
```

Função COM
parâmetros



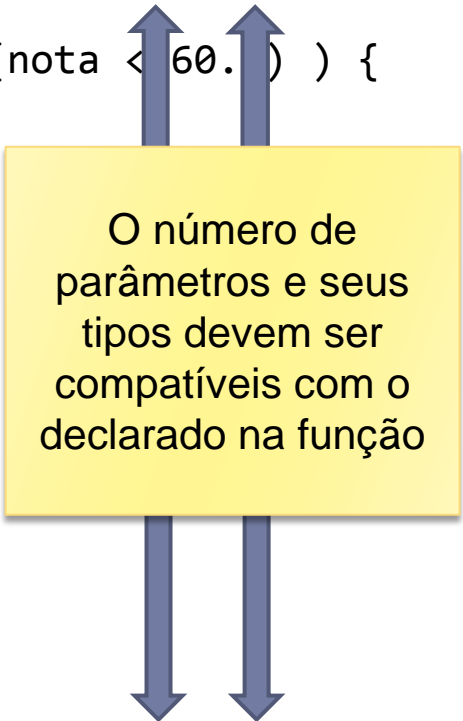
Exemplos - Função com retorno e com parâmetros

```
// função com retorno e com parâmetros
int VerificaAprovacao(double nota, int faltas) {
    int aprovado = 1;
    if ( (faltas > 18) || (nota < 60.0) ) {
        aprovado = 0;
    }

    return aprovado;
}

int main()
{
    int ap;

    ap = VerificaAprovacao(4.0, 5);
    if (ap) {
        printf("Aprovado!!!");
    }
}
```



O número de parâmetros e seus tipos devem ser compatíveis com o declarado na função

The diagram illustrates the flow of data between the `VerificaAprovacao` function and the `main` function. Two blue arrows point upwards from the function call `VerificaAprovacao(4.0, 5)` in `main` to the parameters `double nota` and `int faltas` in the function signature. Two blue arrows point downwards from the `return` statement in the function to the variable `ap` in `main`. A yellow box with black text is positioned between the function and the caller, stating: "O número de parâmetros e seus tipos devem ser compatíveis com o declarado na função".

Exemplos - Função com retorno e com parâmetros

```
// função com retorno e com parâmetros
int VerificaAprovacao(double nota, int faltas) {
    int aprovado = 1;
    if ( (faltas > 18) || (nota < 60.0) ) {
        aprovado = 0;
    }

    return aprovado;
}
```

Função com
retorno

```
main()
{
    int ap;

    ap = VerificaAprovacao(4.0, 5);
    if (ap != 0) {
        printf("Aprovado!!!");
    }
}
```

Exemplos - Função com retorno e com parâmetros

```
// função com retorno e com parâmetros
int VerificaAprovacao(double nota, int faltas) {
    int aprovado = 1;
    if ( (faltas > 18) || (nota < 60.0) ) {
        aprovado = 0;
    }

    return aprovado;
}
```

Outro exemplo de chamada

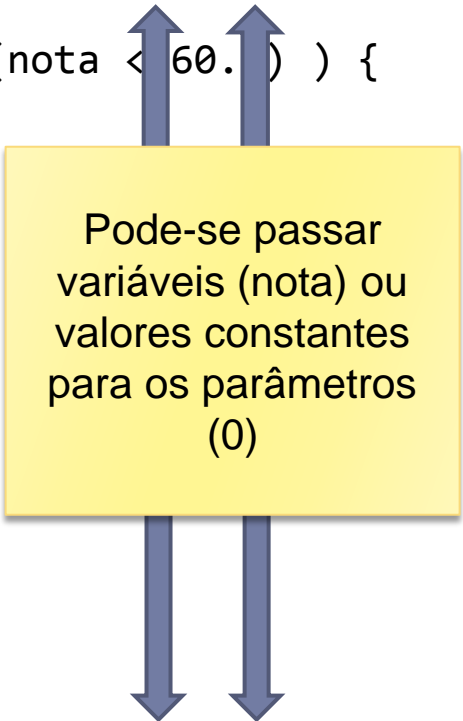
```
int main()
{
    int ap;
    double nota = 70.1;
    ap = VerificaAprovacao(nota, 0);
    if (ap) {
        printf("Aprovado!!!");
    }
}
```

Exemplos - Função com retorno e com parâmetros

```
// função com retorno e com parâmetros
int VerificaAprovacao(double nota, int faltas) {
    int aprovado = 1;
    if ( (faltas > 18) || (nota < 60.) ) {
        aprovado = 0;
    }

    return aprovado;
}

int main()
{
    int ap;
    double nota = 70.1;
    ap = VerificaAprovacao(nota, 0);
    if (ap) {
        printf("Aprovado!!!");
    }
}
```



Pode-se passar variáveis (nota) ou valores constantes para os parâmetros (0)

The diagram illustrates the flow of data between the `main` function and the `VerificaAprovacao` function. Two blue arrows point upwards from the `main` function's arguments (`nota` and `0`) to the corresponding parameters (`double nota` and `int faltas`) in the `VerificaAprovacao` function signature. Two blue arrows point downwards from the `VerificaAprovacao` function's `return` statement to the `ap` variable in the `main` function, indicating the return of the `aprovado` value.

Exemplos - Função com retorno e com parâmetros

```
// função com retorno e com parâmetros
int VerificaAprovacao(double nota, int faltas) {
    int aprovado = 1;
    if ( (faltas > 18) || (nota < 60.0) ) {
        aprovado = 0;
    }

    return aprovado;
}
```

Outro exemplo de chamada

```
int main()
{
    int ap; double nota = 40;
    int faltas = 4;
    ap = VerificaAprovacao(nota, faltas);
    if (ap) {
        printf("Aprovado!!!");
    }
}
```

Exemplos - Função com retorno e com parâmetros

```
// função com retorno e com parâmetros
int VerificaAprovacao(double nota, int faltas) {
    int aprovado = 1;
    if ( (faltas > 18) || (nota < 60.0) ) {
        aprovado = 0;
    }

    return aprovado;
}

int main()
{
    .....
    if (VerificaAprovacao(nota, faltas) != 0) {
        printf("Aprovado!!!");
    }
}
```

Outro exemplo de chamada



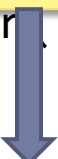
Exemplos - Função com retorno e com parâmetros

```
// função com retorno e com parâmetros
int VerificaAprovacao(double nota, int faltas) {
    int aprovado = 1;
    if ( (faltas > 18) || (nota < 60.0) ) {
        aprovado = 0;
    }

    return aprovado;
}
```

Observe que não foi usada nenhuma variável para receber o tipo de retorno. O valor retornado é reconhecido pelo comando *IF*

```
int main()
{
    ....
    if (VerificaAprovacao(nota, faltas) != 0) {
        printf("Aprovado!!!");
    }
}
```



Exemplos - Função com retorno e com parâmetros

```
// função com retorno e com parâmetros
int VerificaAprovacao(double nota, int faltas) {
    int aprovado = 1;
    if ( (faltas > 18) || (nota < 60.0) ) {
        aprovado = 0;
    }

    return aprovado;
}
```

```
int main()
{
```

```
    VerificaAprovacao(70.0, 0);
```

Outro exemplo de
chamada



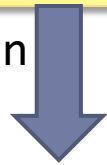
Exemplos - Função com retorno e com parâmetros

```
// função com retorno e com parâmetros
int VerificaAprovacao(double nota, int faltas) {
    int aprovado = 1;
    if ( (faltas > 18) || (nota < 60.0) ) {
        aprovado = 0;
    }
}
```

```
return aprovado;
```

Observe que não foi usada nenhuma variável para receber o tipo de retorno. O valor retornado é perdido (mas o código funciona)

```
int main
{
```



```
VerificaAprovacao(70.0, 0);
```

Declaração de Funções

- ▶ Funções devem ser definidas ou declaradas antes de serem utilizadas, ou seja, antes da cláusula main.
- ▶ A definição (protótipo) apenas indica a existência da função:

tipo_retornado nome_função(parâmetros);

- ▶ Desse modo ela pode ser escrita após a cláusula main().

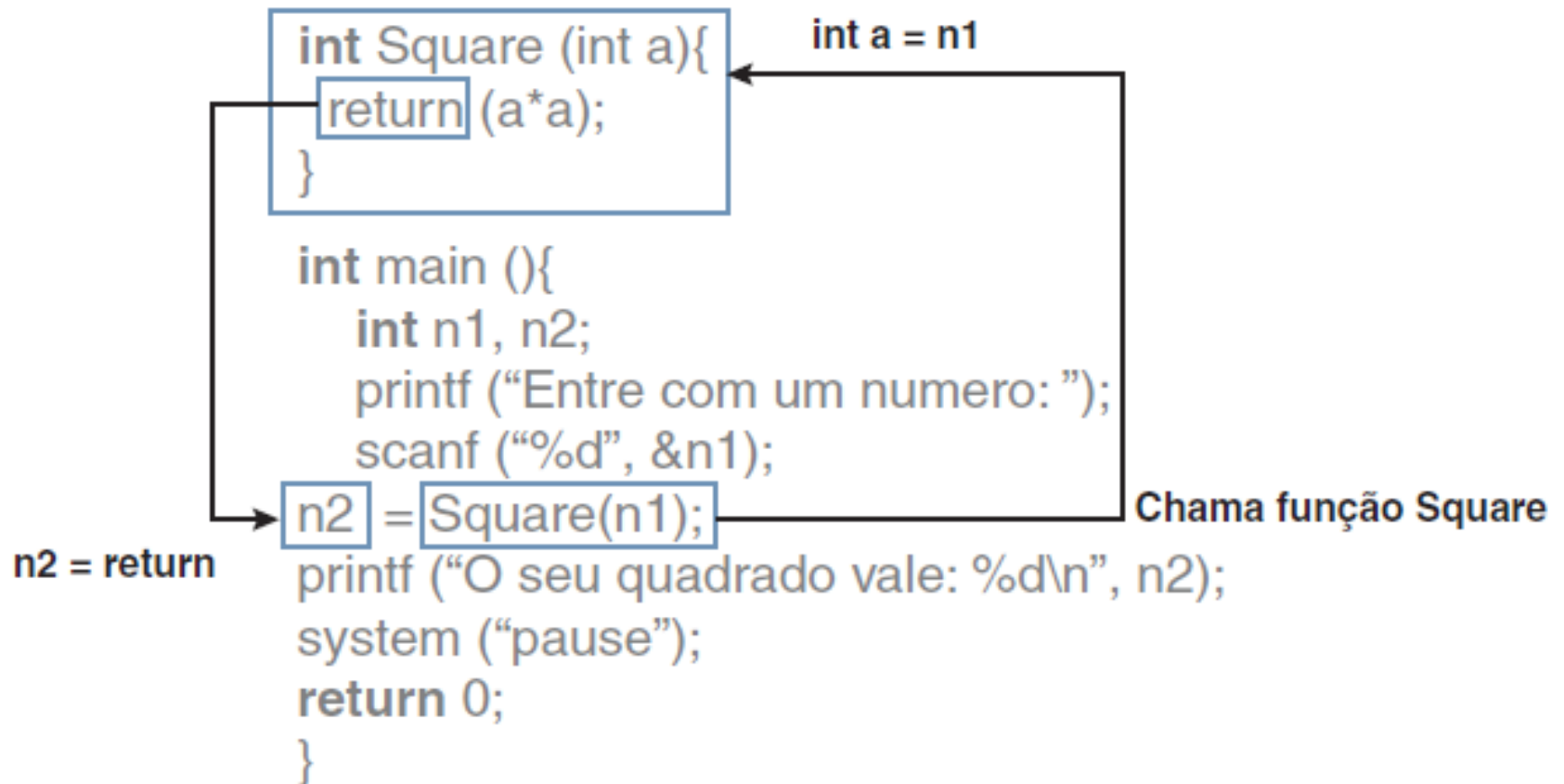


Exemplo

```
01  #include <stdio.h>
02  #include <stdlib.h>
03
04  int Square (int a){
05      return (a*a);
06  }
07
08  int main(){
09      int n1,n2;
10      printf("Entre com um numero: ");
11      scanf("%d", &n1);
12      n2 = Square(n1);
13      printf("O seu quadrado vale: %d\n", n2);
14      system("pause");
15      return 0;
16  }
```



Exemplo




Exemplo – usando protótipos

```
01  #include <stdio.h>
02  #include <stdlib.h>
03  //protótipo da função
04  int Square (int a);
05
06  int main(){
07      int n1,n2;
08      printf("Entre com um numero: ");
09      scanf("%d", &n1);
10      n2 = Square(n1);
11      printf("O seu quadrado vale: %d\n", n2);
12      system("pause");
13      return 0;
14  }
15
16  int Square (int a){
17      return (a*a);
18  }
```



protótipo



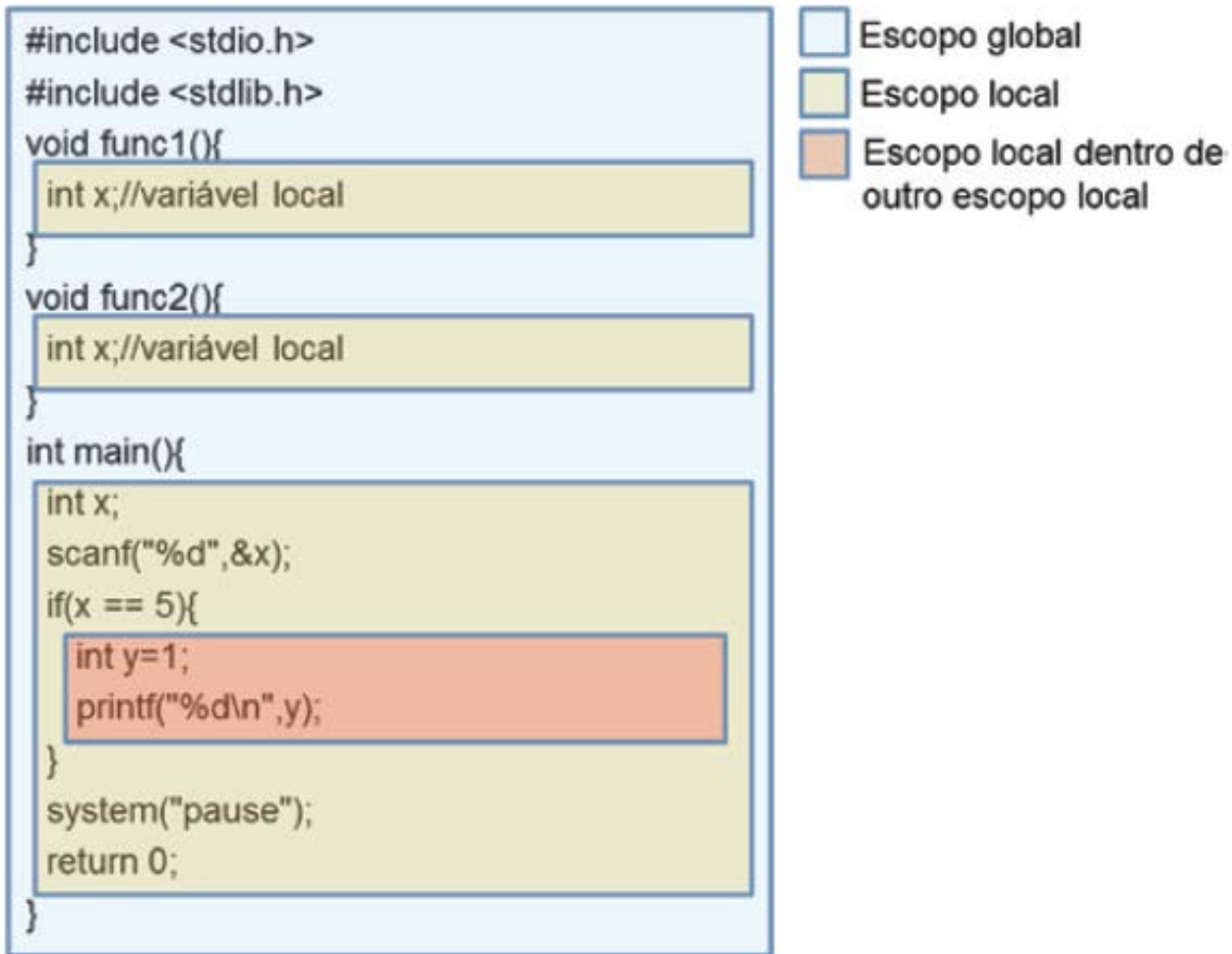
função

Escopo de variáveis

- ▶ **Escopo**
 - ▶ Define onde e quando a variável pode ser usada.
- ▶ **Escopo global**
 - ▶ Fora de qualquer definição de função
 - ▶ Tempo de vida é o tempo de execução do programa
- ▶ **Escopo local**
 - ▶ Bloco ou função



Escopo de variáveis



Escopo

- ▶ Funções também estão sujeitas ao escopo das variáveis
- ▶ O escopo é o conjunto de regras que determinam o uso e a validade de variáveis nas diversas partes do programa.
 - ▶ Variáveis Locais, variáveis Globais e Parâmetros formais.



Escopo

- ▶ Variáveis locais são aquelas que só têm validade dentro do bloco no qual são declaradas.
 - ▶ Um bloco começa quando abrimos uma chave e termina quando fechamos a chave.
 - ▶ Ex.: variáveis declaradas dentro da função.



Escopo

- ▶ Parâmetros formais são declarados como sendo as entradas de uma função.
 - ▶ O parâmetro formal é uma variável local da função.
 - ▶ Ex.: `float square(float x);` // x é um parâmetro formal



Escopo

- ▶ 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.

Evite usar variáveis globais



Escopo de variáveis

- Bloco: visível apenas no interior de um bloco de comandos

```
int main()
{
    int i;
    i = 2;
    printf("Valor de i (antes do bloco): %d\n", i);

    // abertura de um bloco
    {
        int i;
        i = 3;
        printf("Valor de i (dentro do bloco): %d\n", i);
    }

    printf("Valor de i (depois do bloco): %d\n", i);

    return 0;
}
```

Escopo de variáveis


- Bloco: visível apenas no interior de um bloco de comandos

```
int main()
{
    int i;
    i = 2;
    printf("Valor de i (antes do bloco): %d\n", i);

    // abertura de um bloco
    {
        int i;
        i = 3;
        printf("Valor de i (dentro do bloco): %d\n", i);
    }

    printf("Valor de i (depois do bloco): %d\n", i);

    return 0;
}
```



Observe que foi declarada uma variável de mesmo nome de uma pré-declarada

Escopo de variáveis

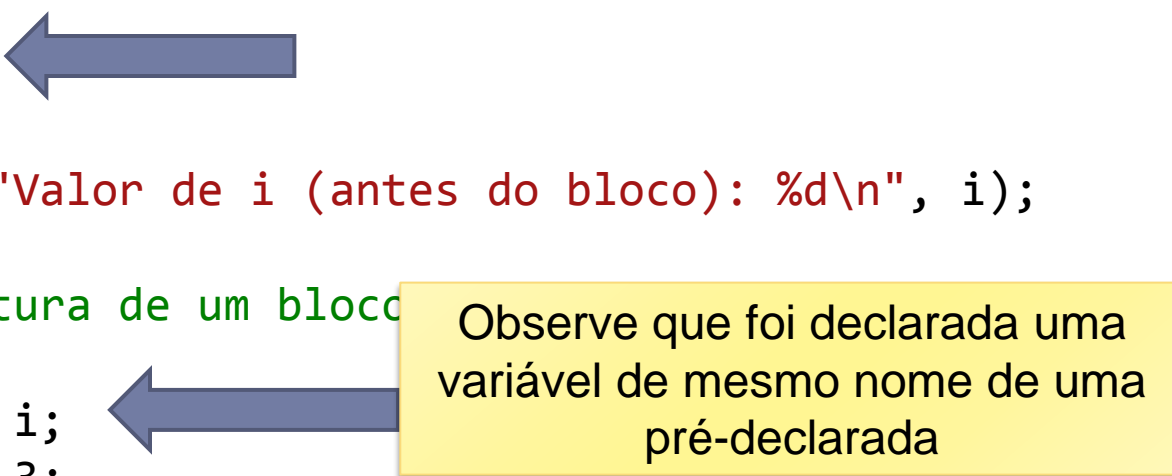
- Bloco: visível apenas no interior de um bloco de comandos

```
int main()
{
    int i;
    i = 2;
    printf("Valor de i (antes do bloco): %d\n", i);

    // abertura de um bloco
    {
        int i;
        i = 3;
        printf("Valor de i (dentro do bloco): %d\n", i);
    }

    printf("Valor de i (depois do bloco): %d\n", i);

    return 0;
}
```



Observe que foi declarada uma variável de mesmo nome de uma pré-declarada

Escopo de variáveis

- Bloco: visível apenas no interior de um bloco de comandos

```
int main()
{
    int i;
    i = 2;
    printf("Valor de i (antes do bloco): %d\n", i);

    // abertura de um bloco
    {
        int i;
        i = 3;
        printf("Valor de i (dentro do bloco): %d\n", i);
    }

    printf("Valor de i (depois do bloco): %d\n", i);

    return 0;
}
```

Esta variável só existe dentro deste bloco

Escopo de variáveis

- Bloco: visível apenas no interior de

```
int main()
{
    int i;
    i = 2;
    printf("Valor de i (antes do bloco): %d\n", i);

    // abertura de um bloco
    {
        int i;
        i = 3;
        printf("Valor de i (dentro do bloco): %d\n", i);
    }

    printf("Valor de i (depois do bloco): %d\n", i);

    return 0;
}
```

Saída

Valor de i (antes do bloco): 2
Valor de i (dentro do bloco): 3
Valor de i (depois do bloco): 2

Escopo Mapa de memória

Endereço	Blocos (1 byte)	Nome variável	Tipo
0 / NULL	indefinido	----	----
1	2	i	int
2			
3			
4			
5			
6			
7			
8			
9			
10			

```
int main()  
{
```

```
    int i;  
    i = 2;
```

```
    printf("Valor de i (antes do bloco): %d\n", i);
```



```
    // abertura de um bloco
```

```
{
```

```
    int i;
```

```
    i = 3;
```

```
    printf("Valor de i (dentro do bloco): %d\n", i);
```

```
}
```

```
    printf("Valor de i (depois do bloco): %d\n", i);
```

```
    return 0;
```



```
}
```

Escopo Mapa de memória


Endereço	Blocos (1 byte)	Nome variável	Tipo
0 / NULL	indefinido	----	----
1	2	i	int
2			
3			
4			
5	lx	i	int
6			
7			
8			
9			
10			

```
int main()
{
    int i;
    i = 2;
    printf("Valor de i (antes do bloco): %d\n", i);

    // abertura de um bloco
    {
        int i;
        i = 3;
        printf("Valor de i (dentro do bloco): %d\n", i);
    }

    printf("Valor de i (depois do bloco): %d\n", i);

    return 0;
}
```



Escopo Mapa de memória

Variáveis de mesmo nome,
mas escopo diferente

```
int main()
{
    int i;
    i = 2;
    printf("Valor de i (antes do bloco): %d\n", i);

    // abertura de um bloco
    {
        int i;
        i = 3;
        printf("Valor de i (dentro do bloco): %d\n", i);
    }

    printf("Valor de i (depois do bloco): %d\n", i);

    return 0;
}
```

Endereço	Blocos (1 byte)	Nome variável	Tipo
0 / NULL	indefinido	----	----
1	2	i	int
2			
3			
4			
5	lx	i	int
6			
7			
8			
9			
10			

Escopo Mapa de memória

```
int main()
{
    int i;
    i = 2;
    printf("Valor de i (antes do bloco): %d\n", i);

    // abertura de um bloco
    {
        int i;
        i = 3;
        printf("Valor de i (dentro do bloco): %d\n", i);
    }

    printf("Valor de i (depois do bloco): %d\n", i);

    return 0;
}
```

Espaço de memória é
válido somente durante
a execução do bloco

Endereço	Blocos (1 byte)	Nome variável	Tipo
0 / NULL	indefinido	----	----
1	2	i	int
2			
3			
4			
5	3	i	int
6			
7			
8			
9			
10			

Escopo Mapa de memória

Endereço	Blocos (1 byte)	Nome variável	Tipo
0 / NULL	indefinido	----	----
1	2	i	int
2			
3			
4			
5			
6			
7			
8			
9			
10			

```
int main()
{
    int i;
    i = 2;
    printf("Valor de i (antes do bloco): %d\n", i);

    // abertura de um bloco
    {
        int i;
        i = 3;
        printf("Valor de i (dentro do bloco): %d\n", i);
    }

    printf("Valor de i (depois do bloco): %d\n", i);

    return 0;
}
```

Endereços reais

Verificando os enderecos

Endereco de i (antes do bloco): 2686748
Endereco de i (dentro do bloco): 2686744
Endereco de i (depois do bloco): 2686748

```
int main()
{
    // verificando endereços das
    printf("\n\n Verificando os enderecos \n\n");
    int i;
    i = 2;
    printf("Endereco de i (antes do bloco):  %u\n", &i);

    // abertura de um bloco
    {
        int i;
        i = 3;
        printf("Endereco de i (dentro do bloco): %u\n", &i);

    }

    printf("Endereco de i (depois do bloco): %u\n", &i);

    return 0;
```



}

Escopo de variáveis

▶ Escopo local

- ▶ **Bloco:** visível apenas no interior de um bloco de comandos

```
if (teste == TRUE) {  
    int i;  
    i = i+1;  
}
```



Escopo de variáveis

▶ Escopo local em Funções

- ▶ Função: declarada na lista de parâmetros da função ou definida dentro da função

```
int minha_fun (int x, int y) {  
    int i, j;  
}
```

- ▶ Variáveis x, y, i e j são locais a função e não são acessíveis a nenhuma outra função ou ao programa principal



Exemplo

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

```
int maior(int a, int b){
    if ( a > b)
        return a;
    else
        return b;
}
```

Mapa de Memória?

```
int main()
{
    int a,b,c,d,m;

    a = 1;
    b = 2;
    m = maior(a,b);
    printf("\n0 maior valor entre (%d,%d) eh %d", a,b,m);

    c = -1;
    d = -50;
    printf("\n0 maior valor entre (%d,%d) eh %d", c,d,maior(c,d));

    return 0;
}
```



Exemplo

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

```
int maior(int a, int b){
    if ( a > b)
        return a;
    else
        return b;
}
```

```
int main()
{
    int a,b,c,d,m;
    a = 1;
    b = 2;
    m = maior(a,b);
    printf("\n0 maior valor entre (%d,%d) eh %d", a,b,m);

    c = -1;
    d = -50;
    printf("\n0 maior valor entre (%d,%d) eh %d", c,d,maior(c,d));

    return 0;
}
```

Endereço	Blocos (1 byte)	Nome variável	Tipo	Endereço	Blocos (1 byte)	Nome variável	Tipo
0 / NULL	indefinido	----	----	47			int
1	lx	a	int	48			
2				49			
3				50			
4				51			
5	lx	b	int	52			
6				53			
7				54			
8				55			
9	lx	c	int	56			
10				57			
11				58			
12				59			
13	lx	d	int	60			
14				61			
15				62			
16				63			
17	lx	m		64			
18				65			
19				66			
20				67			
21				68			
22				69			
23				70			
24				71			

Exemplo

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

```
int maior(int a, int b){
    if ( a > b)
        return a;
    else
        return b;
}
```

```
int main()
{
    int a,b,c,d,m;
```

```
    a = 1;
    b = 2;
    m = maior(a,b);
    printf("\n0 maior valor entre (%d,%d) eh %d", a,b,m);
```

```
    c = -1;
    d = -50;
    printf("\n0 maior valor entre (%d,%d) eh %d", c,d,maior(c,d));
```

```
    return 0;
```

```
}
```

Endereço	Blocos (1 byte)	Nome variável	Tipo	Endereço	Blocos (1 byte)	Nome variável	Tipo
0 / NULL	indefinido	----	----	47			int
1	1	a	int	48			
2				49			
3				50			
4				51			
5	2	b	int	52			
6				53			
7				54			
8				55			
9	lx	c	int	56			
10				57			
11				58			
12				59			
13	lx	d	int	60			
14				61			
15				62			
16				63			
17	lx	m		64			
18				65			
19				66			
20				67			
21				68			

Exemplo

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

```
int maior(int a, int b){
    if ( a > b)
        return a;
    else
        return b;
}
```

```
int main()
{
    int a,b,c,d,m;
```

```
    a = 1;
```

```
    b = 2;
```

```
    m = maior(a,b);
```

```
    printf("\n0 maior valor entre (%d,%d) eh %d", a,b,m);
```

```
    c = -1;
```

```
    d = -50;
```

```
    printf("\n0 maior valor entre (%d,%d) eh %d", c,d,maior(c,d));
```

```
    return 0;
```

```
}
```

Endereço	Blocos (1 byte)	Nome variável	Tipo	Endereço	Blocos (1 byte)	Nome variável	Tipo
0 / NULL	indefinido	----	----	47			int
1	1	a	int	48			
2				49			
3				50			
4				51			
5	2	b	int	52			
6				53			
7				54			
8				55			
9	1x	c	int	56			
10				57			
11				58			
12				59			
13	1x	d	int	60			
14				61			
15				62			
16				63			
17	1x	m		64			
18				65			
19				66			
20				67			
21				68			

Exemplo

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

```
int maior(int a, int b){
    if ( a > b)
        return a;
    else
        return b;
}
```

```
int main()
{
    int a,b,c,d,m;

    a = 1;
    b = 2;
    m = maior(a,b);
    printf("\n0 maior valor entre (%d,%d) eh %d", a,b,m);

    c = -1;
    d = -50;
    printf("\n0 maior valor entre (%d,%d) eh %d", c,d,maior(c,d));

    return 0;
}
```

Endereço	Blocos (1 byte)	Nome variável	Tipo	Endereço	Blocos (1 byte)	Nome variável	Tipo
0 / NULL	indefinido	----	----	47			
1	1	a	int	48	1	a	int
2				49			
3				50			
4				51		b	int
5	2	b	int	52	2		
6				53			
7				54			
8				55			
9	1x	c	int	56			
10				57			
11				58			
12				59			
13	1x	d	int	60			
14				61			
15				62			
16				63			
17	1x	m		64			
18				65			
19				66			
20				67			
21				68			
22				69			
23				70			
24				71			

Exemplo

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

```
int maior(int a, int b){
    if ( a > b)
        return a;
    else
        return b;
}
```

```
int main()
{
    int a,b,c,d,m;

    a = 1;
    b = 2;
    m = maior(a,b);
    printf("\n0 maior valor entre (%d,%d) eh %d", a,b,m);

    c = -1;
    d = -50;
    printf("\n0 maior valor entre (%d,%d) eh %d", c,d,maior(c,d));

    return 0;
}
```

Endereço	Blocos (1 byte)	Nome variável	Tipo
0 / NULL	indefinido	----	----
1	1	a	int
2			
3			
4			
5	2	b	int
6			
7			
8			
9	lx	c	int
10			
11			
12			
13	lx	d	int
14			
15			
16			
17	lx	m	
18			
19			
20			
21			

Endereço	Blocos (1 byte)	Nome variável	Tipo
47	1	a	int
48			
49			
50			
51	2	b	int
52			
53			
54			
55			
56			
57			
58			
59			
60			
61			
62			
63			
64			
65			
66			
67			
68			

Variáveis com mesmo nome das variáveis do programa principal, mas com escopo diferente

Exemplo

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

```
int maior(int a, int b){
    if ( a > b)
        return a;
    else
        return b;
}
```

```
int main()
{
```

```
    int a,b,c,d,m;
```

```
    a = 1;
```

```
    b = 2;
```

```
    m = maior(a,b);
```

```
    printf("\n0 maior valor entre (%d,%d) eh %d", a,b,m);
```

```
    c = -1;
```

```
    d = -50;
```

```
    printf("\n0 maior valor entre (%d,%d) eh %d", c,d,maior(c,d));
```

```
    return 0;
```

```
}
```

Endereço	Blocos (1 byte)	Nome variável	Tipo
0 / NULL	indefinido	----	----
1	1	a	int
2			
3			
4			
5	2	b	int
6			
7			
8			
9	lx	c	int
10			
11			
12			
13	lx	d	int
14			
15			
16			
17	lx	m	
18			
19			
20			
21			

Endereço	Blocos (1 byte)	Nome variável	Tipo
47	1	a	int
48			
49			
50			
51	2	b	int
52			
53			
54			
55			
56			
57			
58			
59			
60			
61			
62			
63			
64			
65			
66			
67			
68			

Ao término da função, suas variáveis são **destruídas (apagadas)** da memória

Exemplo

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

```
int maior(int a, int b){
    if ( a > b)
        return a;
    else
        return b;
}
```

```
int main()
{
    int a,b,c,d,m;
```

```
    a = 1;
```

```
    b = 2;
```

```
    m = maior(a,b);
```

```
    printf("\n0 maior valor entre (%d,%d) eh %d", a,b,m);
```

```
    c = -1;
```

```
    d = -50;
```

```
    printf("\n0 maior valor entre (%d,%d) eh %d", c,d,maior(c,d));
```

```
    return 0;
```

```
}
```

Endereço	Blocos (1 byte)	Nome variável	Tipo
0 / NULL	indefinido	----	----
1	1	a	int
2			
3			
4			
5	2	b	int
6			
7			
8			
9	1x	c	int
10			
11			
12			
13	1x	d	int
14			
15			
16			
17	2	m	
18			
19			
20			
21			

Endereço	Blocos (1 byte)	Nome variável	Tipo
47			
48			
49			
50			
51			
52			
53			
54			
55			
56			
57			
58			
59			
60			
61			
62			
63			
64			
65			
66			
67			
68			

m recebe o valor retornado pela função

Exemplo

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

```
int maior(int a, int b){
    if ( a > b)
        return a;
    else
        return b;
}
```

```
int main()
{
    int a,b,c,d,m;
```

```
    a = 1;
    b = 2;
    m = maior(a,b);
    printf("\n0 maior valor entre (%d,%d) eh %d", a,b,m);
```

```
    c = -1;
    d = -50;
    printf("\n0 maior valor entre (%d,%d) eh %d", c,d,maior(c,d));
```

```
    return 0;
```

```
}
```

Endereço	Blocos (1 byte)	Nome variável	Tipo
0 / NULL	indefinido	----	----
1	1	a	int
2			
3			
4			
5	2	b	int
6			
7			
8			
9	-1	c	int
10			
11			
12			
13	-50	d	int
14			
15			
16			
17	2	m	
18			
19			
20			
21			

Endereço	Blocos (1 byte)	Nome variável	Tipo
47			
48			
49			
50			
51			
52			
53			
54			
55			
56			
57			
58			
59			
60			
61			
62			
63			
64			
65			
66			
67			
68			

Exemplo

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

```
int maior(int a, int b){
    if ( a > b)
        return a;
    else
        return b;
}
```

```
int main()
{
    int a,b,c,d,m;

    a = 1;
    b = 2;
    m = maior(a,b);
    printf("\n0 maior valor entre (%d,%d) eh %d", a,b,m);

    c = -1;
    d = -50;
    printf("\n0 maior valor entre (%d,%d) eh %d", c,d,maior(c,d));

    return 0;
}
```

Endereço	Blocos (1 byte)	Nome variável	Tipo
0 / NULL	indefinido	----	----
1	1	a	int
2			
3			
4			
5	2	b	int
6			
7			
8			
9	-1	c	int
10			
11			
12			
13	50	d	int
14			
15			
16			
17	2	m	
18			
19			
20			
21			

Endereço	Blocos (1 byte)	Nome variável	Tipo
47	-1	a	int
48			
49			
50	-50	b	int
51			
52			
53	-		
54			
55			
56			
57			
58			
59			
60			
61			
62			
63			
64			
65			
66			
67			
68			

Na segunda chamada da função, suas variáveis são criadas novamente e recebem os valores passado por parâmetro

Exemplo

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

```
int maior(int a, int b){
    if ( a > b)
        return a;
    else
        return b;
}
```

```
int main()
{
    int a,b,c,d,m;

    a = 1;
    b = 2;
    m = maior(a,b);
    printf("\n0 maior valor entre (%d,%d) eh %d", a,b,m);

    c = -1;
    d = -50;
    printf("\n0 maior valor entre (%d,%d) eh %d", c,d,maior(c,d));

    return 0;
}
```

Endereço	Blocos (1 byte)	Nome variável	Tipo
0 / NULL	indefinido	----	----
1	1	a	int
2			
3			
4			
5	2	b	int
6			
7			
8			
9	-1	c	int
10			
11			
12			
13	50	d	int
14			
15			
16			
17	2	m	
18			
19			
20			
21			

Endereço	Blocos (1 byte)	Nome variável	Tipo
47	-1	a	int
48			
49			
50	-50	b	int
51			
52			
53			
54	[apagadas]		
55			
56			
57			
58			
59			
60			
61			
62			
63			
64			
65			
66			
67			
68			

Ao término da função, as variáveis são apagadas

Exemplo

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

```
int maior(int a, int b){
    if ( a > b)
        return a;
    else
        return b;
}
```

```
int main()
{
    int a,b,c,d,m;

    a = 1;
    b = 2;
    m = maior(a,b);
    printf("\n0 maior valor entre (%d,%d) eh %d", a,b,m);

    c = -1;
    d = -50;
    printf("\n0 maior valor entre (%d,%d) eh %d", c,d,maior(c,d));

    return 0;
}
```

Endereço	Blocos (1 byte)	Nome variável	Tipo
0 / NULL	indefinido	----	----
1	1	a	int
2			
3			
4			
5	2	b	int
6			
7			
8			
9	-1	c	int
10			
11			
12			
13	-50	d	int
14			
15			
16			
17	2	m	
18			
19			
20			
21			

Endereço	Blocos (1 byte)	Nome variável	Tipo
47			
48			
49			
50			
51			
52			
53			
54			
55			
56			
57			
58			
59			
60			
61			
62			
63			
64			
65			
66			
67			
68			

Passagem de Parâmetros

- ▶ Na linguagem C, os parâmetros de uma função são sempre passados por **valor**, ou seja, uma cópia do valor do parâmetro é feita e passada para a função.
- ▶ Mesmo que esse valor mude dentro da função, nada acontece com o valor de fora da função.



Passagem por valor

```
void soma_mais_um(int x) {  
    x = x + 1;  
    printf("Dentro da funcao: x = %d\n" , x);  
}
```

```
int main()  
{  
    int x = 5;  
    printf("Antes da funcao: x = %d\n",x);  
  
    soma_mais_um(x);  
  
    printf("Depois da funcao: x = %d\n",x);  
  
    return 0;  
}
```



Passagem por valor

```
void soma_mais_um(int x) {  
    x = x + 1;  
    printf("Dentro da funcao: x = %d\n" , x);  
}
```

```
int main()  
{  
    int x = 5;  
    printf("Antes da funcao: x = %d\n",x);  
  
    soma_mais_um(x);  
  
    printf("Depois da funcao: x = %d\n",x);  
  
    return 0;  
}
```

Saída

Antes da funcao: x = 5
Dentro da funcao: x = 6
Depois da funcao: x = 5



Passagem por valor

```
void soma_mais_um(int x) {  
    x = x + 1;  
    printf("Dentro da funcao: x = %d\n", x);  
}
```

```
int main()  
{  
    int x = 1;  
    printf("Antes da funcao: x = %d\n", x);  
  
    soma_mais_um(x);  
  
    printf("Depois da funcao: x = %d\n", x);  
  
    return 0;  
}
```

Não conseguimos mudar o valor de x (que pertence ao main) por meio da função



Mapa

Endereço	Blocos (1 byte)	Nome variável	Tipo
0 / NULL	indefinido	----	----
1	5	x	int
2			
3			
4			
5			
6			
7			
8			
9			
10			

Endereço	Blocos (1 byte)	Nome variável	Tipo
47			
48			
49			
50			
51			
52			
53			
54			
55			
56			
57			

```
void soma_mais_um(int x) {  
    x = x + 1;  
    printf("Dentro da funcao: x = %d\n" , x);  
}
```

```
int main()  
{  
    int x = 5;  
    printf("Antes da funcao:  x = %d\n",x);  
    soma_mais_um(x);  
  
    printf("Depois da funcao:  x = %d\n",x);  
  
    return 0;  
}
```

Mapa

Endereço	Blocos (1 byte)	Nome variável	Tipo
0 / NULL	indefinido	----	----
1	5	x	int
2			
3			
4			
5			
6			
7			
8			
9			
10			

Endereço	Blocos (1 byte)	Nome variável	Tipo
47			
48	5	x	int
49			
50			
51			
52			
53			
54			
55			
56			
57			

```
void soma_mais_um(int x) {
    x = x + 1;
    printf("Dentro da funcao: x = %d\n" , x);
}
```

```
int main()
{
    int x = 5;
    printf("Antes da funcao: x = %d\n",x);
```

```
    soma_mais_um(x);

    printf("Depois da funcao: x = %d\n",x);

    return 0;
```

```
}
```

Mapa

Endereço	Blocos (1 byte)	Nome variável	Tipo
0 / NULL	indefinido	----	----
1	5	x	int
2			
3			
4			
5			
6			
7			
8			
9			
10			

Endereço	Blocos (1 byte)	Nome variável	Tipo
47			
48	5	x	int
49			
50			
51			
52			
53			
54			
55			
56			
57			

```
void soma_mais_um(int x) {
    x = x + 1;
    printf("Dentro da funcao: x = %d\n" , x);
}
```

```
int main()
{
    int x = 5;
    printf("Antes da funcao: x = %d\n",x);

    soma_mais_um(x);

    printf("Depois da funcao: x = %d\n",x);

    return 0;
}
```

Observe que uma nova variável x é criada, em um outro escopo (da função soma_mais_um). Essa variável é inicializada com o valor passado como parâmetro

Mapa

Endereço	Blocos (1 byte)	Nome variável	Tipo
0 / NULL	indefinido	----	----
1	5	x	int
2			
3			
4			
5			
6			
7			
8			
9			
10			

Endereço	Blocos (1 byte)	Nome variável	Tipo
47			
48	6	x	int
49			
50			
51			
52			
53			
54			
55			
56			
57			

```
void soma_mais_um(int x) {  
    x = x + 1;  
    printf("Dentro da funcao: x = %d\n", x);  
}
```

```
int main()  
{  
    int x = 5;  
    printf("Antes da funcao: x = %d\n", x);
```

→ soma_mais_um(x);

```
    printf("Depois da funcao: x = %d\n", x);
```

```
    return 0;
```

```
}
```

Mapa

Endereço	Blocos (1 byte)	Nome variável	Tipo
0 / NULL	indefinido	----	----
1	5	x	int
2			
3			
4			
5			
6			
7			
8			
9			
10			

Endereço	Blocos (1 byte)	Nome variável	Tipo
47	6		
48		x	int
49			
50			
51			
52			
53			
54			
55			
56			
57			

```
void soma_mais_um(int x) {  
    x = x + 1;  
    printf("Dentro da funcao: x = %d\n", x);  
}
```

```
int main()  
{  
    int x = 5;  
    printf("Antes da funcao: x = %d\n", x);  
    soma_mais_um(x);  
    printf("Depois da funcao: x = %d\n", x);  
    return 0;  
}
```

Observe que a variável x
(do escopo da função)
mudou de valor,
enquanto que a variável x
do programa principal
permanece inalterada

Mapa

Endereço	Blocos (1 byte)	Nome variável	Tipo
0 / NULL	indefinido	----	----
1	5	x	int
2			
3			
4			
5			
6			
7			
8			
9			
10			

Endereço	Blocos (1 byte)	Nome variável	Tipo
47			
48			
49			
50			
51			
52			
53			
54			
55			
56			
57			

```
void soma_mais_um(int x) {  
    x = x + 1;  
    printf("Dentro da funcao: x = %d\n" , x);  
}
```

```
int main()  
{  
    int x = 5;  
    printf("Antes da funcao: x = %d\n",x);
```

```
    soma_mais_um(x);  
  
    printf("Depois da funcao: x = %d\n",x);  
  
    return 0;  
}
```


Mapa

Endereço	Blocos (1 byte)	Nome variável	Tipo
0 / NULL	indefinido	----	----
1	5	x	int
2			
3			
4			
5			
6			
7			
8			
9			
10			

Endereço	Blocos (1 byte)	Nome variável	Tipo
47			
48			
49			
50			
51			
52			
53			
54			
55			
56			
57			

```
void soma_mais_um(int x) {  
    x = x + 1;  
    printf("Dentro da funcao: x = %d\n" , x);  
}
```

```
int main()  
{  
    int x = 5;  
    printf("Antes da funcao:  x = %d\n",x);
```

```
    soma_mais_um(x);  
  
    printf("Depois da funcao:  x = %d\n",x);  
  
    return 0;  
}
```

Ao término da função, x (do escopo da função) é apagada.

Passagem por referência

- ▶ Quando se quer que o valor da variável mude dentro da função, usa-se passagem de parâmetros por ***referência***.
- ▶ Neste tipo de chamada, não se passa para a função o valor da variável, mas a sua ***referência*** (seu endereço na memória);



Passagem por referência

- ▶ Para passar um parâmetro por referência, coloca-se um asterisco “*” na frente do nome do parâmetro na declaração da função (ou seja, um ponteiro):

float sqr (float *num);

- ▶ Ao se chamar a função, é necessário agora utilizar o operador “&”, igual como é feito com a função **scanf()**:

y = sqr(&x);



Passagem por referência

- ▶ No corpo da função, é necessário usar colocar um asterisco “*” sempre que se desejar acessar o conteúdo do parâmetro passado por referência.

Por valor

```
void soma_mais_um(int n){  
    n = n + 1;  
}
```

Por referência

```
void soma_mais_um(int *n){  
    *n = *n + 1;  
}
```

Passagem por referência

```
void soma_mais_um(int *x){  
    *x = *x + 1;  
    printf("Dentro da funcao: x = %d\n" , *x);  
}
```

```
int main()  
{  
    int x = 5;  
    printf("Antes da funcao:  x = %d\n",x);  
  
    soma_mais_um(&x);  
  
    printf("Depois da funcao:  x = %d\n",x);  
  
    return 0;  
}
```



Passagem por referência

```
void soma_mais_um(int *x){
    *x = *x + 1;
    printf("Dentro da funcao: x = %d\n", *x);
}

int main()
{
    int x = 5;
    printf("Antes da funcao: x = %d\n", x);

    soma_mais_um(&x);

    printf("Depois da funcao: x = %d\n", x);

    return 0;
}
```

Saída

Antes da funcao: x = 5
Dentro da funcao: x = 6
Depois da funcao: x = 6



Passagem por referência

```
void soma_mais_um(int *x){  
    *x = *x + 1;  
    printf("Depois da funcao:  x = %d\n",*x);  
}
```

```
int main()  
{  
    int x = 5;  
    printf("Antes da funcao:  x = %d\n",x);  
  
    soma_mais_um(&x);  
  
    printf("Depois da funcao:  x = %d\n",x);  
  
    return 0;  
}
```

Conseguimos mudar o valor de x (que pertence ao main) por meio da função



Mapa

Endereço	Blocos (1 byte)	Nome variável	Tipo
0 / NULL	indefinido	----	----
1	5	x	int
2			
3			
4			
5			
6			
7			
8			
9			
10			

Endereço	Blocos (1 byte)	Nome variável	Tipo
47			
48			
49			
50			
51			
52			
53			
54			
55			
56			
57			

```
void soma_mais_um(int *x){  
    *x = *x + 1;  
    printf("Dentro da funcao: x = %d\n" , *x);  
}
```

```
int main()  
{  
    int x = 5;  
    printf("Antes da funcao:  x = %d\n",x);  
    soma_mais_um(&x);  
    printf("Depois da funcao:  x = %d\n",x);  
    return 0;  
}
```



Mapa

Endereço	Blocos (1 byte)	Nome variável	Tipo
0 / NULL	indefinido	----	----
1	5	x	int
2			
3			
4			
5			
6			
7			
8			
9			
10			

Endereço	Blocos (1 byte)	Nome variável	Tipo
47			
48	1	x	int *
49			
50			
51			
52			
53			
54			
55			
56			
57			

```
void soma_mais_um(int *x){
    *x = *x + 1;
    printf("Dentro da funcao: x = %d\n" , *x);
}
```

```
int main()
{
    int x = 5;
    printf("Antes da funcao:  x = %d\n",x);
```

```
    soma_mais_um(&x);

    printf("Depois da funcao:  x = %d\n",x);
```

```
    return 0;
}
```

Mapa

Endereço	Blocos (1 byte)	Nome variável	Tipo
0 / NULL	indefinido	----	----
1	5	x	int
2			
3			
4			
5			
6			
7			
8			
9			
10			

Endereço	Blocos (1 byte)	Nome variável	Tipo
47			
48	1	x	int *
49			
50			
51			
52			
53			
54			
55			
56			
57			

```
void soma_mais_um(int *x){
    *x = *x + 1;
    printf("Dentro da funcao: x = %d\n" , *x);
}
```

```
int main()
{
    int x = 5;
    printf("Antes da funcao:  x = %d\n",x);
```

```
    soma_mais_um(&x);

    printf("Depois da funcao:  x = %d\n",x);

    return 0;
}
```

Obseve que o tipo de x da função é int *

Mapa

Endereço	Blocos (1 byte)	Nome variável	Tipo
0 / NULL	indefinido	----	----
1	5	x	int
2			
3			
4			
5			
6			
7			
8			
9			
10			

Endereço	Blocos (1 byte)	Nome variável	Tipo
47			
48	1	x	int *
49			
50			
51			
52			
53			
54			
55			
56			
57			

```
void soma_mais_um(int *x){
    *x = *x + 1;
    printf("Dentro da funcao: x = %d\n" , *x);
}
```

```
int main()
{
    int x = 5;
    printf("Antes da funcao:  x = %d\n",x);
```

```
    soma_mais_um(&x);

    printf("Depois da funcao:  x = %d\n",x);

    return 0;
}
```

Observe que o valor copiado para x da função é 1, e não 5, pois foi passada a função o endereço de x, e não x

Mapa

Endereço	Blocos (1 byte)	Nome variável	Tipo
0 / NULL	indefinido	----	----
1	6	x	int
2			
3			
4			
5			
6			
7			
8			
9			
10			

Endereço	Blocos (1 byte)	Nome variável	Tipo
47			
48	1	x	int *
49			
50			
51			
52			
53			
54			
55			
56			
57			

```
void soma_mais_um(int *x){
    *x = *x + 1;
    printf("Dentro da funcao: x = %d\n" , *x);
}
```

```
int main()
{
    int x = 5;
    printf("Antes da funcao:  x = %d\n",x);
```

```
    soma_mais_um(&x);

    printf("Depois da funcao:  x = %d\n",x);

    return 0;
}
```

Mapa

Endereço	Blocos (1 byte)	Nome variável	Tipo
0 / NULL	indefinido	----	----
1		x	int
2			
3			
4			
5			
6			
7			
8			
9			
10			

Endereço	Blocos (1 byte)	Nome variável	Tipo
47			
48		x	int *
49			
50			
51			
52			
53			
54			
55			
56			
57			

```
void soma_mais_um(int *x){  
    *x = *x + 1;  
    printf("Dentro da funcao: x = %d\n" , *x);  
}
```

```
int main()  
{  
    int x = 5;  
    printf("Antes da funcao:  x = %d\n",x);  
    soma_mais_um(&x);  
    printf("Depois da funcao:  x = %d\n",x);  
    return 0;  
}
```

Observe que o valor de x (pertencente ao main) foi alterado pela função, por meio de um ponteiro

Mapa


Endereço	Blocos (1 byte)	Nome variável	Tipo
0 / NULL	indefinido	----	----
1	6	x	int
2			
3			
4			
5			
6			
7			
8			

Endereço	Blocos (1 byte)	Nome variável	Tipo
47			
48			
49			
50			
51			
52			
53			
54			
55			

```
void soma_mais_um(int *x){
    *x = *x + 1;
    printf("Dentro da funcao: x = %d\n" , *x);
}
```



```
int main()
{
    int x = 5;
    printf("Antes da funcao:  x = %d\n",x);
```

 soma_mais_um(&x);

```
    printf("Depois da funcao:  x = %d\n",x);
```

```
    return 0;
```

```
}-----  

```

Mapa

Endereço	Blocos (1 byte)	Nome variável	Tipo
0 / NULL	indefinido	----	----
1	6	x	int
2			
3			
4			
5			
6			
7			
8			

Endereço	Blocos (1 byte)	Nome variável	Tipo
47			
48			
49			
50			
51			
52			
53			
54			
55			

```
void soma_mais_um(int *x){
    *x = *x + 1;
    printf("Dentro da funcao: x = %d\n" , *x);
}
```



```
int main()
{
    int x = 5;
    printf("Antes da funcao:  x = %d\n",x);
    soma_mais_um(&x);
    printf("Depois da funcao:  x = %d\n",x);
    return 0;
}
```

Ao fim da função,
o ponteiro x é
apagado

Mapa

Endereço	Blocos (1 byte)	Nome variável	Tipo
0 / NULL	indefinido	----	----
1	6	x	int
2			
3			
4			
5			
6			
7			
8			

Endereço	Blocos (1 byte)	Nome variável	Tipo
47			
48			
49			
50			
51			
52			
53			
54			
55			

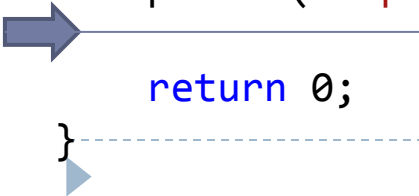
```
void soma_mais_um(int *x){
    *x = *x + 1;
    printf("Dentro da funcao: x = %d\n" , *x);
}
```

```
int main()
{
    int x = 5;
    printf("Antes da funcao:  x = %d\n",x);

    soma_mais_um(&x);

    printf("Depois da funcao:  x = %d\n",x);

    return 0;
}
```



Passagem por referência

- ▶ Utilizando o endereço da variável, qualquer alteração que a variável sofra dentro da função será refletida fora da função. Ex: função `scanf()`
- ▶ sempre que desejamos ler algo do teclado, passamos para a função **`scanf()`** o nome da variável onde o dado será armazenado. Essa variável tem seu valor modificado dentro da função **`scanf()`**, e seu valor pode ser acessado no programa principal



Passagem por referência

```
01  #include <stdio.h>
02  #include <stdlib.h>
03  int main(){
04      int x = 5;
05      printf("Antes do scanf: x = %d\n",x);
06      printf("Digite um numero: ");
07      scanf("%d",&x);
08      printf("Depois do scanf: x = %d\n",x);
09      system("pause");
10      return 0;
11  }
```



Exercício

- ▶ Crie uma função que troque o valor de dois números inteiros passados por referência.



Exercício

```
void Troca (int*a,int*b){  
    int temp;  
    temp = *a;  
    *a = *b;  
    *b = temp;  
}
```



Arrays como parâmetros

- ▶ Para utilizar arrays como parâmetros de funções alguns cuidados simples são necessários.



Arrays como parâmetros

- ▶ Arrays são sempre passados por referência para uma função;
 - ▶ A passagem de arrays **por referência** evita a cópia desnecessária de grandes quantidades de dados para outras áreas de memória durante a chamada da função, o que afetaria o desempenho do programa.



Arrays como parâmetros

- ▶ É necessário declarar um segundo parâmetro (em geral uma variável inteira) para passar para a função o tamanho do array separadamente.
- ▶ Quando passamos um array por parâmetro, independente do seu tipo, o que é de fato passado é o endereço do primeiro elemento do array.



Arrays como parâmetros

- ▶ Na passagem de um array como parâmetro de uma função podemos declarar a função de diferentes maneiras, todas equivalentes:

```
void imprime (int *m, int n);
```

```
void imprime (int m[], int n);
```

```
void imprime (int m[5], int n);
```



Arrays como parâmetros

```
void imprime (int *m,int n){  
    int i;  
    for (i=0; i< n;i++)  
        printf ("%d \n", m[i]);  
}
```

```
int main (){  
    int n[5] = {1,2,3,4,5};  
    → imprime(n,5);  
    return 0;  
}
```

Endereço	Blocos (1 byte)	Nome variável	Tipo
0 / NULL	indefinido	----	----
11	1	n[0]	int
12			
13			
14			
15	2	n[1]	int
16			
17			
18			
19	3	n[2]	int
20			
21			
22			
23	4	n[3]	int
24			
25			
26			
27	5	n[4]	int
28			
29			
30			
31			

Endereço	Blocos (1 byte)	Nome variável	Tipo
47			
48			
49			
50			
51			
52			
53			
54			
55			
56			
57			
58			
59			
60			
61			
62			
63			
64			
65			
66			
67			
68			

Arrays como parâmetros

```
void imprime (int *m,int n){  
    int i;  
    for (i=0; i< n;i++)  
        printf ("%d \n", m[i]);  
}
```

```
int main (){  
    int n[5] = {1,2,3,4,5};  
    → imprime(n,5);  
    return 0;  
}
```

Endereço	Blocos (1 byte)	Nome variável	Tipo
0 / NULL	indefinido	----	----
11	1	n[0]	int
12			
13			
14			
15	2	n[1]	int
16			
17			
18			
19	3	n[2]	int
20			
21			
22			
23	4	n[3]	int
24			
25			
26			
27	5	n[4]	int
28			
29			
30			
31			

Endereço	Blocos (1 byte)	Nome variável	Tipo
47			
48			
49			
50			
51			
52			
53			
54			
55			
56			
57			
58			
59			
60			
61			
62			
63			
64			
65			
66			
67			
68			

Arrays como parâmetros

```
void imprime (int *m,int n){  
    int i;  
    for (i=0; i< n;i++)  
        printf ("%d \n", m[i]);  
}
```

```
int main (){  
    int n[5] = {1,2,3,4,5};  
    → imprime(n,5);  
    return 0;  
}
```

Endereço	Blocos (1 byte)	Nome variável	Tipo
0 / NULL	indefinido	----	----
11	1	n[0]	int
12			
13			
14			
15	2	n[1]	int
16			
17			
18			
19	3	n[2]	int
20			
21			
22			
23	4	n[3]	int
24			
25			
26			
27	5	n[4]	int
28			
29			
30			
31			

Endereço	Blocos (1 byte)	Nome variável	Tipo
47			
48	?	m	int*
49			
50			
51			
52	?	n	int
53			
54			
55			
56			
57			
58			
59			
60			
61			
62			
63			
64			
65			
66			
67			
68			

Arrays como parâmetros

```
void imprime (int *m,int n){  
    int i;  
    for (i=0; i< n;i++)  
        printf ("%d \n", m[i]);  
}
```

```
int main (){  
    int n[5] = {1,2,3,4,5};  
    → imprime(n,5);  
    return 0;  
}
```

Endereço	Blocos (1 byte)	Nome variável	Tipo
0 / NULL	indefinido	----	----
11	1	n[0]	int
12			
13			
14			
15	2	n[1]	int
16			
17			
18			
19	3	n[2]	int
20			
21			
22			
23	4	n[3]	int
24			
25			
26			
27	5	n[4]	int
28			
29			
30			
31			

Endereço	Blocos (1 byte)	Nome variável	Tipo
47	11		
48		m	int*
49			
50			
51	5		
52		n	int
53			
54			
55			
56			
57			
58			
59			
60			
61			
62			
63			
64			
65			
66			
67			
68			

Arrays como parâmetros

```
void imprime (int *m,int n){  
    int i;  
    for (i=0; i< n;i++)  
        printf ("%d \n", m[i]);  
}
```

```
int main (){  
    int n[5] = {1,2,3,4,5};  
    imprime(n,5);  
    return 0;  
}
```

Endereço	Blocos (1 byte)	Nome variável	Tipo	Endereço	Blocos (1 byte)	Nome variável	Tipo
0 / NULL	indefinido	----	----	47			
11		n[0]	int	48		m	int*
12	1			49	11		
13				50			
14				51			
15		n[1]	int	52			
16	2			53	5	n	int
17				54			
18				55			
19		n[2]	int	56			
20	3			57			
21				58			
22				59			
23		n[3]	int	60			
24	4			61			
25				62			
26				63			
27		n[4]	int	64			
28	5			65			
29				66			
30				67			
31				68			

Arrays como parâmetros

```
void imprime (int *m,int n){
    int i;
    for (i=0; i< n;i++)
        printf ("%d \n", m[i]);
}
```

```
int main (){
    int n[5] = {1,2,3,4,5};
    imprime(n,5);
    return 0;
}
```

Endereço	Blocos (1 byte)	Nome variável	Tipo
0 / NULL	indefinido	----	----
11	1	n[0]	int
12			
13			
14			
15	2	n[1]	int
16			
17			
18			
19	3	n[2]	int
20			
21			
22			
23	4	n[3]	int
24			
25			
26			
27	5	n[4]	int
28			
29			
30			

Endereço	Blocos (1 byte)	Nome variável	Tipo
47			
48	11	m	int*
49			
50			
51			
52	5	n	int
53			
54			
55			
56	ix	i	int
57			
58			
59			
60			
61			
62			
63			
64			
65			
66			
67			

Arrays como parâmetros

```
void imprime (int *m,int n){  
    int i;  
    for (i=0; i< n;i++)  
        printf ("%d \n", m[i]);  
}
```

```
int main (){  
    int n[5] = {1,2,3,4,5};  
    → imprime(n,5);  
    return 0;  
}
```

Endereço	Blocos (1 byte)	Nome variável	Tipo
0 / NULL	indefinido	----	----
11	1	n[0]	int
12			
13			
14	2	n[1]	int
15			
16			
17	3	n[2]	int
18			
19			
20	4	n[3]	int
21			
22			
23	5	n[4]	int
24			
25			
26			
27			
28			
29			
30			

Endereço	Blocos (1 byte)	Nome variável	Tipo
47			
48	11	m	int*
49			
50			
51	5	n	int
52			
53			
54	0	i	int
55			
56			
57			
58			
59			
60			
61			
62			
63			
64			
65			
66			
67			

Arrays como parâmetros

```
void imprime (int *m,int n){  
    int i;  
    for (i=0; i< n;i++)  
        printf ("%d \n", m[i]);  
}
```

```
int main (){  
    int n[5] = {1,2,3,4,5};  
    → imprime(n,5);  
    return 0;  
}
```

Endereço	Blocos (1 byte)	Nome variável	Tipo
0 / NULL	indefinido	----	----
11	1	n[0]	int
12			
13			
14			
15	2	n[1]	int
16			
17			
18			
19	3	n[2]	int
20			
21			
22			
23	4	n[3]	int
24			
25			
26			
27	5	n[4]	int
28			
29			
30			

Endereço	Blocos (1 byte)	Nome variável	Tipo
47			
48	11	m	int*
49			
50			
51			
52	5	n	int
53			
54			
55			
56	1	i	int
57			
58			
59			
60			
61			
62			
63			
64			
65			
66			
67			

Arrays como parâmetros

```
void imprime (int *m,int n){  
    int i;  
    for (i=0; i< n;i++)  
        printf ("%d \n", m[i]);  
}
```

```
int main (){  
    int n[5] = {1,2,3,4,5};  
    → imprime(n,5);  
    return 0;  
}
```

Endereço	Blocos (1 byte)	Nome variável	Tipo
0 / NULL	indefinido	----	----
11	1	n[0]	int
12			
13			
14			
15	2	n[1]	int
16			
17			
18			
19	3	n[2]	int
20			
21			
22			
23	4	n[3]	int
24			
25			
26			
27	5	n[4]	int
28			
29			
30			

Endereço	Blocos (1 byte)	Nome variável	Tipo
47			
48	11	m	int*
49			
50			
51			
52	5	n	int
53			
54			
55			
56	1	i	int
57			
58			
59			
60			
61			
62			
63			
64			
65			
66			
67			

Arrays como parâmetros

```
void imprime (int *m,int n){  
    int i;  
    for (i=0; i< n;i++)  
        printf ("%d \n", m[i]);  
}
```

```
int main (){  
    int n[5] = {1,2,3,4,5};  
    → imprime(n,5);  
    return 0;  
}
```

m[2]

Endereço	Blocos (1 byte)	Nome variável	Tipo
0 / NULL	indefinido	----	----
11	1	n[0]	int
12			
13			
14			
15	2	n[1]	int
16			
17			
18			
19	3	n[2]	int
20			
21			
22			
23	4	n[3]	int
24			
25			
26			
27	5	n[4]	int
28			
29			
30			

Endereço	Blocos (1 byte)	Nome variável	Tipo
47	11		
48		m	int*
49			
50			
51	5		
52		n	int
53			
54			
55	2		
56		i	int
57			
58			
59			
60			
61			
62			
63			
64			
65			
66			
67			

Arrays como parâmetros

```
void imprime (int *m,int n){  
    int i;  
    for (i=0; i< n;i++)  
        printf ("%d \n", m[i]);  
}
```

```
int main (){  
    int n[5] = {1,2,3,4,5};  
    → imprime(n,5);  
    return 0;  
}
```

Endereço	Blocos (1 byte)	Nome variável	Tipo
0 / NULL	indefinido	----	----
11	1	n[0]	int
12			
13			
14			
15	2	n[1]	int
16			
17			
18			
19	3	n[2]	int
20			
21			
22			
23	4	n[3]	int
24			
25			
26			
27	5	n[4]	int
28			
29			
30			

Endereço	Blocos (1 byte)	Nome variável	Tipo
47	11		
48		m	int*
49			
50			
51	5		
52		n	int
53			
54			
55	3		
56		i	int
57			
58			
59			
60			
61			
62			
63			
64			
65			
66			
67			

Arrays como parâmetros

- ▶ Vimos que para arrays, não é necessário especificar o número de elementos para a função.

```
void imprime (int*m, int n);
```

```
void imprime (int m[], int n);
```

- ▶ No entanto, para arrays com mais de uma dimensão, é necessário especificar o tamanho de todas as dimensões, exceto a primeira

```
void imprime (int m[][5], int n);
```



Arrays como parâmetros

- ▶ Na passagem de um array para uma função, o compilador precisar saber o tamanho de cada elemento, não o número de elementos.
- ▶ Uma matriz pode ser interpretada como um array de arrays.
 - ▶ **int m[4][5]**: array de 4 elementos onde cada elemento é um array de 5 posições inteiras.



Arrays como parâmetros

- ▶ Logo, o compilador precisa saber o tamanho de cada elemento do array.

```
int m[4][5]
```

```
void imprime (int m[][5], int n);
```

- ▶ Na notação acima, informamos ao compilador que estamos passando um array, onde cada elemento dele é outro array de 5 posições inteiras.



Arrays como parâmetros

- ▶ Isso é necessário para que o programa saiba que o array possui mais de uma dimensão e mantenha a notação de um conjunto de colchetes por dimensão.
- ▶ As notações abaixo funcionam para arrays com mais de uma dimensão. Mas o array é tratado como se tivesse apenas uma dimensão dentro da função
`void imprime (int*m, int n);`
`void imprime (int m[], int n);`



Recursão

- ▶ Na linguagem C, uma função pode chamar outra função.
 - ▶ A função `main()` pode chamar qualquer função, seja ela da biblioteca da linguagem (como a função `printf()`) ou definida pelo programador (função `imprime()`).
- ▶ Uma função também pode chamar a si própria
 - ▶ A qual chamamos de ***função recursiva***.



Recursão

- ▶ A recursão também é chamada de definição circular. Ela ocorre quando algo é definido em termos de si mesmo.
- ▶ Um exemplo clássico de função que usa recursão é o cálculo do fatorial de um número:
 - ▶ $3! = 3 * 2!$
 - ▶ $4! = 4 * 3!$
 - ▶ $n! = n * (n - 1)!$



Recursão

$$0! = 1$$

$$1! = 1 * 0!$$

$$2! = 2 * 1!$$

$$3! = 3 * 2!$$

$$4! = 4 * 3!$$

$$4! = 4 * 3!$$

$$3! = 3 * 2!$$

$$2! = 2 * 1!$$

$$1! = 1 * 0!$$

$$0! = 1$$

$$1! = 1 * 1$$

$$2! = 2 * 1$$

$$3! = 3 * 2!$$

$$4! = 4 * 6 = 24$$

$$n! = n * (n - 1)!$$

$$0! = 1 : \text{caso-base}$$

Ida

Volta



Recursão

Com Recursão

```
int fatorial (int n){  
    if (n == 0)  
        return 1;  
    else  
        return n*fatorial(n-1);  
}
```

Sem Recursão

```
int fatorial (int n){  
    if (n == 0)  
        return 1;  
    else {  
        int i;  
        int f=1;  
        for (i=2; i <= n;i++)  
            f = f * i;  
        return f;  
    }  
}
```



Recursão

- ▶ Em geral, formulações recursivas de algoritmos são freqüentemente consideradas "mais enxutas" ou "mais elegantes" do que formulações iterativas.
- ▶ Porém, algoritmos recursivos tendem a necessitar de mais espaço do que algoritmos iterativos.



Recursão

- ▶ Todo cuidado é pouco ao se fazer funções recursivas.
 - ▶ Critério de parada: determina quando a função deverá parar de chamar a si mesma.
 - ▶ O parâmetro da chamada recursiva deve ser sempre modificado, de forma que a recursão chegue a um término.



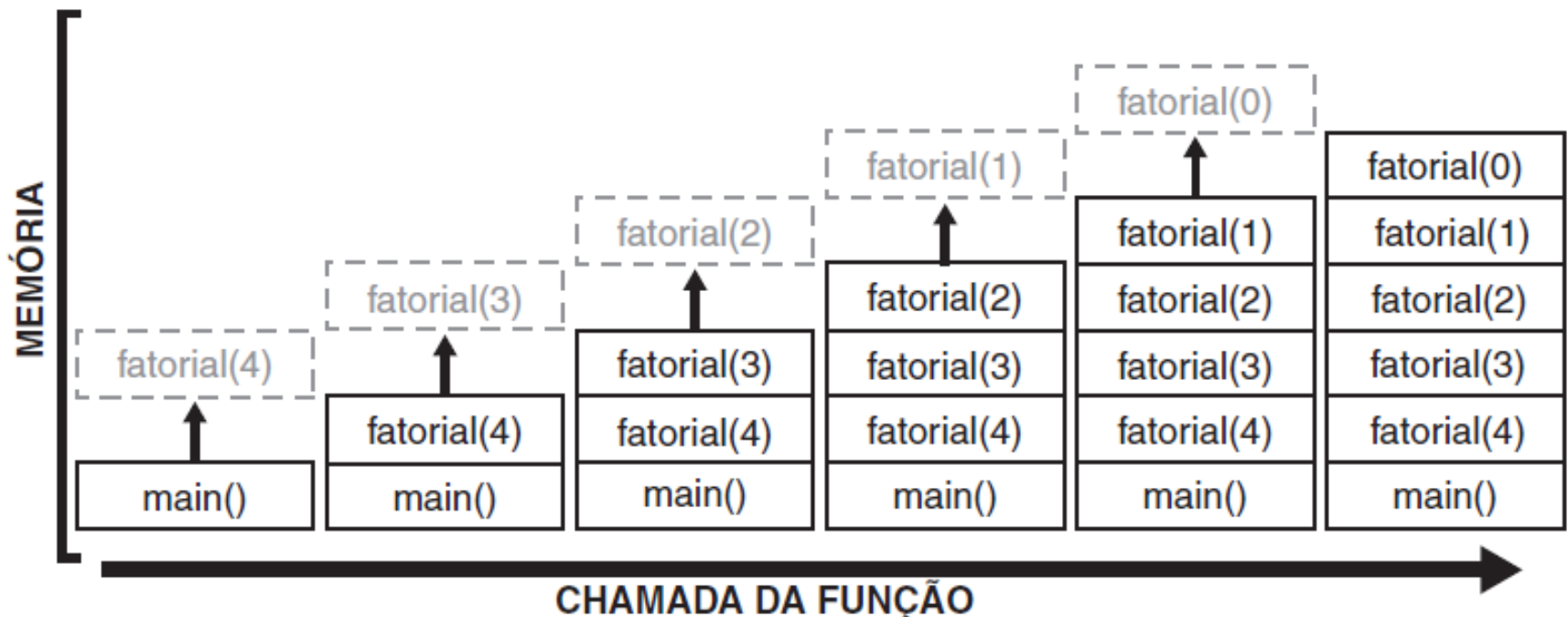
Recursão

```
int fatorial (int n){  
    if (n == 0)//critério de parada  
        return 1;  
    else  
        return n*fatorial(n-1); /*parâmetro de fatorial sempre  
        muda*/  
}
```



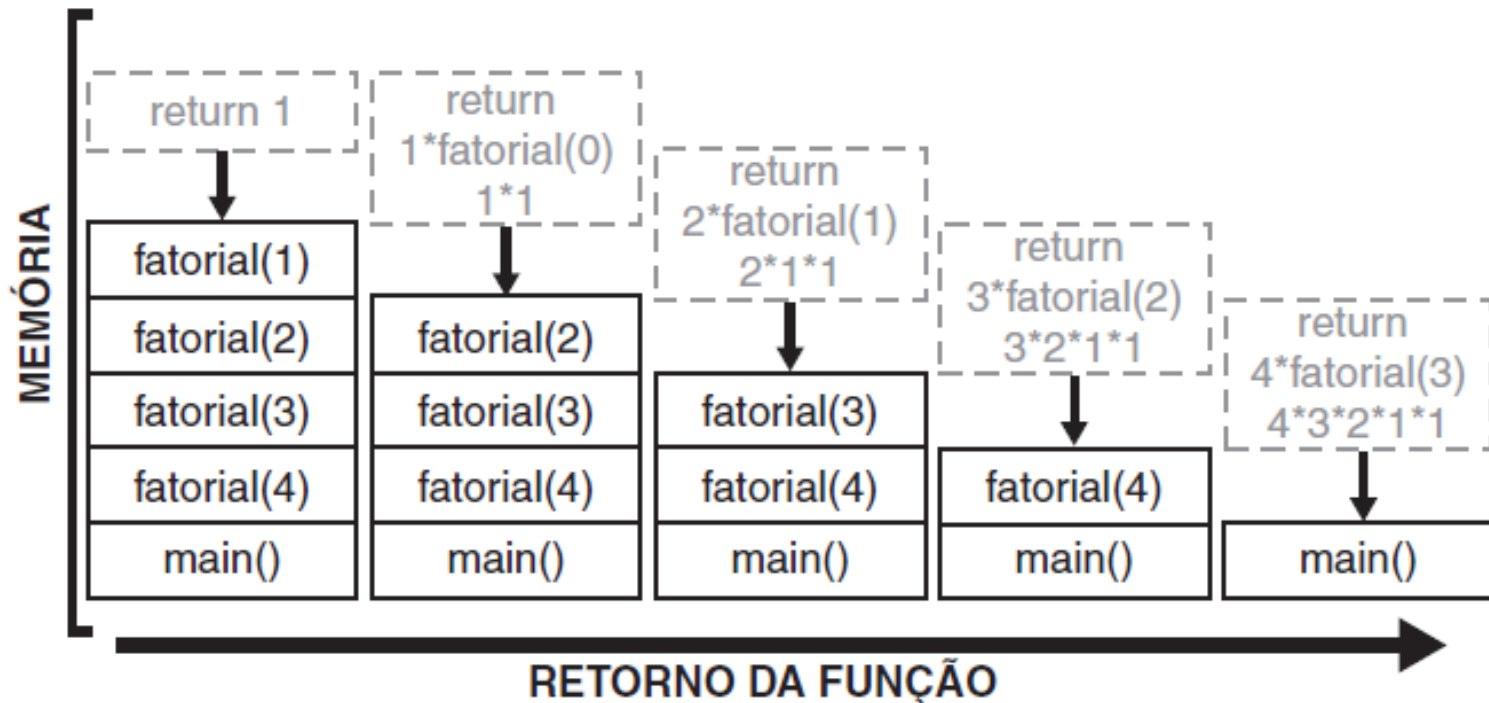
Recursão

- ▶ O que acontece na chamada da função fatorial com um valor como $n = 4$?
 - ▶ `int x = fatorial (4);`



Recursão

- Uma vez que chegamos ao caso-base, é hora de fazer o caminho de volta da recursão.



Fibonacci

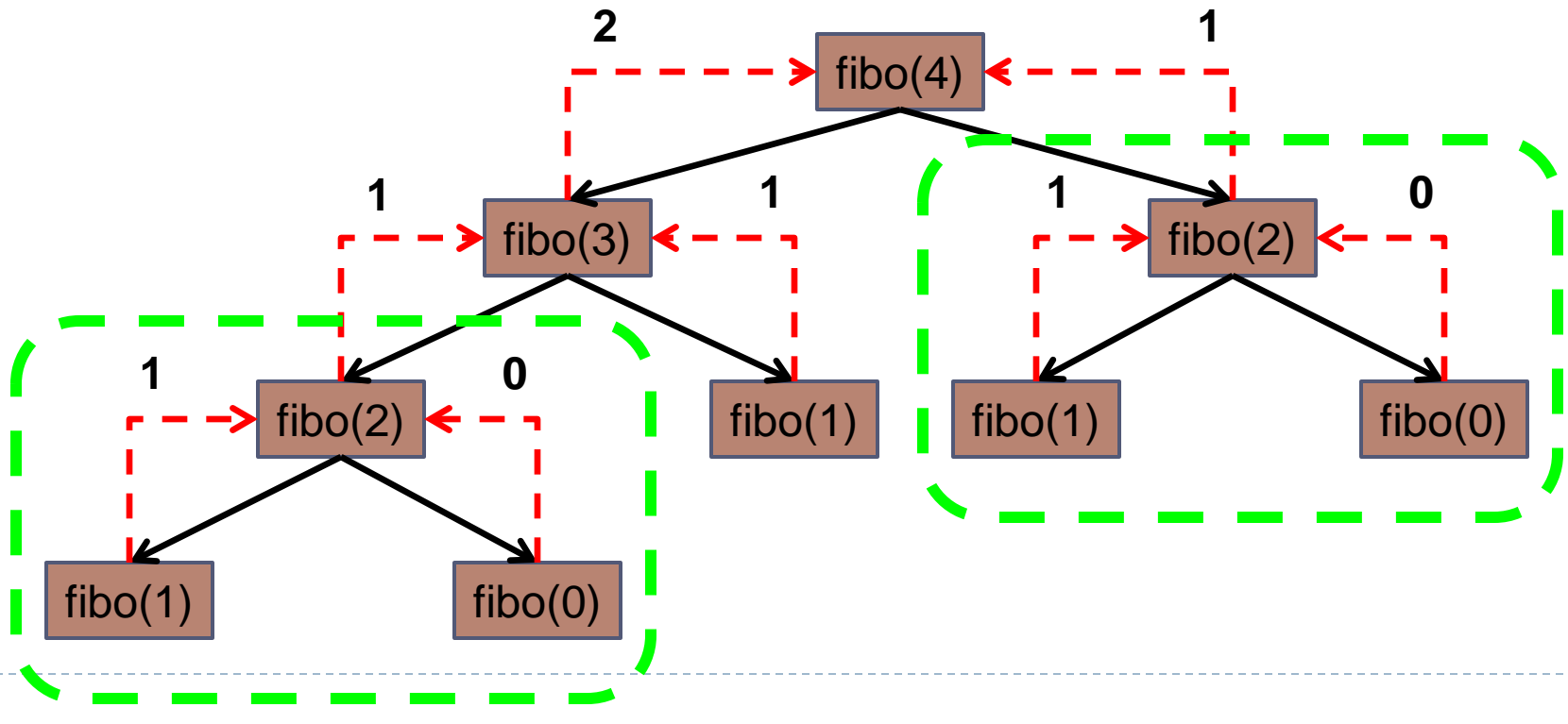
- ▶ Essa seqüência é um clássico da recursão
 - ▶ 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ...
- ▶ Sua solução recursiva é muito elegante ...

```
int fibo(int n){  
    if (n == 0 || n == 1)  
        return n;  
    else  
        return fibo(n-1) + fibo(n-2);  
}
```



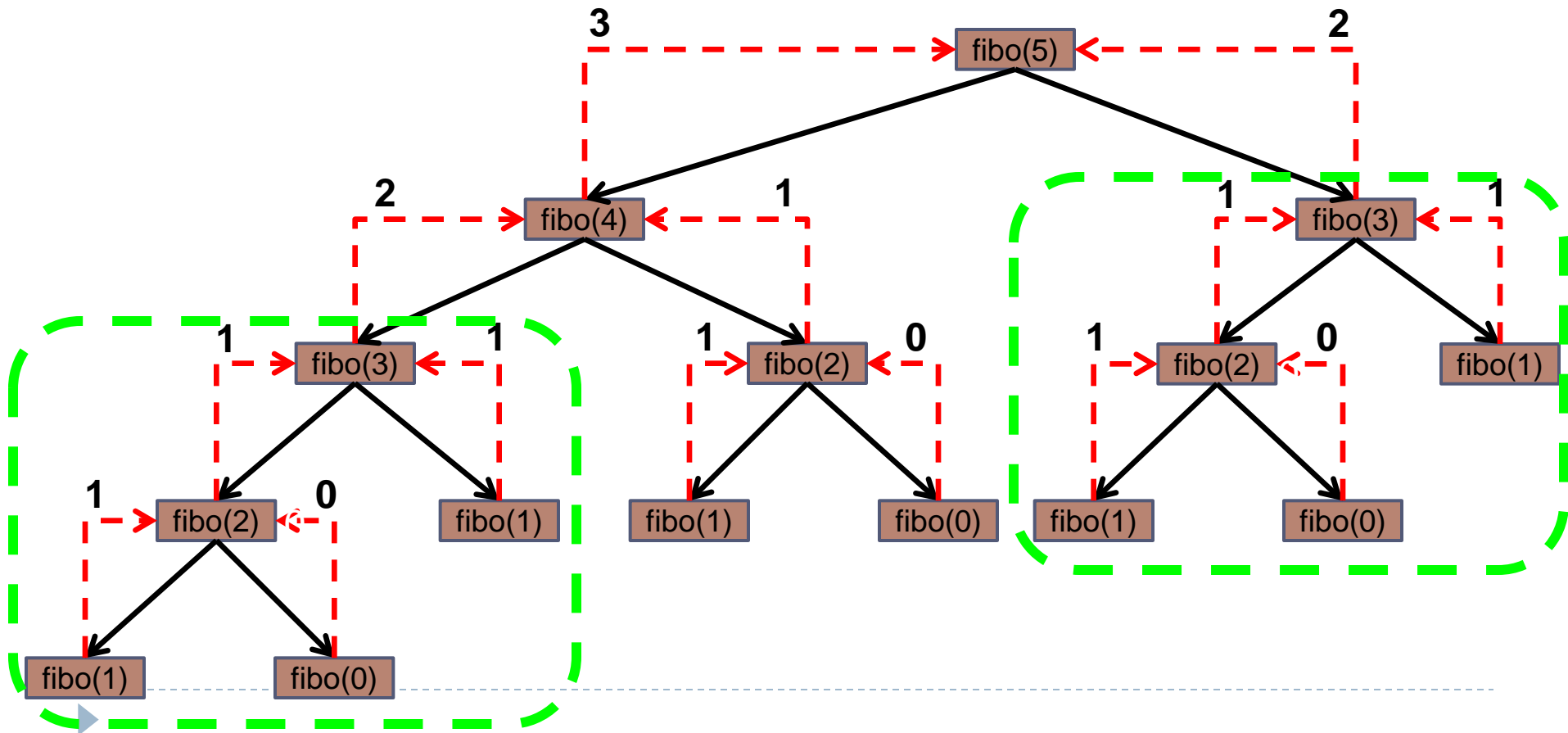
Fibonacci

- ... mas como se verifica na imagem, elegância não significa eficiência



Fibonacci

- ▶ Aumentando para **fibo(5)**



Material Complementar

▶ Vídeo Aulas

- ▶ Aula 43: Função – Visão Geral
- ▶ Aula 44: Função – Parâmetros
- ▶ Aula 45: Função – Corpo
- ▶ Aula 46: Função – Retorno
- ▶ Aula 47: Função – Passagem por Valor
- ▶ Aula 48: Função – Passagem por Referência
- ▶ Aula 49: Função – Array como parâmetro
- ▶ Aula 50: Função – Struct como parâmetro
- ▶ Aula 51: Recursão pt.1 – Definição
- ▶ Aula 52: Recursão pt.2 – Funcionamento
- ▶ Aula 53: Recursão pt.3 – Cuidados
- ▶ Aula 54: Recursão pt.4 – Soma de 1 até N

