Funções

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

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 parte principal.
  - 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 - 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 (1) - 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 (1) - Função sem retorno e sem parâmetro

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();
   ....
```

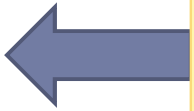
Exemplos (1) - Função sem retorno e sem

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 (1) - 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 (1) - 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();
   ....
```

Função sem parâmetros

Exemplos (2) - Função com retorno e sem parâmetros de entrada

```
// 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 - [illegible]etorno e sem parâmetros

Sem parâmetros. Coloque parênteses na frente do nome da função

```
// 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();
```

...o com retorno e sem ...rada

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

```
// 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 com retorno e sem parâmetros de entrada

```
// 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 com retorno e sem parâmetros de entrada

```
// 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();
```

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

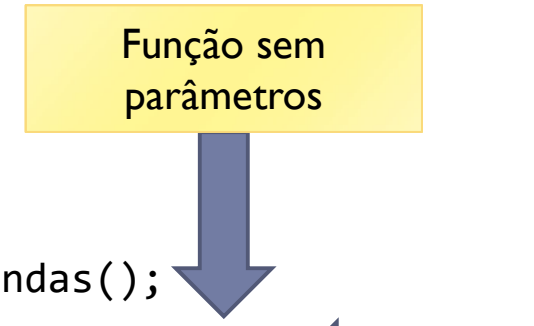
Exemplos - Função com retorno e sem parâmetros de entrada

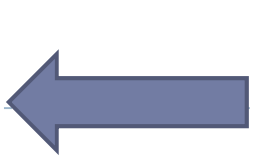
```c
// função sem parâmetros de entrada, mas com retorno
char MenuPrincipal(){

   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 com retorno e sem parâmetros de entrada

```c
// 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 com retorno e sem parâmetros de entrada

```c
// 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

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

   escolha = MenuPrincipal();
```

Chamando a função

Exemplos (3) - Função com retorno e com parâmetros de entrada

```
// 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 (3 com parâme

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

  return aprovado;

}
```

ção com retorno e
e entrada

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

```
// 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;

}
```

ção com retorno e
e entrada

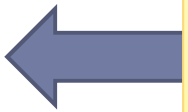
Com retorno. Coloque o tipo de dado que será retornado. Neste exemplo, é um int

```
// 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 (3) - Função com retorno e com parâmetros de entrada

```
// 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!!!");
    }
```

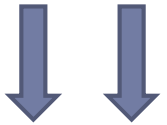


Chamando a função

Exemplos (3) - Função com retorno e com parâmetros de entrada

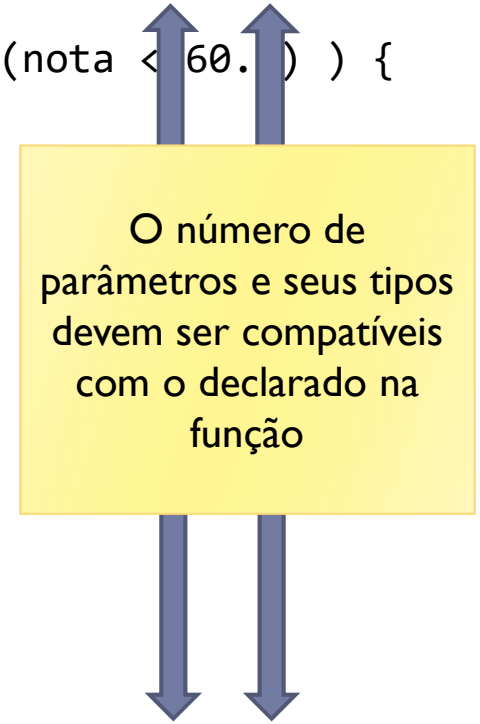
```
// função com retorno e com parâmetros
int VerificaAprovacao(double nota, int faltas) {
   int aprovado  = 1;
   if ( (faltas > 18) || (nota < 60.0) ) {
     aprovado = 0;
   }

  return aprovado;

}



int main()
{
    int ap;

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

Função COM parâmetros



Exemplos (3) - Função com retorno e com parâmetros de entrada

```
// 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!!!");
    }
```



Exemplos (3) - Função com retorno e com parâmetros de entrada

```
// 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

```
ain()

   int ap;

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

Exemplos (3) - Função com retorno e com parâmetros de entrada

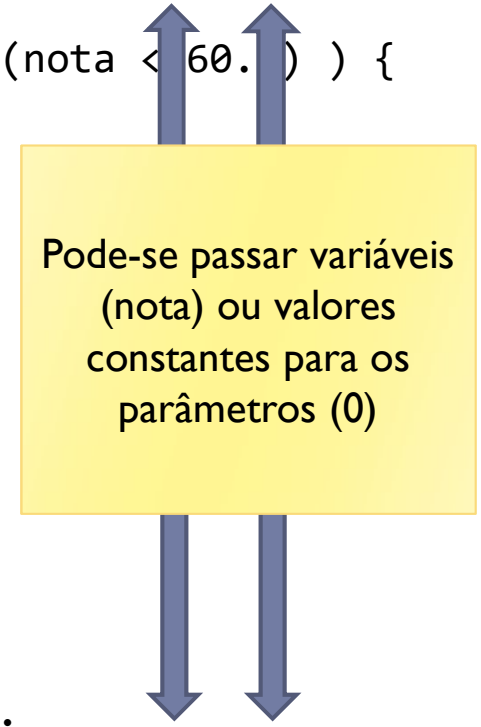
```
// função com retorno e com parâmetros
int VerificaAprovacao(double nota, int faltas) {
   int aprovado  = 1;
   if ( (faltas > 18) || (nota < 60.0) ) {
     aprovado = 0;
   }

  return aprovado;

}
```

Outro exemplo de chamada

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

Exemplos (3) - Função com retorno e com parâmetros de entrada

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



Exemplos (3) - Função com retorno e com parâmetros de entrada

```
// 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 (3) - Função com retorno e com parâmetros de entrada

```
// 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;

}
```

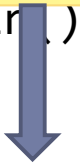
Outro exemplo de chamada

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

Exemplos (3) - Função com retorno e com parâmetros de entrada

```
// 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 (3) - Função com retorno e com parâmetros de entrada

```
// 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;

}
```

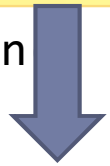
Outro exemplo de chamada

```
int main()
{

    VerificaAprovacao(70.0, 0);
```

Exemplos (3) - Função com retorno e com parâmetros de entrada

```
// 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

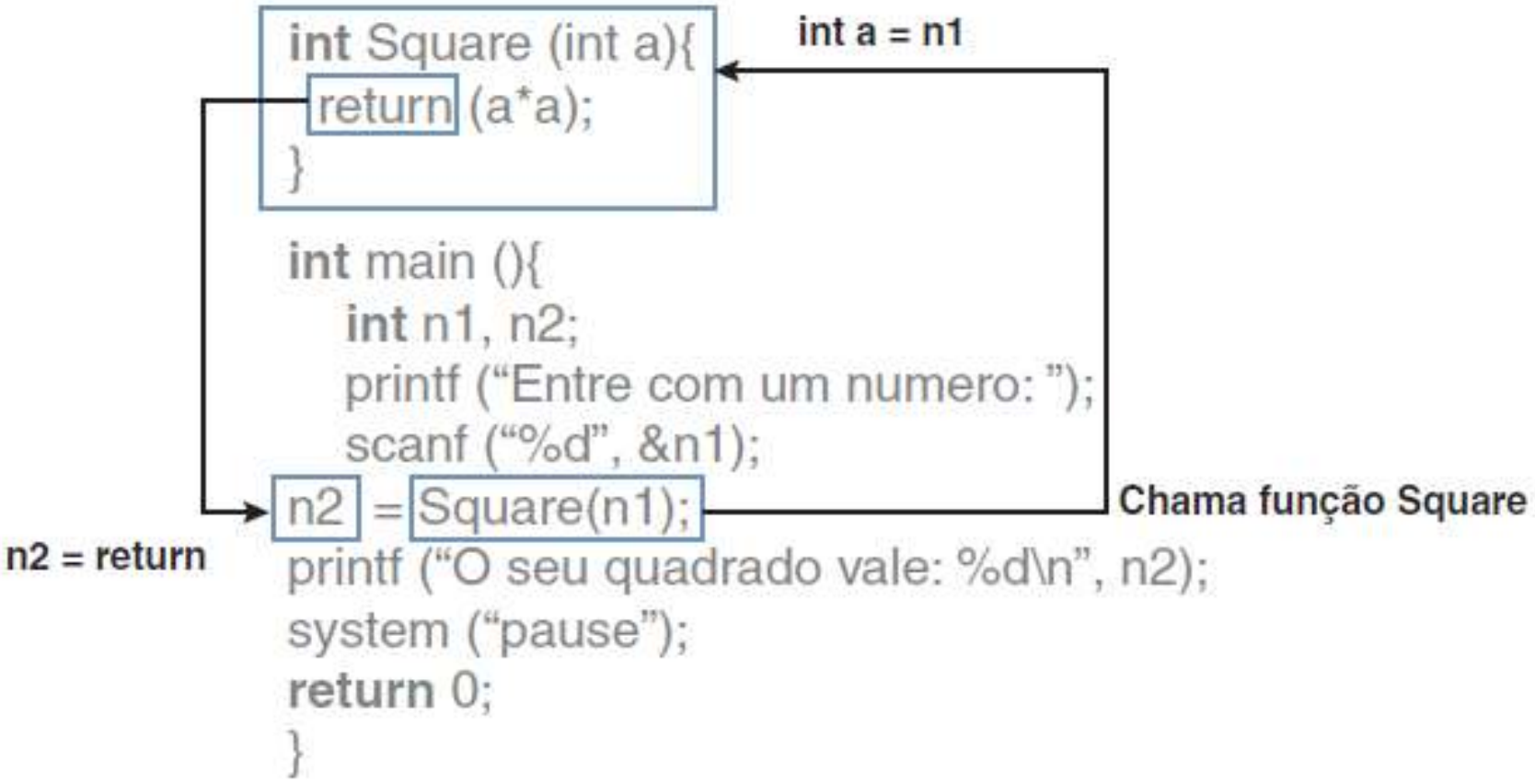
- Funções devem ser definidas ou declaradas antes de serem utilizadas, ou seja, antes da cláusula main.
- A definição (protótipo) apenas indica a existência da função:

  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


int Square (int a){
return (a*a);
}
int a = n1
int main (){
int n1, n2;
printf ("Entre com um numero: ");
scanf ("%d", &n1);
n2 = Square(n1);
Chama função Square
n2 = return
printf ("O seu quadrado vale: %d\n", n2);
system ("pause");
return 0;
}

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  }
```

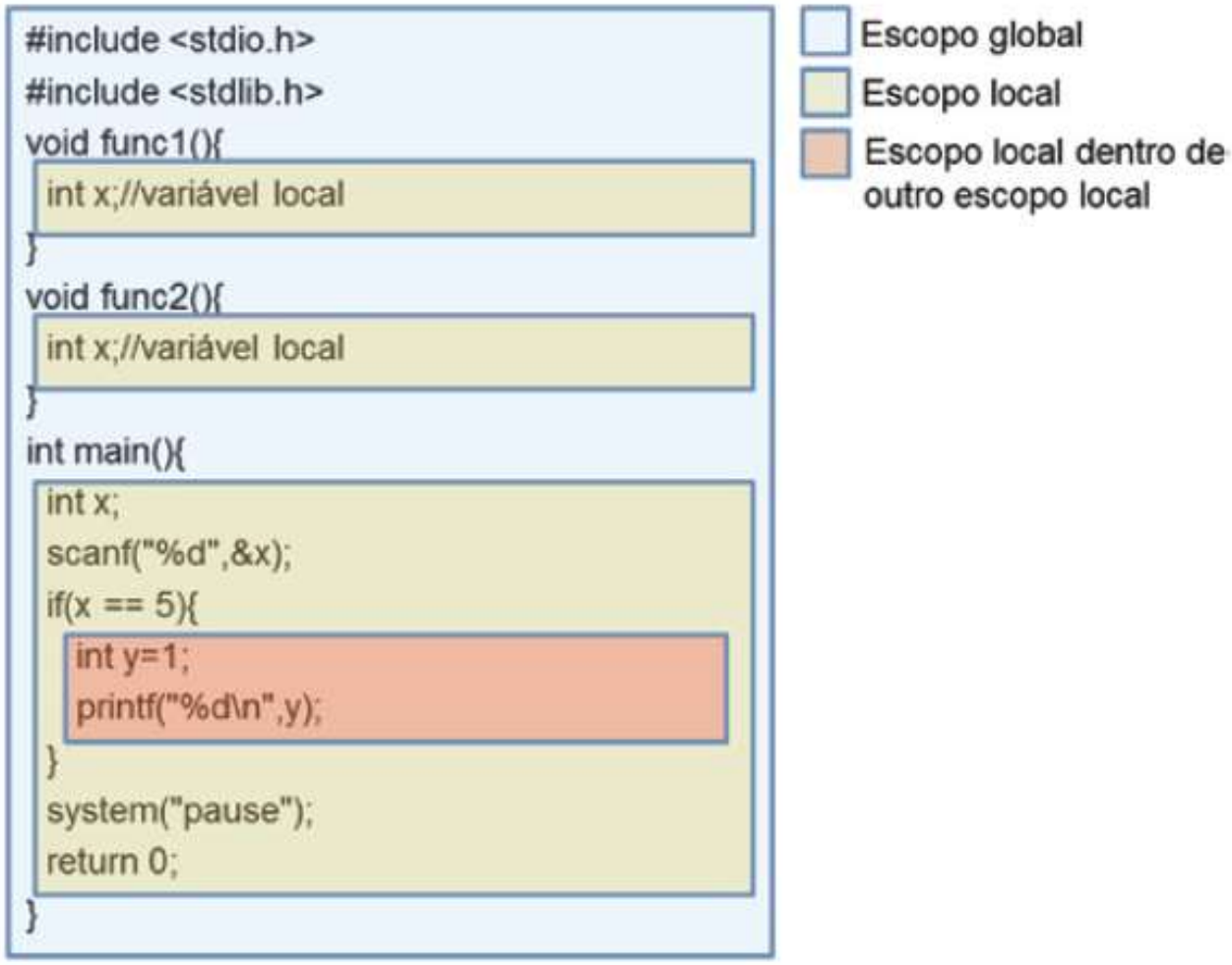
protótipo

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 um

```c
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 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

```c
#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("\nO maior valor entre (%d,%d) eh %d", a,b,m);

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

    return 0;
}
```

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 =
    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

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("Den
}

int main()
{
    int x = 5;
    printf("An

    soma_mais_um(&x);

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

    return 0;
}
```

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

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

- 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);
```

Exemplo: passagem de array como parâmetro

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void imprime (int *n
      , int m){
5   int i;
6   for (i=0; i<m; i++)
7     printf(''%d \n''
          , n[i]);
8 }
9
10 int main (){
11   int v[5] =
        {1,2,3,4,5};
12   imprime(v,5);
13   system(''pause'');
14   return 0;
15 }
```

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.

Exemplo: passagem de matriz como parâmetro

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3
 4 void imprime_matriz(int m[][2], int n){
 5   int i,j;
 6   for (i=0; i<n;i++)
 7     for (j=0; j< 2;j++)
 8       printf(''%d \n'', m[i][j]);
 9 }
10
11 int main (){
12   int mat[3][2] = {{1,2},{3,4},{5,6}};
13   imprime_matriz(mat,3);
14   system(''pause'');
15   return 0;
16 }
```

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);
```

Exemplo: matriz como array de uma dimensão

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void imprime_matriz(int *m, int n){
5   int i;
6   for (i=0; i<n;i++)
7     printf(''%d \n'', m[i]);
8 }
9
10 int main (){
11   int mat[3][2] = {{1,2},{3,4},{5,6}};
12   imprime_matriz(&mat[0][0],6);
13   system(''pause'');
14   return 0;
15 }
```

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

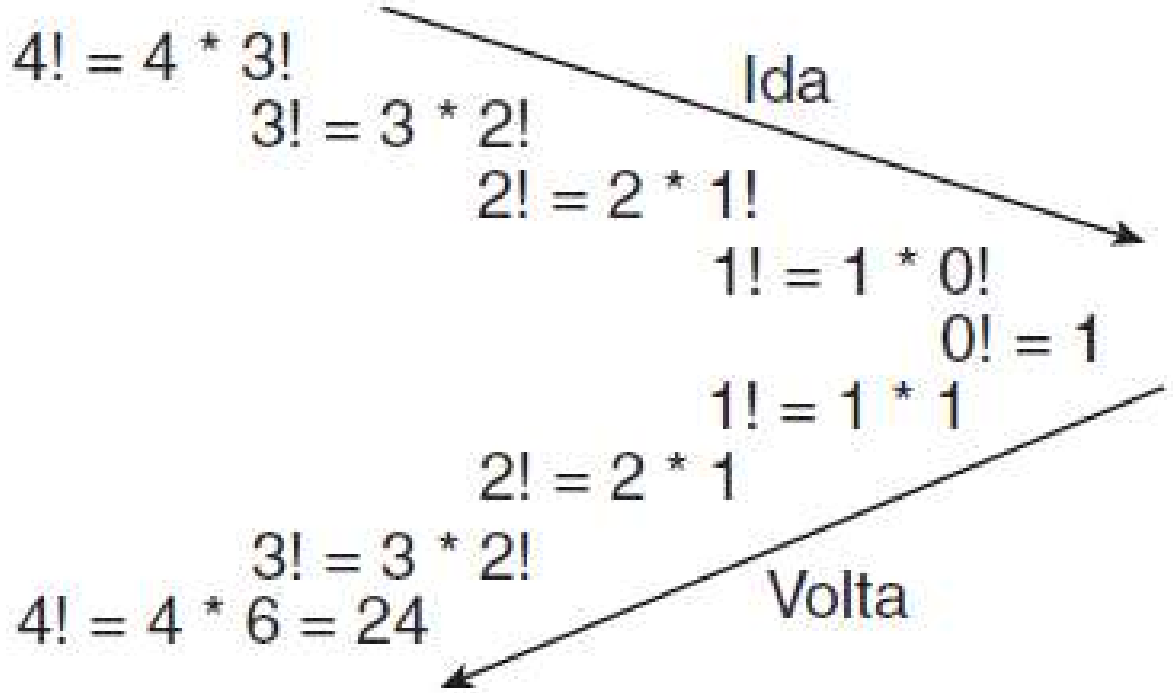
0! = 1

1! = 1 * 0!

2! = 2 * 1!

3! = 3 * 2!

4! = 4 * 3!



n! = n * (n - 1)! : [illegible]

0! = 1 : caso-base

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

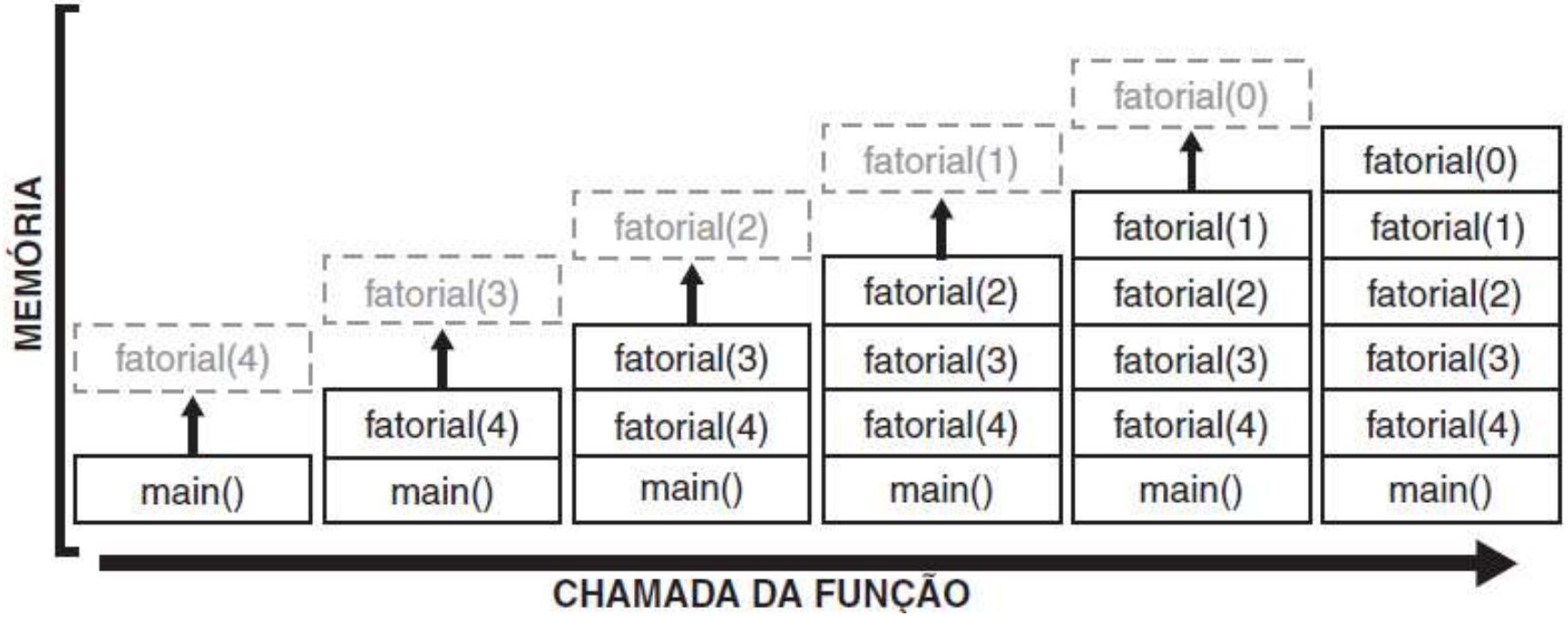
- Todo cuidado é pouco ao se fazer funções recursivas.
  - Critério de parada: determina quando a função deverá parar de chamar a si mesma.
  - O parâmetro da chamada recursiva deve ser sempre modificado, de forma que a recursão chegue a um término.

Recursão

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

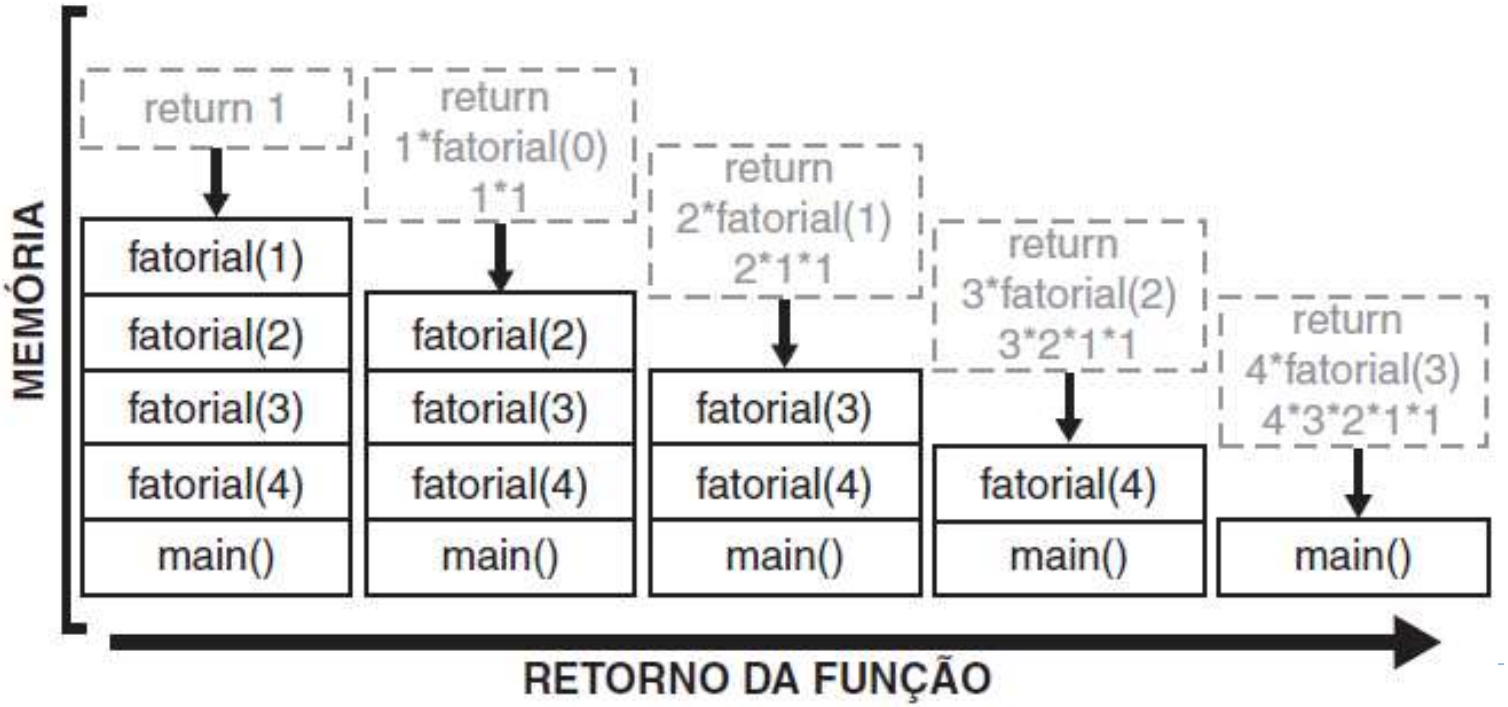
Recursão

- 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.



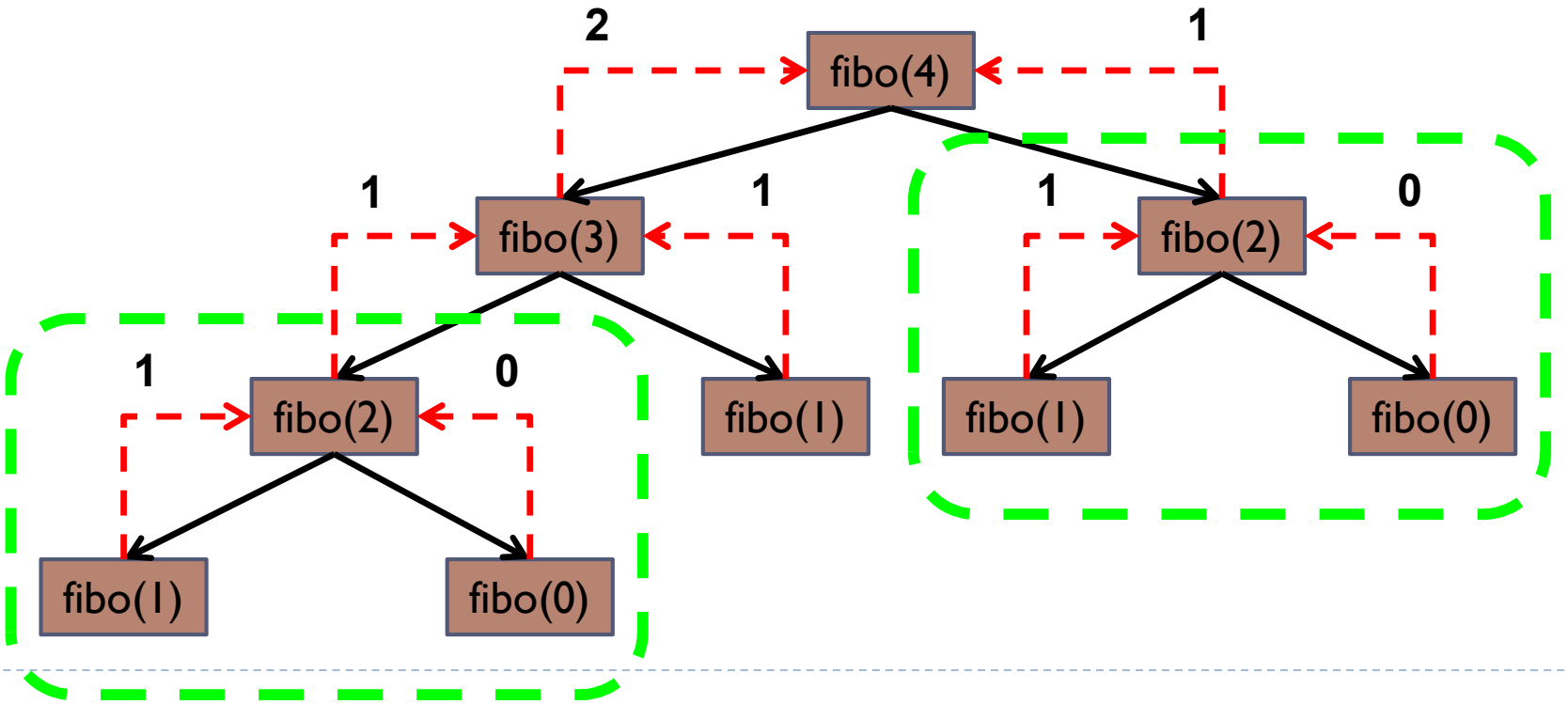
Fibonacci

- 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)**