Introducao a Programacao: Funções, recursividade

periclesmiranda@gmail.com

Plano de aula

Definição: Função

- Bloco de instruções desenvolvido para executar alguma atividade específica;
- A estrutura de uma função C é semelhante à da função `main()`;
- A diferença é que `main()` possui um nome especial e é a primeira a ser chamada (automaticamente) quando o programa é executado;
- Uma função pode ser chamada a partir de outras funções...

Exemplo de função

```
// Funcao que escreve 20 '_' consecutivos
void linha()
{
    int i;
    for(i=0; i<20; ++i)
        printf("_");
    printf("\n");
}

main()
{
    linha(); // Chamando a funcao linha
    printf("Titulo\n");
    linha(); // Chamando a funcao linha
}
```

Passo a passo

```
void linha() // Funcao que escreve 20 '_' a seguir
{
   int i;
   for(i=0; i<20; ++i)
      printf("_");
   printf("\n");
}

main()
{
   linha(); // Chamando a funcao linha
   printf("Titulo\n");
   linha(); // Chamando a funcao linha
}
```

Tela

Passo a passo

```
void linha() // Funcao que escreve 20 '_' a seguir
{
   int i;
   for(i=0; i<20; ++i)
      printf("_");
   printf("\n");
}

main()
{
   linha(); // Chamando a funcao linha
   printf("Titulo\n");
   linha(); // Chamando a funcao linha
}
```

Tela

Passo a passo

```
void linha() // Funcao que escreve 20 '_' a seguir
{
   int i;
   for(i=0; i<20; ++i)
      printf("_");
   printf("\n");
}

main()
{
   linha(); // Chamando a funcao linha
   printf("Titulo\n");
   linha(); // Chamando a funcao linha
}
```

Tela

Passo a passo

```
void linha() // Funcao que escreve 20 '_' a seguir
{
   int i;
   for(i=0; i<20; ++i)
      printf("_");
   printf("\n");
}

main()
{
   linha(); // Chamando a funcao linha
   printf("Titulo\n");
   linha(); // Chamando a funcao linha
}
```

Tela

Passo a passo

```
void linha() // Funcao que escreve 20 '_' a seguir
{
   int i;
   for(i=0; i<20; ++i)
      printf("_");
   printf("\n");
}

main()
{
   linha(); // Chamando a funcao linha
   printf("Titulo\n");
   linha(); // Chamando a funcao linha
}
```

Tela

```
____________________
```

Passo a passo

```
void linha() // Funcao que escreve 20 '_' a seguir
{
   int i;
   for(i=0; i<20; ++i)
      printf("_");
   printf("\n");
}

main()
{
   linha(); // Chamando a funcao linha
   printf("Titulo\n");
   linha(); // Chamando a funcao linha
}
```

Tela

```
____________________
```

Passo a passo

```
void linha() // Funcao que escreve 20 '_' a seguir
{
   int i;
   for(i=0; i<20; ++i)
      printf("_");
   printf("\n");
}

main()
{
   linha(); // Chamando a funcao linha
   printf("Titulo\n");
   linha(); // Chamando a funcao linha
}
```

Tela

```
____________________
Titulo
```

Passo a passo

```
void linha() // Funcao que escreve 20 '_' a seguir
{
   int i;
   for(i=0; i<20; ++i)
      printf("_");
   printf("\n");
}

main()
{
   linha(); // Chamando a funcao linha
   printf("Titulo\n");
   linha(); // Chamando a funcao linha
}
```

Tela

```
____________________
Titulo
```

Passo a passo

```
void linha() // Funcao que escreve 20 '_' a seguir
{
   int i;
   for(i=0; i<20; ++i)
      printf("_");
   printf("\n");
}

main()
{
   linha(); // Chamando a funcao linha
   printf("Titulo\n");
   linha(); // Chamando a funcao linha
}
```

Tela

```
____________________
Titulo
```

Passo a passo

```
void linha() // Funcao que escreve 20 '_' a seguir
{
   int i;
   for(i=0; i<20; ++i)
      printf("_");
   printf("\n");
}

main()
{
   linha(); // Chamando a funcao linha
   printf("Titulo\n");
   linha(); // Chamando a funcao linha
}
```

Tela

```
____________________
Titulo
```

Passo a passo

```
void linha() // Funcao que escreve 20 '_' a seguir
{
   int i;
   for(i=0; i<20; ++i)
      printf("_");
   printf("\n");
}

main()
{
   linha(); // Chamando a funcao linha
   printf("Titulo\n");
   linha(); // Chamando a funcao linha
}
```

Tela

```
____________________
Titulo

____________________
```

Passo a passo

```
void linha() // Funcao que escreve 20 '_' a seguir
{
   int i;
   for(i=0; i<20; ++i)
      printf("_");
   printf("\n");
}

main()
{
   linha(); // Chamando a funcao linha
   printf("Titulo\n");
   linha(); // Chamando a funcao linha
}
```

Tela

```
____________________
Titulo

____________________
```

Passo a passo

```
void linha() // Funcao que escreve 20 '_' a seguir
{
   int i;
   for(i=0; i<20; ++i)
      printf("_");
   printf("\n");
}

main()
{
   linha(); // Chamando a funcao linha
   printf("Titulo\n");
   linha(); // Chamando a funcao linha
}
```

Tela

```
____________________
Titulo

____________________
```

Vantagens

- Evita que se escreva o mesmo código inúmeras vezes em um mesmo programa;
- Modularização do código:
  - Dividir o programa em blocos;
  - Cada bloco executa uma tarefa;
  - Economia de memória;
  - Mais fácil manter;
  - Construção de bibliotecas.

Chamadas de função

- Do mesmo modo que se chama funções da biblioteca C (`printf()`, `sqrt(5)`, ...) é possível chamarmos nossas próprias funções;
- Os parênteses que seguem as funções servem para diferenciar uma chamada de função de uma variável;
- Para algumas funções é necessário fornecer argumentos:
  - valor fixo: `sqrt(5)`;
  - variável: `sqrt(val)`;
  - resultado de outra função: `sqrt(sqrt(5))`;
- Os argumentos são separados por ’,’

Declaração de função

- Uma função deve ser declarada fora de outras funções;
- As funções devem ser conhecidas pelo compilador no momento de sua chamada:
  ⇒Declarar as funções antes das funções que as chamam.

Estrutura do arquivo

1. `includes`
2. `variáveis globais`
3. `função1`
4. `função2`

   `...`
5. `função main`

Sintaxe: Declaração

```
tipo_retorno nome_funcao(parametros)
{
  Declaracao de variaveis
  Sequencia de instrucoes
}
```

- `tipo_retorno`: tipo do resultado (`int`, `char`, ...)
  `void` se não retorna resultado;
- `nome_funcao`: Identificador da função;
- `parâmetros`: Parâmetros de chamada da função (pode ser vazio);
- `Declaração de variáveis`: Variáveis locais da função;
- `Sequência de instruções`: Instruções a serem executadas quando a função for chamada.

Retorno de função

- Usar o comando:
  `return (valor_a_retornar);`
- Instruções após o comando `return` não serão executadas;

Exemplo

A função seguinte vai retornar 0

```
int inicializacao()
{
    int a = 0;
    return (a);
    a = 1;
    return (a);
}
```

Protótipos

`tipo_retorno nome_funcao(parametros)`

é o protótipo da função. Exemplos:

- `void linha()`
- `int lerNumeroPositivo()`
- `void escreveTabuada(int valor)`
- `float distancia(float xa, float ya, float xb, float yb)`

Exercício: Função linha:

Escrever uma função que imprima uma linha de 20 caracteres '*' na tela:

```
void linha()
```

Exercício: Número aleatório

Escrever uma função retorna um número aleatório inteiro de valor de 0 até 50.

```
int aleatorio()
```

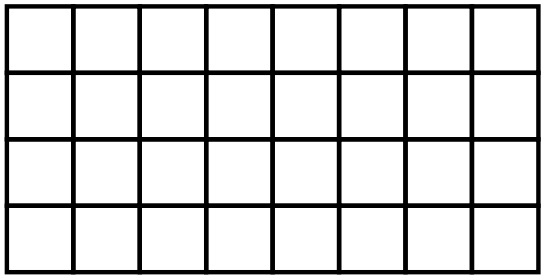
Exercício: Leitura e impressão de um número aleatório

Escrever um programa que imprima um número aleatório inteiro de valor de 0 até 50 entre duas linhas de 20 caracteres '*'.

Variáveis locais

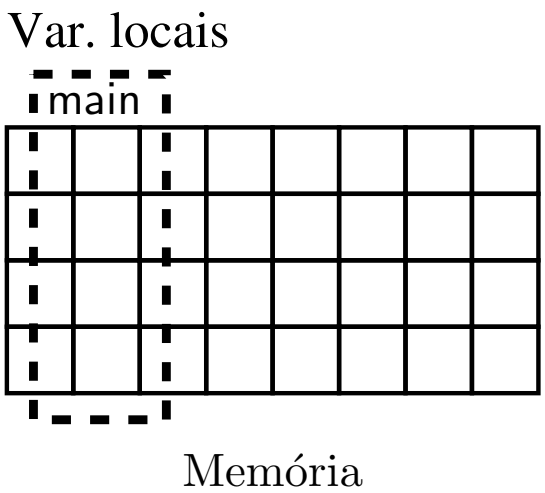
- As variáveis declaradas dentro de uma função são chamadas de variáveis locais e são conhecidas apenas dentro da função;
- Uma variável local existe apenas durante a execução do bloco de código onde ela foi declarada;
- A variável local é criada quando a função for chamada e destruída na saída da função;
- A variável local não é conhecida nas funções chamadas pela função onde ela foi definida;
- Mesmo com o mesmo nome, uma variável declarada fora da função e uma variável declarada dentro da função serão diferentes.

```
void linha()
{
  int i;
  for (i=0; i < 20; i++)
    printf("_");
  printf("\n");
}
main()
{
  int a,b,c;
  linha();
  a = 5;
}
```

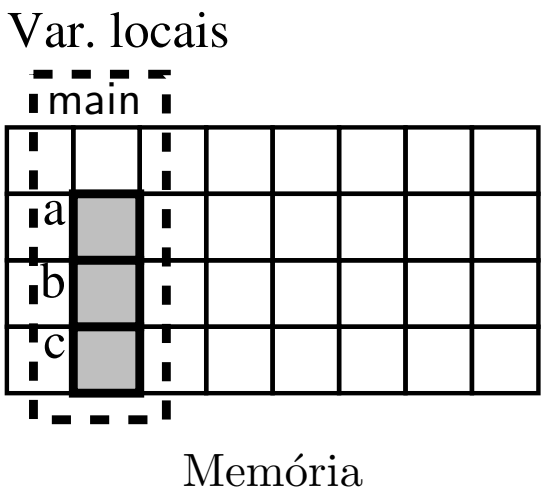


Memória

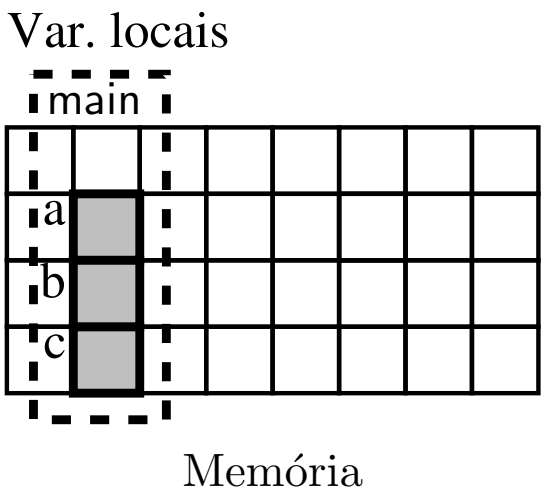
```
void linha()
{
  int i;
  for (i=0; i < 20; i++)
    printf("_");
  printf("\n");
}
main()
{
  int a,b,c;
  linha();
  a = 5;
}
```



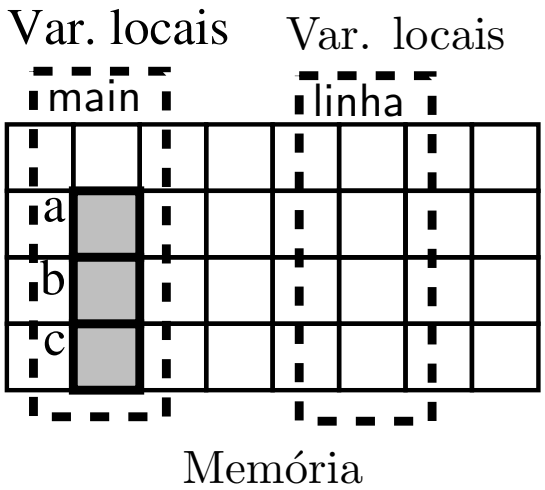
```
void linha()
{
  int i;
  for (i=0; i < 20; i++)
    printf("_");
  printf("\n");
}
main()
{
  int a,b,c;
  linha();
  a = 5;
}
```



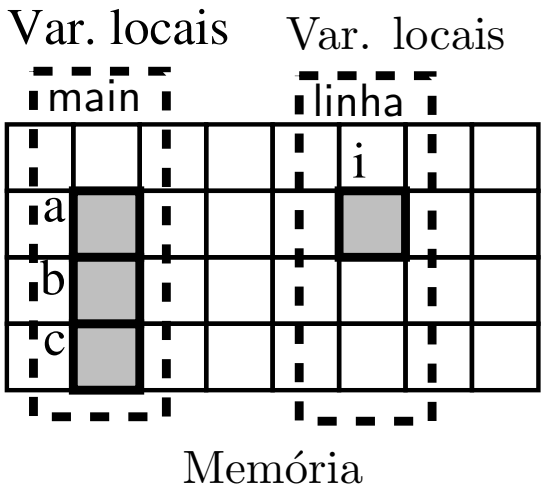
```
void linha()
{
  int i;
  for (i=0; i < 20; i++)
    printf("_");
  printf("\n");
}
main()
{
  int a,b,c;
  linha();
  a = 5;
}
```



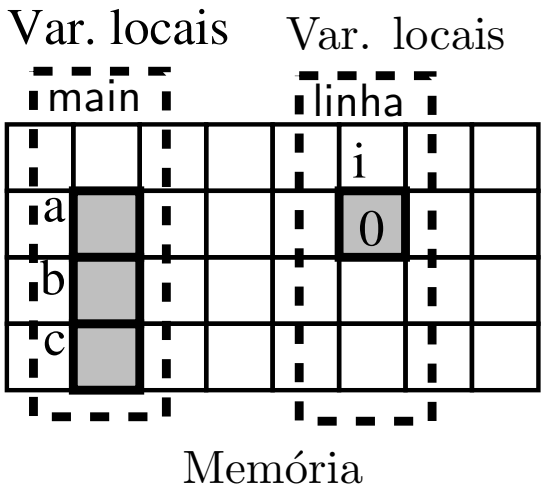
```
void linha()
{
  int i;
  for (i=0; i < 20; i++)
    printf("_");
  printf("\n");
}
main()
{
  int a,b,c;
  linha();
  a = 5;
}
```



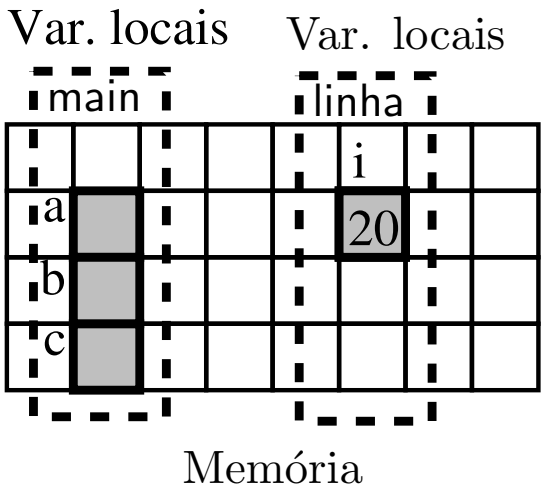
```
void linha()
{
  int i;
  for (i=0; i < 20; i++)
    printf("_");
  printf("\n");
}
main()
{
  int a,b,c;
  linha();
  a = 5;
}
```



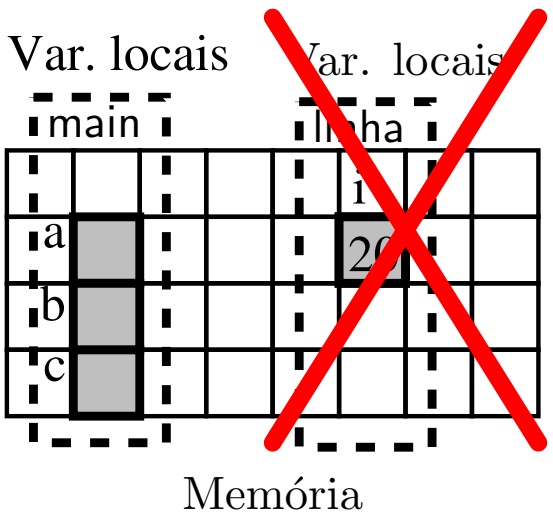
```
void linha()
{
  int i;
  for (i=0; i < 20; i++)
    printf("_");
  printf("\n");
}
main()
{
  int a,b,c;
  linha();
  a = 5;
}
```



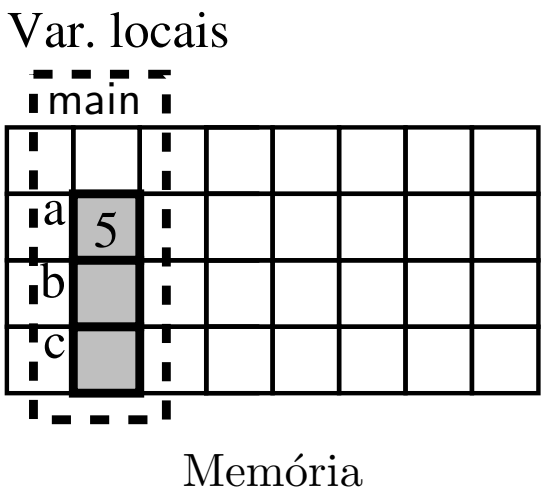
```
void linha()
{
  int i;
  for (i=0; i < 20; i++)
    printf("_");
  printf("\n");
}
main()
{
  int a,b,c;
  linha();
  a = 5;
}
```



```
void linha()
{
  int i;
  for (i=0; i < 20; i++)
    printf("_");
  printf("\n");
}
main()
{
  int a,b,c;
  linha();
  a = 5;
}
```



```
void linha()
{
  int i;
  for (i=0; i < 20; i++)
    printf("_");
  printf("\n");
}
main()
{
  int a,b,c;
  linha();
  a = 5;
}
```



O código seguinte está incorreto: `a` não existe na função `inicializacao`.

```
void inicializacao()
{
    a = 15;
    printf("a inicializada\n");
}

main()
{
    int a;
    inicializacao();
    printf("a=%d\n", a);
}
```

O código seguinte está incorreto: `a` não existe na função `main`.

```
void inicializacao()
{
    int a;
    a = 15;
    printf("a inicializada\n");
}

main()
{
    inicializacao();
    printf("a=%d\n", a);
}
```

Cuidado! As variáveis `a` das funções `main` e `inicializacao` são diferentes.

```
void inicializacao()
{
    int a;
    a = 15;
    printf("a inicializada\n");
}
main()
{
    int a;
    a = 10;
    inicializacao();
    printf("a=%d\n", a);
}
```

```
a inicializada
a=10
```

Exercício: Função lerNumeroPositivo

Escrever uma função que fique pedindo ao usuário um número inteiro. Se o número for positivo finalizar e retornar o número.

```
int lerNumeroPositivo()
```

Exercício: Estatística

Escrever uma função que peça ao usuário 10 valores reais, e que imprima na tela a média dos valores, o maior e o menor número.

```
void estatistica()
```

Exercício: Estatísticas grupos

Escrever um programa que peça ao usuário uma quantidade de grupos de 10 números, e que para cada grupo imprima a média dos valores, o maior e o menor número. O programa deve garantir que a quantidade de grupos é válida.

Definição: variáveis globais

- É possível definir variáveis conhecidas de todo mundo;
- São chamadas variáveis globais;
- Declaração no cabeçalho do programa;
- Uso não aconselhado:
  - Ocupam memória;
  - Todo mundo pode modificar o valor!

```c
int a;
void inicializacao()
{
    a = 15;
    printf("a inicializada\n");
}
main()
{
    a = 10;
    inicializacao();
    printf("a=%d\n", a);
}
```

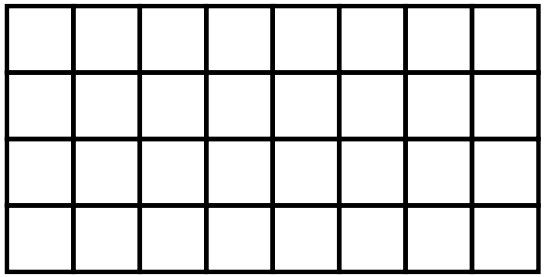
```
a inicializada
a=15
```

Cuidado! As variáveis `a` globais e locais (da função `main`) são diferentes.

```
int a;
void inicializacao()
{
   a = 15;
   printf("a inicializada\n");
}
main()
{
   int a;
   a = 10;
   inicializacao();
   printf("a=%d\n", a);
}
```

```
a inicializada
a=10
```

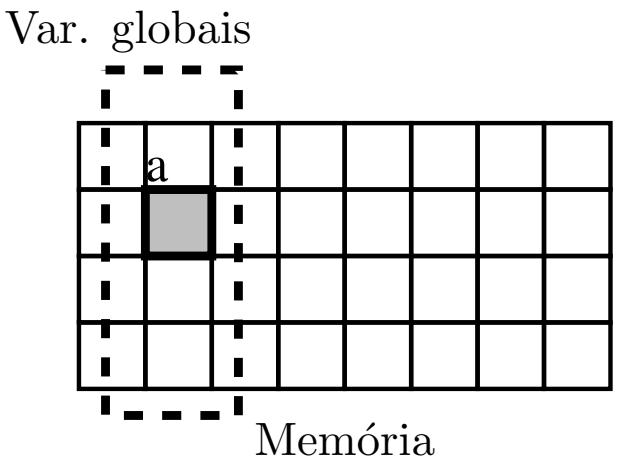
```
int a;
void inicializacao()
{
   a = 15;
   printf("a inicializada\n");
}
main()
{
   int a;
   a = 10;
   inicializacao();
   printf("a=%d\n", a);
}
```



Memória

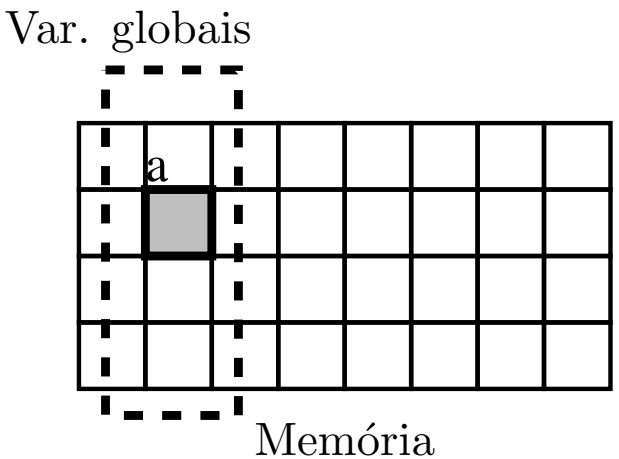
Tela

```
int a;
void inicializacao()
{
   a = 15;
   printf("a inicializada\n");
}
main()
{
   int a;
   a = 10;
   inicializacao();
   printf("a=%d\n", a);
}
```



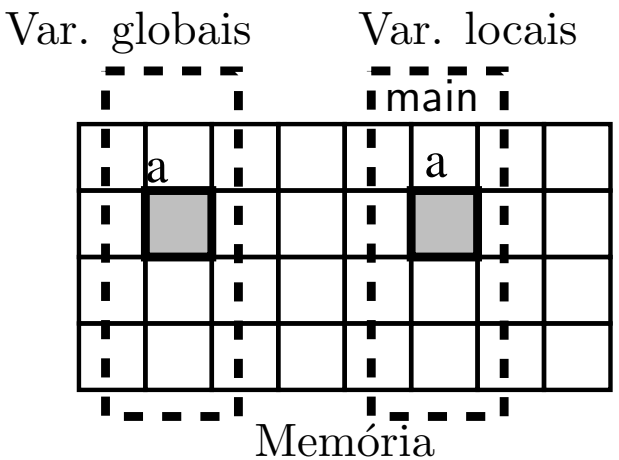
Tela

```
int a;
void inicializacao()
{
   a = 15;
   printf("a inicializada\n");
}
main()
{
   int a;
   a = 10;
   inicializacao();
   printf("a=%d\n", a);
}
```



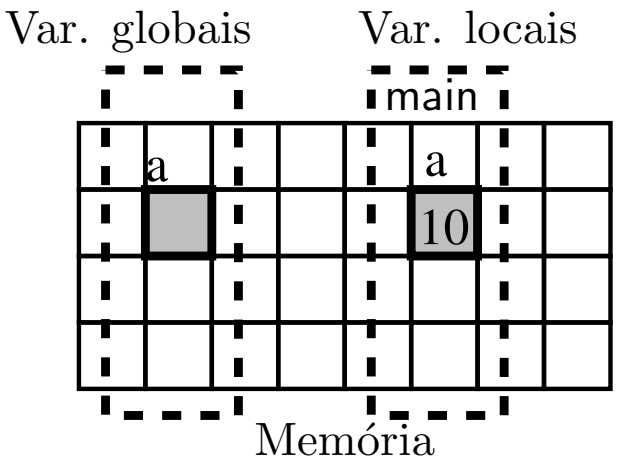
Tela

```
int a;
void inicializacao()
{
   a = 15;
   printf("a inicializada\n");
}
main()
{
   int a;
   a = 10;
   inicializacao();
   printf("a=%d\n", a);
}
```



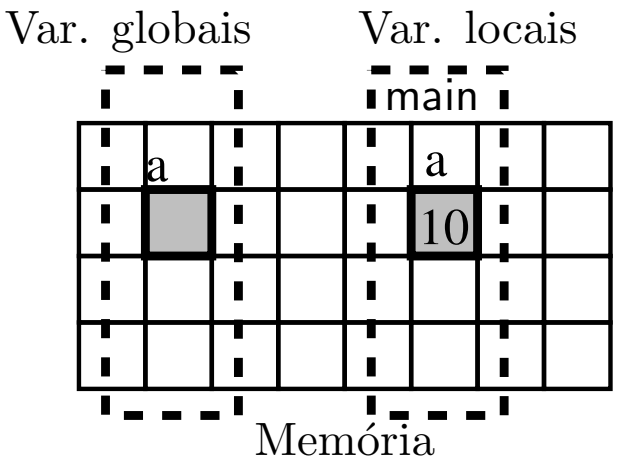
Tela

```
int a;
void inicializacao()
{
   a = 15;
   printf("a inicializada\n");
}
main()
{
   int a;
   a = 10;
   inicializacao();
   printf("a=%d\n", a);
}
```



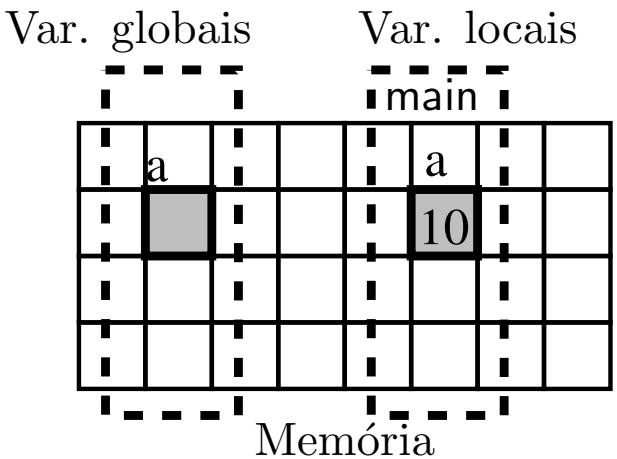
Tela

```
int a;
void inicializacao()
{
   a = 15;
   printf("a inicializada\n");
}
main()
{
   int a;
   a = 10;
   inicializacao();
   printf("a=%d\n", a);
}
```



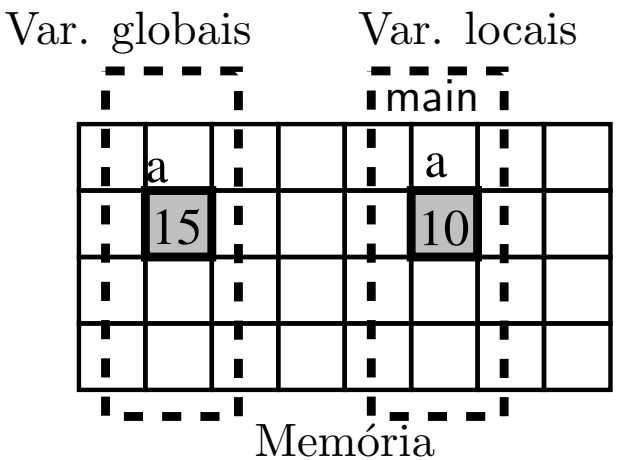
Tela

```
int a;
void inicializacao()
{
   a = 15;
   printf("a inicializada\n");
}
main()
{
   int a;
   a = 10;
   inicializacao();
   printf("a=%d\n", a);
}
```



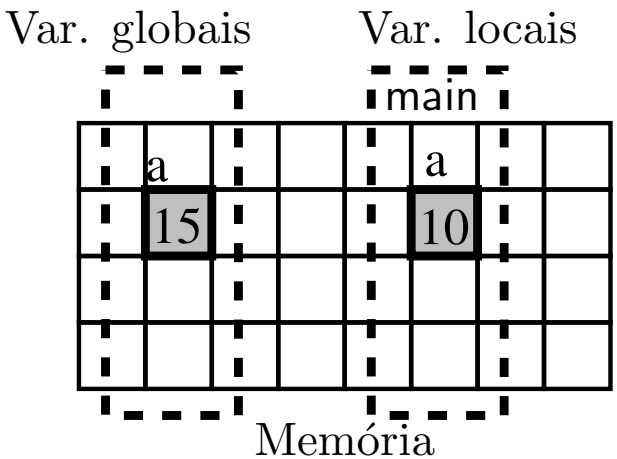
Tela

```
int a;
void inicializacao()
{
   a = 15;
   printf("a inicializada\n");
}
main()
{
   int a;
   a = 10;
   inicializacao();
   printf("a=%d\n", a);
}
```



Tela

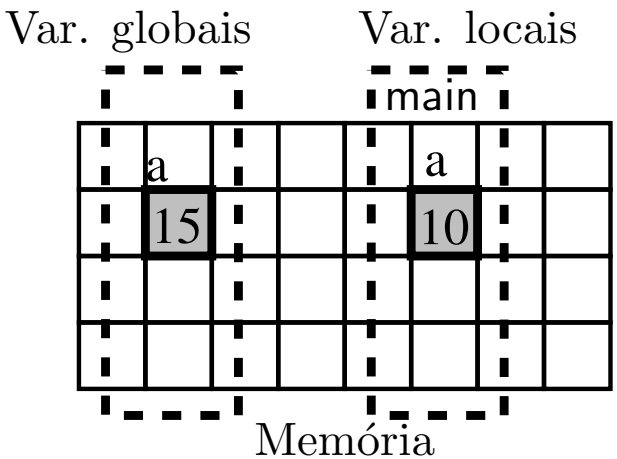
```
int a;
void inicializacao()
{
   a = 15;
   printf("a inicializada\n");
}
main()
{
   int a;
   a = 10;
   inicializacao();
   printf("a=%d\n", a);
}
```



Tela

```
a inicializada
```

```
int a;
void inicializacao()
{
   a = 15;
   printf("a inicializada\n");
}
main()
{
   int a;
   a = 10;
   inicializacao();
   printf("a=%d\n", a);
}
```



Tela

```
a inicializada
a=10
```

Exercício: Inicialização de matriz

Escrever uma função que inicialize uma matriz `a` de 6x6 inteiros com números aleatórios entre 0 e 99.

```
void inicializacao()
```

Exercício: Impressão de matriz

Escrever uma função que imprima uma matriz `a` de 6x6 inteiros.

```
void impressao()
```

Exercício: Matriz

Escrever um programa que inicialize e imprima uma matriz 6x6.

Definição: Parâmetros

- Conjunto de variáveis utilizadas para a função receber os valores para os quais a função deve ser executada;
- Estes valores são chamados argumentos, que constituem os dados de entrada da função;
- Na declaração da função, devemos declarar os parâmetros e o tipo deles.

Exemplo

Para escrever uma tabuada de um inteiro qualquer, a função deve saber para qual inteiro imprimir a tabuada.

```
void escreveTabuada(int num)
```

Sintaxe: Parâmetros

```
tipo nomeProc(tipo1 var1, tipo2 var2 ...);
```

Observação

- Uma função pode ter mais que um parâmetro;
- Os tipos dos parâmetros podem ser diferentes;

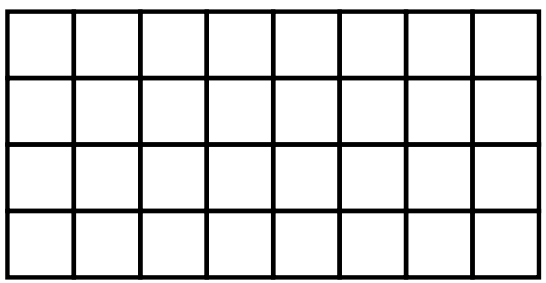
Exemplo: Parâmetros

```
void escreveTabuada(int num)
```

- Na função `escreveTabuada`, temos acesso:
  - Às variáveis globais;
  - Às variáveis locais da função `escreveTabuada`;
  - À cópia do valor do parâmetro usando o nome `num`.
- A chamada será:

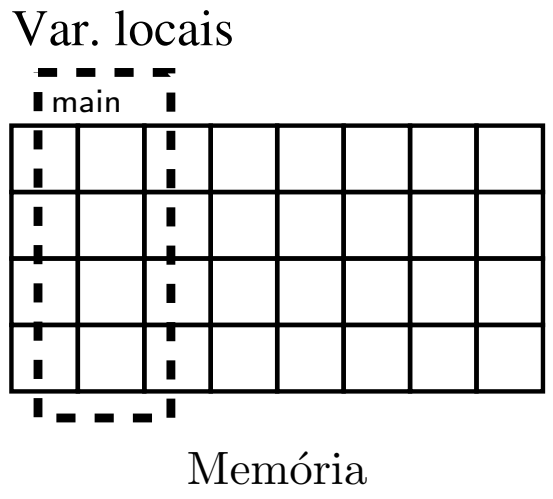
```
escreveTabuada(5);
```

```
void escreveTabuada(int num)
{
  int i;
  for (i=1; i<=10; i++)
    printf("%d x %d=%d\n",num,i,num*i);
  printf("\n");
}

main()
{
  int i;
  for (i=1; i<=10; i++)
    escreveTabuada(i);
}
```

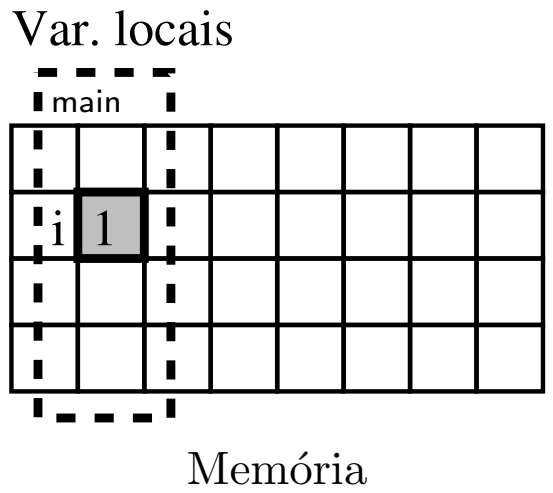


Memória

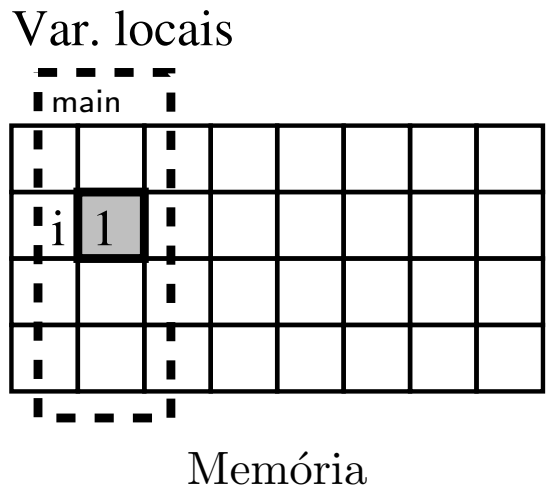
```
void escreveTabuada(int num)
{
  int i;
  for (i=1; i<=10; i++)
    printf("%d x %d=%d\n",num,i,num*i);
  printf("\n");
}

main()
{
  int i;
  for (i=1; i<=10; i++)
    escreveTabuada(i);
}
```



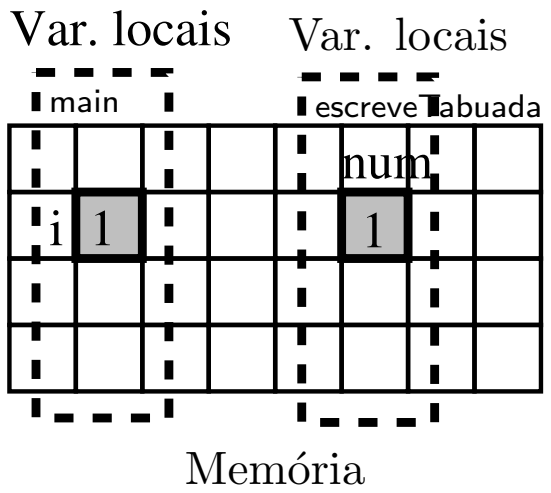
```
void escreveTabuada(int num)
{
  int i;
  for (i=1; i<=10; i++)
    printf("%d x %d=%d\n",num,i,num*i);
  printf("\n");
}

main()
{
  int i;
  for (i=1; i<=10; i++)
    escreveTabuada(i);
}
```



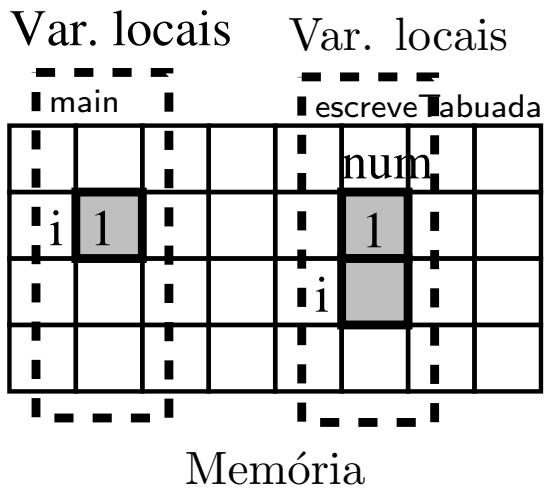
```
void escreveTabuada(int num)
{
  int i;
  for (i=1; i<=10; i++)
    printf("%d x %d=%d\n",num,i,num*i);
  printf("\n");
}

main()
{
  int i;
  for (i=1; i<=10; i++)
    escreveTabuada(i);
}
```



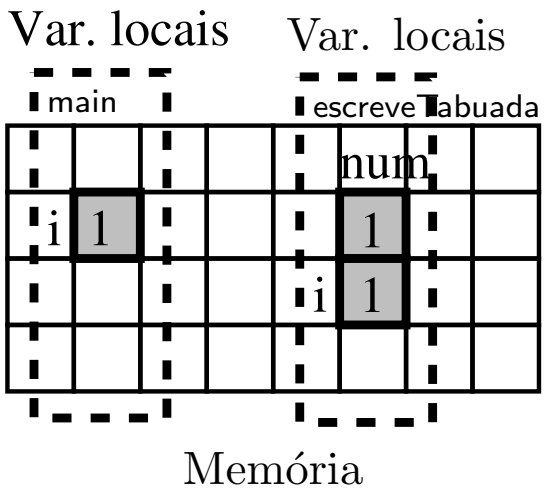
```
void escreveTabuada(int num)
{
  int i;
  for (i=1; i<=10; i++)
    printf("%d x %d=%d\n",num,i,num*i);
  printf("\n");
}

main()
{
  int i;
  for (i=1; i<=10; i++)
    escreveTabuada(i);
}
```



```
void escreveTabuada(int num)
{
  int i;
  for (i=1; i<=10; i++)
    printf("%d x %d=%d\n",num,i,num*i);
  printf("\n");
}

main()
{
  int i;
  for (i=1; i<=10; i++)
    escreveTabuada(i);
}
```

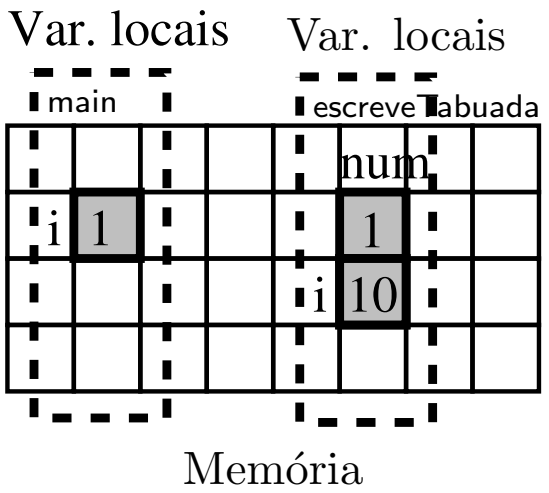


```
void escreveTabuada(int num)
{
  int i;
  for (i=1; i<=10; i++)
    printf("%d x %d=%d\n",num,i,num*i);
  printf("\n");
}

main()
{
  int i;
  for (i=1; i<=10; i++)
    escreveTabuada(i);
}
```

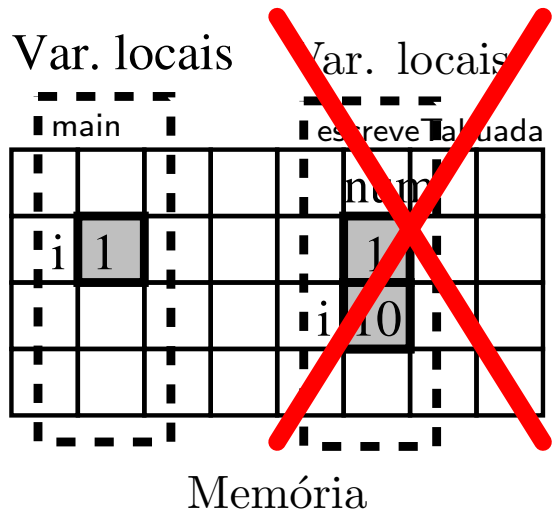


Memória

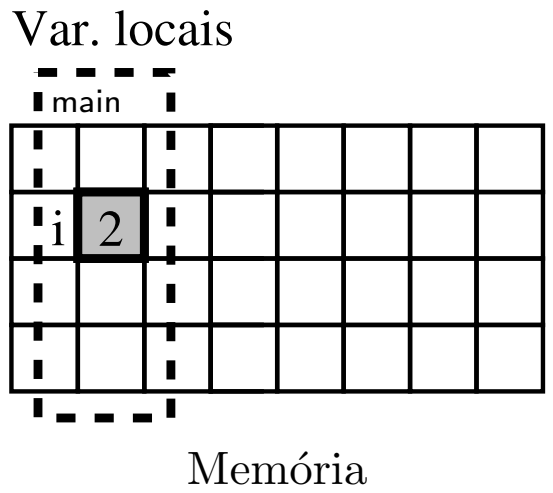
```
void escreveTabuada(int num)
{
  int i;
  for (i=1; i<=10; i++)
    printf("%d x %d=%d\n",num,i,num*i);
  printf("\n");
}

main()
{
  int i;
  for (i=1; i<=10; i++)
    escreveTabuada(i);
}
```



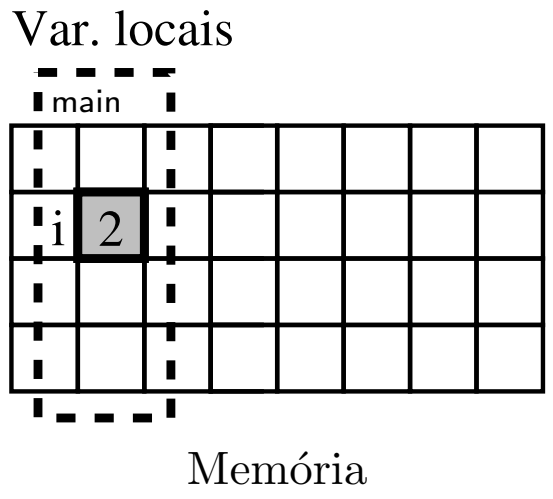
```
void escreveTabuada(int num)
{
  int i;
  for (i=1; i<=10; i++)
    printf("%d x %d=%d\n",num,i,num*i);
  printf("\n");
}

main()
{
  int i;
  for (i=1; i<=10; i++)
    escreveTabuada(i);
}
```



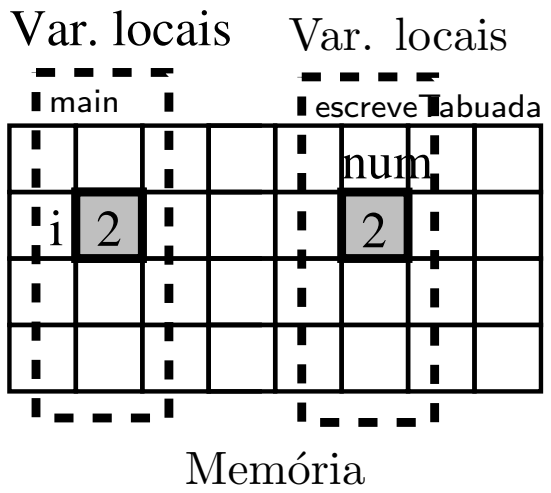
```
void escreveTabuada(int num)
{
  int i;
  for (i=1; i<=10; i++)
    printf("%d x %d=%d\n",num,i,num*i);
  printf("\n");
}

main()
{
  int i;
  for (i=1; i<=10; i++)
    escreveTabuada(i);
}
```



```
void escreveTabuada(int num)
{
  int i;
  for (i=1; i<=10; i++)
    printf("%d x %d=%d\n",num,i,num*i);
  printf("\n");
}

main()
{
  int i;
  for (i=1; i<=10; i++)
    escreveTabuada(i);
}
```



```
void escreveTabuada(int num)
{
  int i;
  for (i=1; i<=10; i++)
    printf("%d x %d=%d\n",num,i,num*i);
  printf("\n");
}

main()
{
  int i;
  for (i=1; i<=10; i++)
    escreveTabuada(i);
}
```



Exercício: Distância entre dois pontos

Escrever a função que retorna a distância entre os pontos de coordenados `(xa,ya)` e `(xb,yb)`:

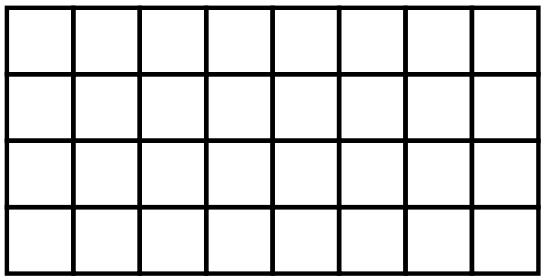
```
float distancia(float xa, float ya,
                float xb, float yb)
```

Escrever um programa em C que peça ao usuário dois pontos $A = (x_a, y_a)$ e $B = (x_b, y_b)$ e que imprima na tela a distância entre A e B.

Passagem de parâmetro por Valor

- A função trabalha com uma cópia dos parâmetros!
- Ou seja, toda alteração dos parâmetros dentro da função será perdida!
- Dizemos que os parâmetros foram passados por valor.

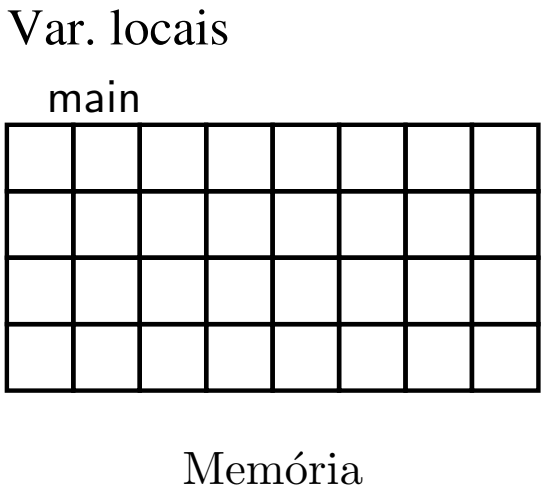
```
void troca(int a, int b)
{
  int temp;
  temp = a;
  a = b;
  b = temp;
}
main()
{
   int i=10; int j=5;
   troca(i,j);
   printf("i=%d j=%d\n",i,j);
}
```



Memória

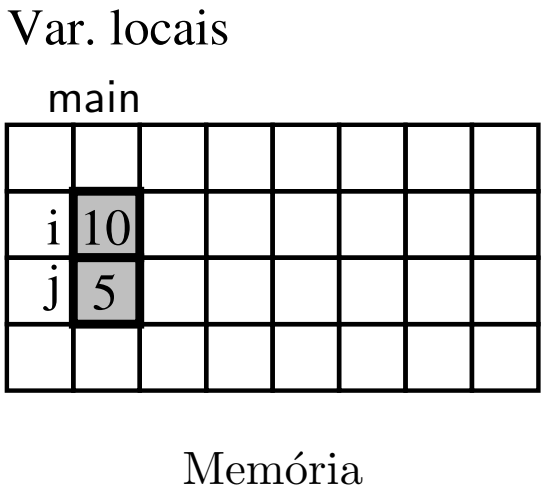
Tela

```
void troca(int a, int b)
{
  int temp;
  temp = a;
  a = b;
  b = temp;
}
main()
{
   int i=10; int j=5;
   troca(i,j);
   printf("i=%d j=%d\n",i,j);
}
```



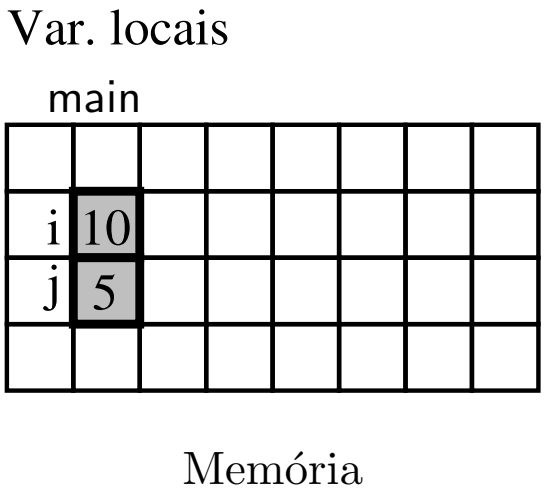
Tela

```
void troca(int a, int b)
{
  int temp;
  temp = a;
  a = b;
  b = temp;
}
main()
{
   int i=10; int j=5;
   troca(i,j);
   printf("i=%d j=%d\n",i,j);
}
```



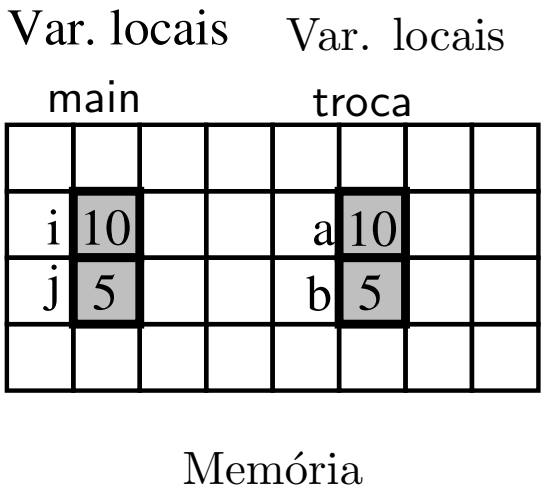
Tela

```
void troca(int a, int b)
{
  int temp;
  temp = a;
  a = b;
  b = temp;
}
main()
{
   int i=10; int j=5;
   troca(i,j);
   printf("i=%d j=%d\n",i,j);
}
```



Tela

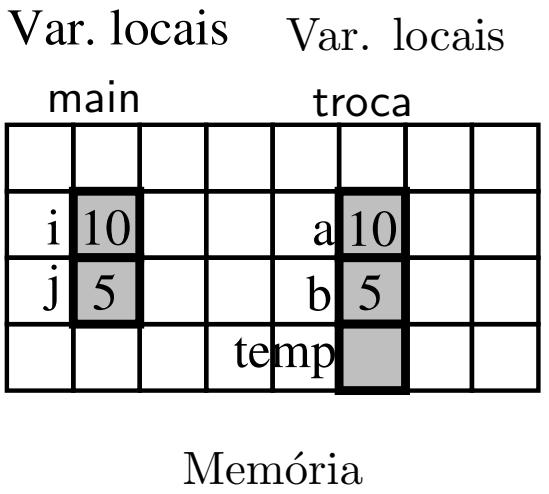
```
void troca(int a, int b)
{
  int temp;
  temp = a;
  a = b;
  b = temp;
}
main()
{
   int i=10; int j=5;
   troca(i,j);
   printf("i=%d j=%d\n",i,j);
}
```



Memória

Tela

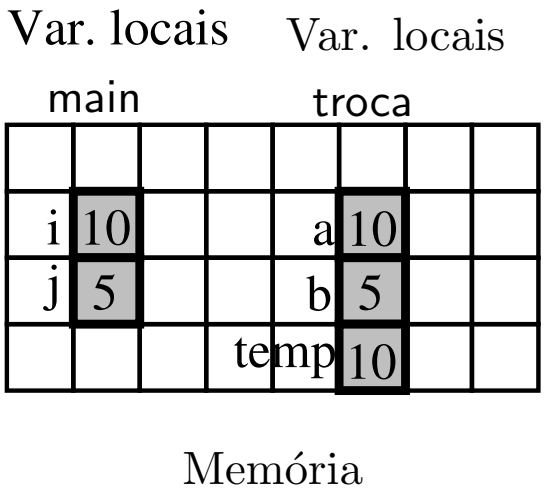
```
void troca(int a, int b)
{
  int temp;
  temp = a;
  a = b;
  b = temp;
}
main()
{
   int i=10; int j=5;
   troca(i,j);
   printf("i=%d j=%d\n",i,j);
}
```



Memória

Tela

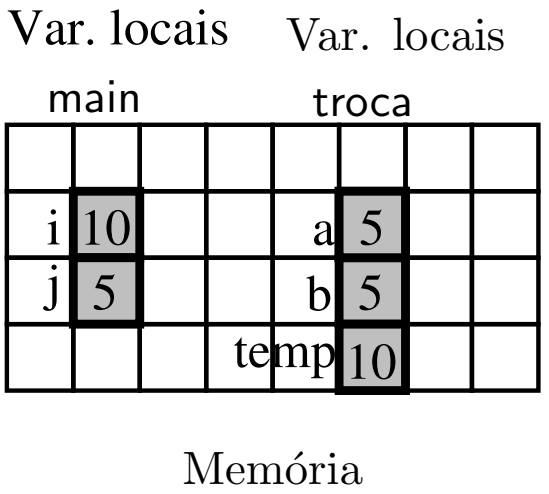
```
void troca(int a, int b)
{
  int temp;
  temp = a;
  a = b;
  b = temp;
}
main()
{
   int i=10; int j=5;
   troca(i,j);
   printf("i=%d j=%d\n",i,j);
}
```



Memória

Tela

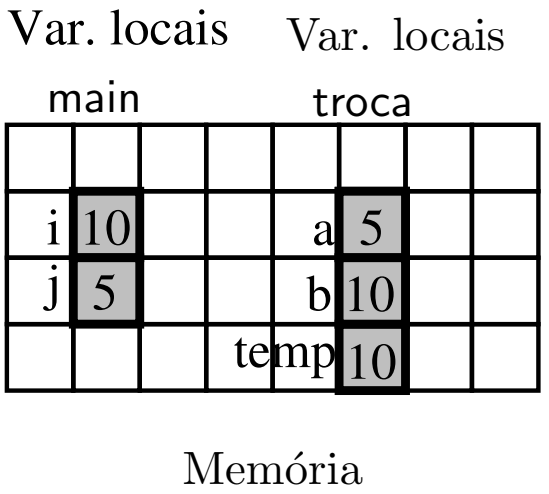
```
void troca(int a, int b)
{
  int temp;
  temp = a;
  a = b;
  b = temp;
}
main()
{
   int i=10; int j=5;
   troca(i,j);
   printf("i=%d j=%d\n",i,j);
}
```



Memória

Tela

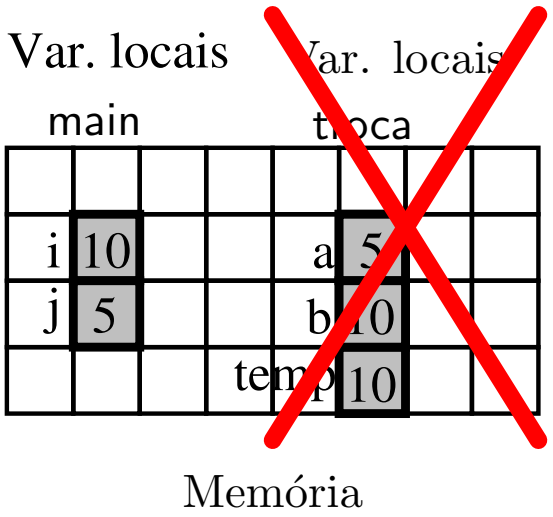
```
void troca(int a, int b)
{
  int temp;
  temp = a;
  a = b;
  b = temp;
}
main()
{
   int i=10; int j=5;
   troca(i,j);
   printf("i=%d j=%d\n",i,j);
}
```



Memória

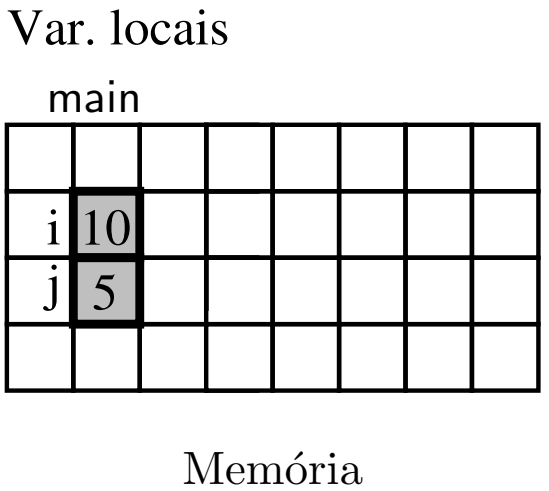
Tela

```
void troca(int a, int b)
{
  int temp;
  temp = a;
  a = b;
  b = temp;
}
main()
{
   int i=10; int j=5;
   troca(i,j);
   printf("i=%d j=%d\n",i,j);
}
```



Tela

```
void troca(int a, int b)
{
  int temp;
  temp = a;
  a = b;
  b = temp;
}
main()
{
   int i=10; int j=5;
   troca(i,j);
   printf("i=%d j=%d\n",i,j);
}
```



Tela

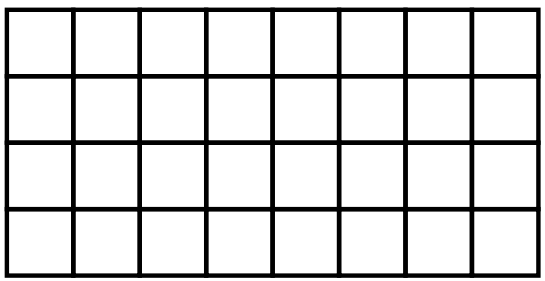
```
i=10 j=5
```

Passagem de parâmetro por referência

- A função trabalha diretamente com os parâmetros!
- Ou seja, toda alteração dos parâmetros dentro da função será mantida!
- Dizemos que os parâmetros foram passados por referência.

```
void trocaValor(int *a, int *b)
{
  int temp;
  temp = *a;
  *a = *b;
  *b = temp;
}
void main()
{
  int i = 10; int j = 5;
  trocaValor(&i, &j);
}
```

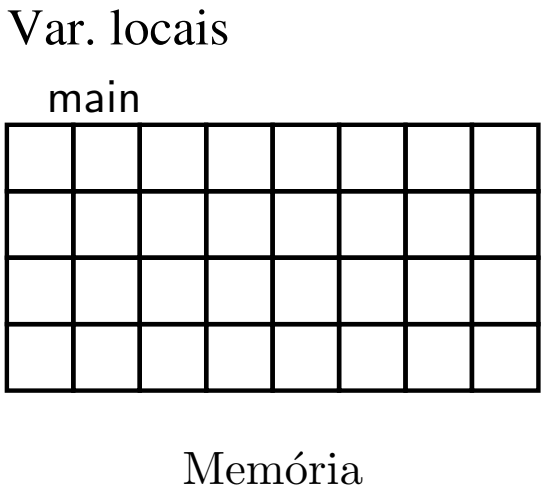
```
void troca(int *a, int *b)
{
  int temp;
  temp = *a;
  *a = *b;
  *b = temp;
}
main()
{
   int i=10; int j=5;
   troca(&i,&j);
   printf("i=%d j=%d\n",i,j);
}
```



Memória

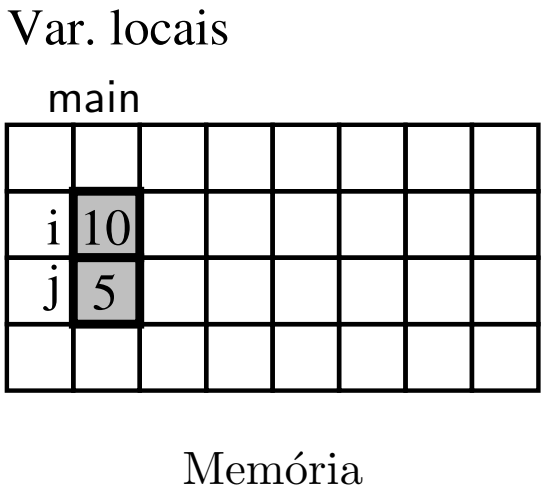
Tela

```
void troca(int *a, int *b)
{
  int temp;
  temp = *a;
  *a = *b;
  *b = temp;
}
main()
{
   int i=10; int j=5;
   troca(&i,&j);
   printf("i=%d j=%d\n",i,j);
}
```



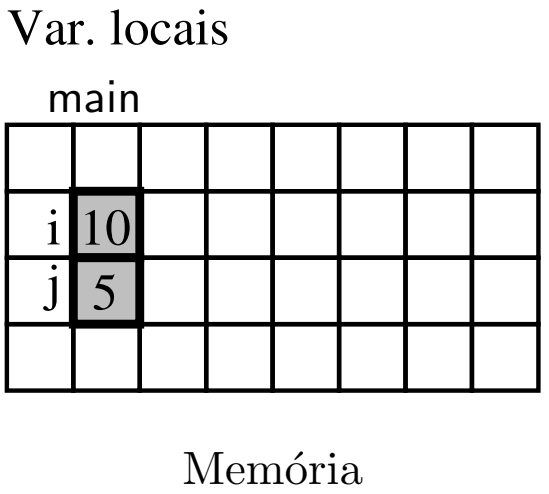
Tela

```
void troca(int *a, int *b)
{
  int temp;
  temp = *a;
  *a = *b;
  *b = temp;
}
main()
{
   int i=10; int j=5;
   troca(&i,&j);
   printf("i=%d j=%d\n",i,j);
}
```



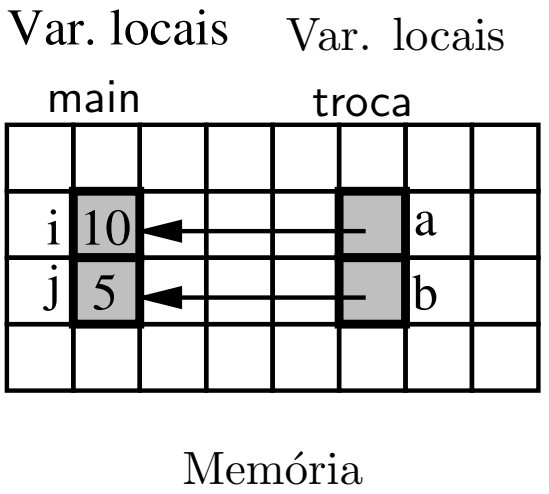
Tela

```
void troca(int *a, int *b)
{
  int temp;
  temp = *a;
  *a = *b;
  *b = temp;
}
main()
{
   int i=10; int j=5;
   troca(&i,&j);
   printf("i=%d j=%d\n",i,j);
}
```



Tela

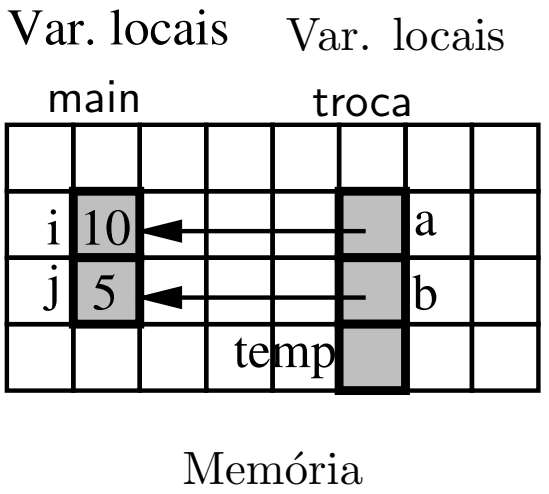
```
void troca(int *a, int *b)
{
  int temp;
  temp = *a;
  *a = *b;
  *b = temp;
}
main()
{
   int i=10; int j=5;
   troca(&i,&j);
   printf("i=%d j=%d\n",i,j);
}
```



Memória

Tela

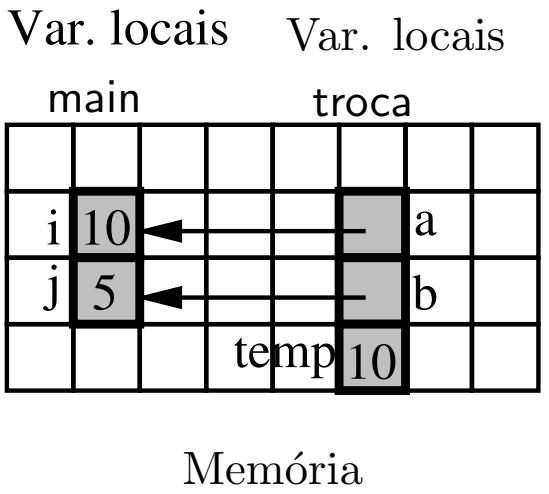
```
void troca(int *a, int *b)
{
  int temp;
  temp = *a;
  *a = *b;
  *b = temp;
}
main()
{
   int i=10; int j=5;
   troca(&i,&j);
   printf("i=%d j=%d\n",i,j);
}
```



Memória

Tela

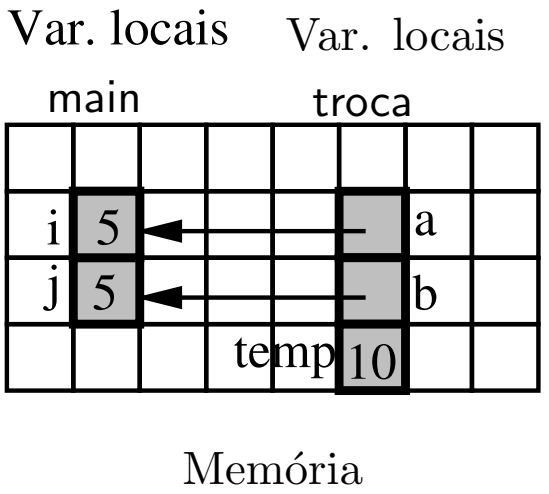
```
void troca(int *a, int *b)
{
  int temp;
  temp = *a;
  *a = *b;
  *b = temp;
}
main()
{
   int i=10; int j=5;
   troca(&i,&j);
   printf("i=%d j=%d\n",i,j);
}
```



Memória

Tela

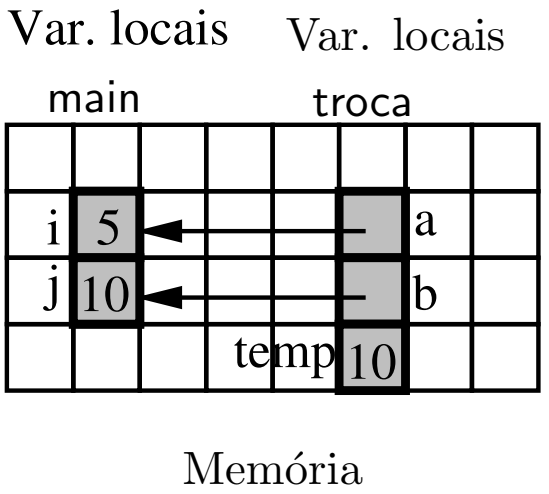
```
void troca(int *a, int *b)
{
  int temp;
  temp = *a;
  *a = *b;
  *b = temp;
}
main()
{
   int i=10; int j=5;
   troca(&i,&j);
   printf("i=%d j=%d\n",i,j);
}
```



Memória

Tela

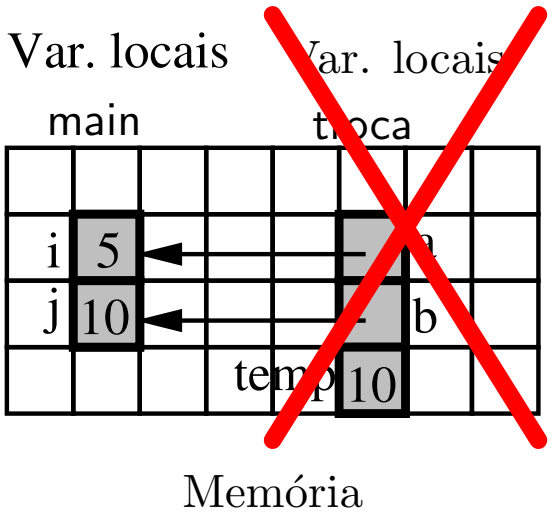
```
void troca(int *a, int *b)
{
  int temp;
  temp = *a;
  *a = *b;
  *b = temp;
}
main()
{
   int i=10; int j=5;
   troca(&i,&j);
   printf("i=%d j=%d\n",i,j);
}
```



Memória

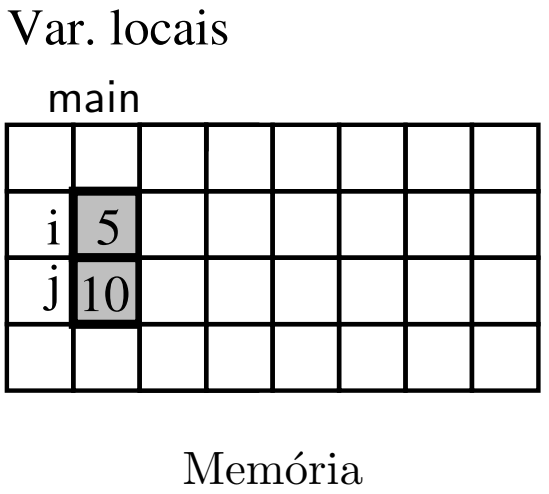
Tela

```
void troca(int *a, int *b)
{
  int temp;
  temp = *a;
  *a = *b;
  *b = temp;
}
main()
{
   int i=10; int j=5;
   troca(&i,&j);
   printf("i=%d j=%d\n",i,j);
}
```



Tela

```
void troca(int *a, int *b)
{
  int temp;
  temp = *a;
  *a = *b;
  *b = temp;
}
main()
{
   int i=10; int j=5;
   troca(&i,&j);
   printf("i=%d j=%d\n",i,j);
}
```



Tela

```
i=5 j=10
```

Sintaxe: Passagem de parâmetro por referência

- No protótipo: `tipo*` ao invés de `tipo`
- Na sequência de comandos da função: `*var` ao invés de `var`
- Na chamada da função: `&var` ao invés de `var`

Observações

- Numa função, podemos passar alguns parâmetros por valor, e outros por referência;
- Lembrar o uso do comando `scanf`:

```
int num;
scanf("%d", &num);
```

Exercício: min, max, media

Escrever uma função:

```
void lerNumerosInteiros(float *min,
                        float *max,
                        float *media)
```

que leia do teclado 10 números reais, e que retorna o valor mínimo, o valor máximo e a media dos números.
Testar a função.

Definição de protótipos

Para definir o protótipo de uma função, temos que responder às seguintes perguntas:

- A função retorna algum valor?
- De qual tipo?
- A função precisa de quais dados?
- Quais são os tipos deles?
- Alguns destes dados vão ser modificados?

Protótipo

```
tipo_retorno nome_funcao(tipo1 par1, tipo2* par2, ...)
```

Definição de protótipos

Para definir o protótipo de uma função, temos que responder às seguintes perguntas:

- A função retorna algum valor?
- De qual tipo?
- A função precisa de quais dados?
- Quais são os tipos deles?
- Alguns destes dados vão ser modificados?

Protótipo

```
tipo_retorno nome_funcao(tipo1 par1, tipo2* par2, ...)
```

Definição de protótipos

Para definir o protótipo de uma função, temos que responder às seguintes perguntas:

- A função retorna algum valor?
- De qual tipo?
- A função precisa de quais dados?
- Quais são os tipos deles?
- Alguns destes dados vão ser modificados?

Protótipo

```
tipo_retorno nome_funcao(tipo1 par1, tipo2* par2, ...)
```

Definição: Função recursiva

- Função que chama ela mesma.

Exemplo: Potência

Em matemática:

$$x^n = \begin{cases} 1 & se \quad n = 0 \\ x & se \quad n = 1 \\ x \times x^{n-1} & se \quad n > 1 \end{cases}$$

3^3

- $3^3 = 3 \times 3^2$
- $3^3 = 3 \times 3 \times 3^1$
- $3^3 = 3 \times 3 \times 3$
- $3^3 = 3 \times 9$
- $3^3 = 27$

Exemplo: Potência

Em matemática:

$$x^n = \begin{cases} 1 & se \quad n = 0 \\ x & se \quad n = 1 \\ x \times x^{n-1} & se \quad n > 1 \end{cases}$$

3^3

- $3^3 = 3 \times 3^2$
- $3^3 = 3 \times 3 \times 3^1$
- $3^3 = 3 \times 3 \times 3$
- $3^3 = 3 \times 9$
- $3^3 = 27$

Exemplo: Potência

Em matemática:

$$x^n = \begin{cases} 1 & se \quad n = 0 \\ x & se \quad n = 1 \\ x \times x^{n-1} & se \quad n > 1 \end{cases}$$

3^3

- $3^3 = 3 \times 3^2$
- $3^3 = 3 \times 3 \times 3^1$
- $3^3 = 3 \times 3 \times 3$
- $3^3 = 3 \times 9$
- $3^3 = 27$

Exemplo: Potência

Em matemática:

$$x^n = \begin{cases} 1 & se \quad n = 0 \\ x & se \quad n = 1 \\ x \times x^{n-1} & se \quad n > 1 \end{cases}$$

3^3

- $3^3 = 3 \times 3^2$
- $3^3 = 3 \times 3 \times 3^1$
- $3^3 = 3 \times 3 \times 3$
- $3^3 = 3 \times 9$
- $3^3 = 27$

Exemplo: Potência

Em matemática:

$$x^n = \begin{cases} 1 & se \quad n = 0 \\ x & se \quad n = 1 \\ x \times x^{n-1} & se \quad n > 1 \end{cases}$$

3^3

- $3^3 = 3 \times 3^2$
- $3^3 = 3 \times 3 \times 3^1$
- $3^3 = 3 \times 3 \times 3$
- $3^3 = 3 \times 9$
- $3^3 = 27$

Exemplo: Potência

Uma função

```
float potencia(float x, int n)
```

que retorna:

- 1 se $n = 0$;
- x se $n = 1$;
- x `*potencia(`$x, n-1$`)` se $n > 1$.

```c
float potencia(float x, int n)
{
  switch (n)
  {
    case 0:
      result = 1;
      break;
    case 1:
      result = x;
      break;
    default:
      result = x * potencia(x,n-1);
  }
  return(result);
}
main()
{
  int n = 5;
  float x = 2.2;
  float pot;
  pot = potencia(x,n);
  printf("Potencia: %.3f^%d=%.3f\n", x, n, pot);
}
```

Exercício: Fatorial

Escrever uma função

```
long int fatorial(int x)
```

que retorna:

- Um erro se $x < 0$;
- 1 se $x = 0$;
- x `*fatorial(`$x - 1$`)` se $x > 1$.

Testar a função.

Observações

- Toda função recursiva pode ser escrita de forma iterativa;
- Geralmente a forma recursiva é menos eficiente que a forma iterativa pois tem uma grande quantidade de chamadas;
- Em alguns casos, a forma recursiva pode ser muito elegante;
- Mas problemas podem aparecer...

Exercício: Fibonacci

Escrever uma função iterativa e uma função recursiva que computam o $n^{\text{ésimo}}$ elemento da sequência de Fibonacci:

$$\begin{cases} u_0 = 0 \\ u_1 = 1 \\ u_i = u_{i-1} + u_{i-2} \text{ se } i \geq 2 \end{cases}$$

Comparar o tempo de execução para $n = 40$.