Computação eletrônica: Ponteiros

Gurvan Huiban
ghuiban@cin.ufpe.br

14 de janeiro de 2014

Plano de aula

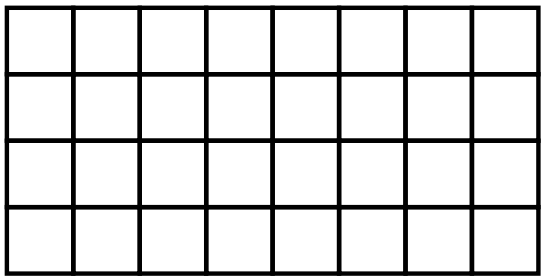
Exercício: Troca do valor de `a` e `b`

Escrever uma função que troca o valor dos parâmetros inteiros `a` e `b`.
Escrever um programa em C que testa se a função funcionou corretamente.

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.

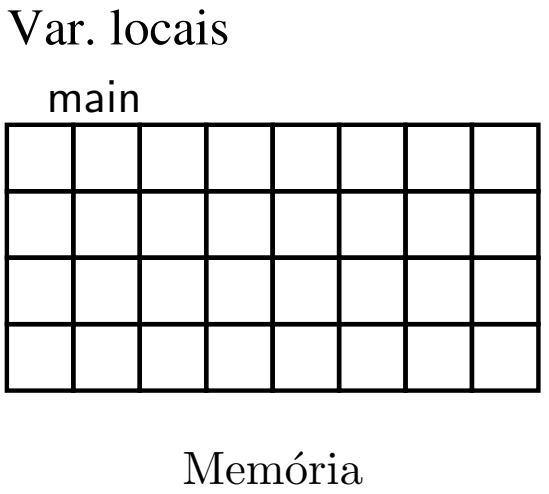
```
void troca(int a, int b)
{
  int temp;
  temp = a;
  a = b;
  b = temp;
}
main()
{
   int i=10; int j=5;
   troca(i,j);
   printf("i=%d j=%d\n",i,j);
}
```



Memória

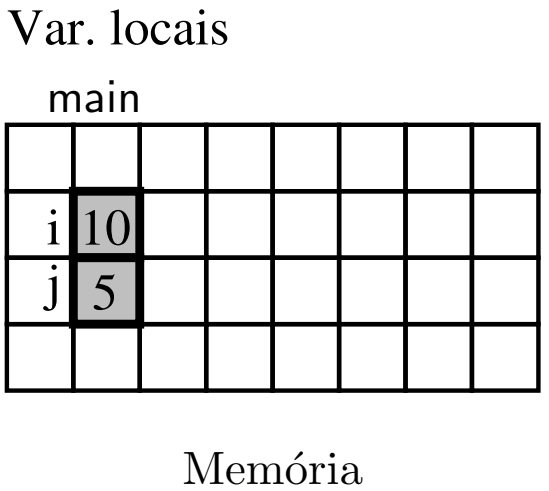
Tela

```
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);
}
```



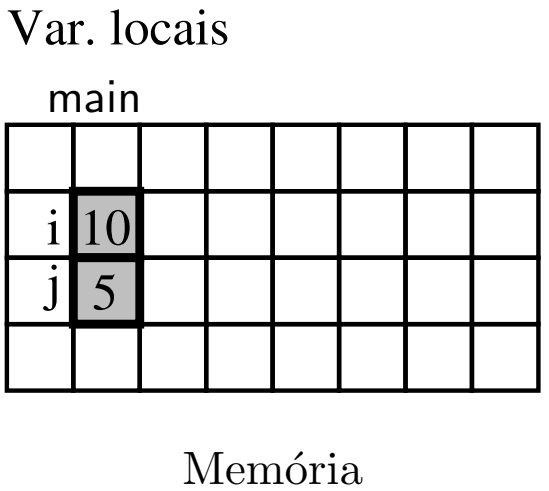
Tela

```
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);
}
```



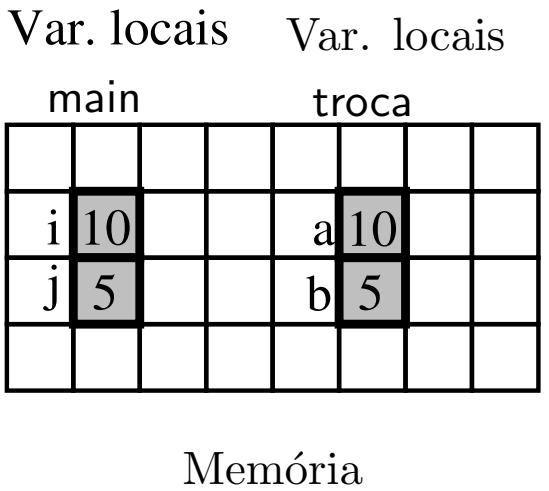
Tela

```
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

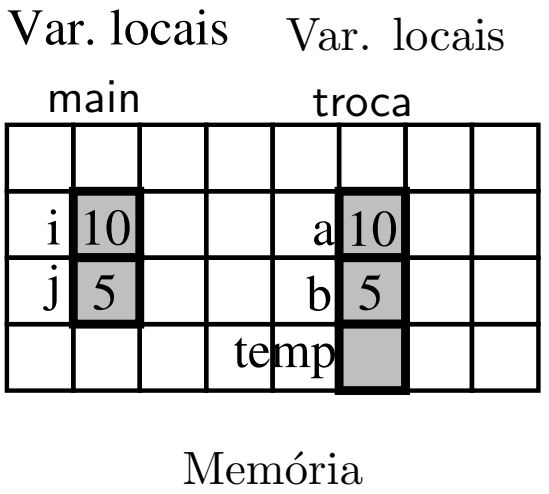
```
void troca(int a, int b)
{
  int temp;
  temp = a;
  a = b;
  b = temp;
}
main()
{
   int i=10; int j=5;
   troca(i,j);
   printf("i=%d j=%d\n",i,j);
}
```



Memória

Tela

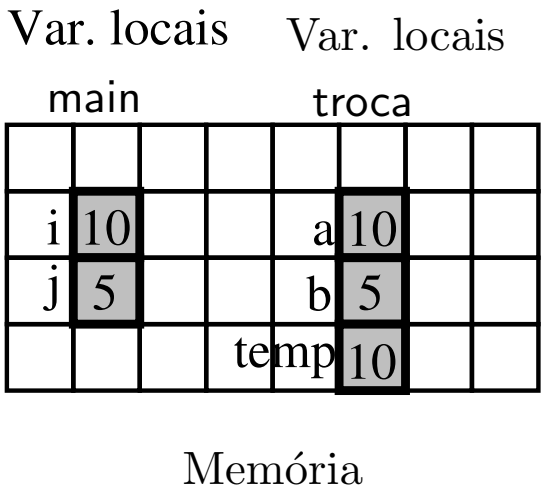
```
void troca(int a, int b)
{
  int temp;
  temp = a;
  a = b;
  b = temp;
}
main()
{
   int i=10; int j=5;
   troca(i,j);
   printf("i=%d j=%d\n",i,j);
}
```



Memória

Tela

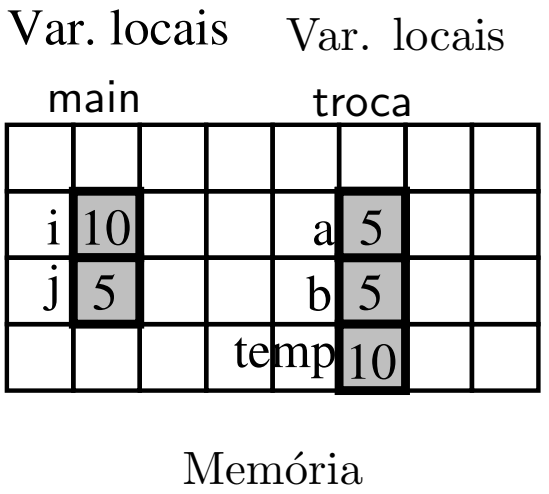
```
void troca(int a, int b)
{
  int temp;
  temp = a;
  a = b;
  b = temp;
}
main()
{
   int i=10; int j=5;
   troca(i,j);
   printf("i=%d j=%d\n",i,j);
}
```



Memória

Tela

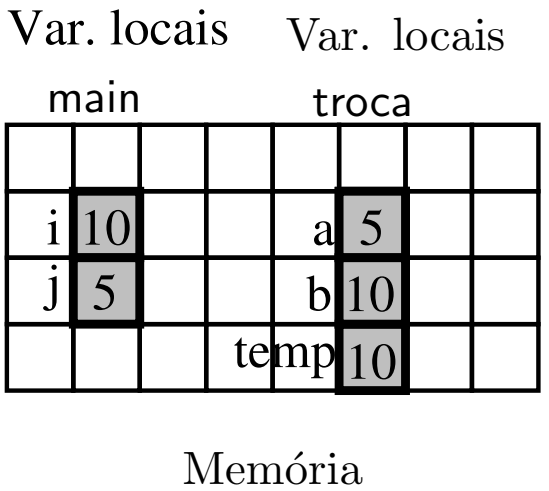
```
void troca(int a, int b)
{
  int temp;
  temp = a;
  a = b;
  b = temp;
}
main()
{
   int i=10; int j=5;
   troca(i,j);
   printf("i=%d j=%d\n",i,j);
}
```



Memória

Tela

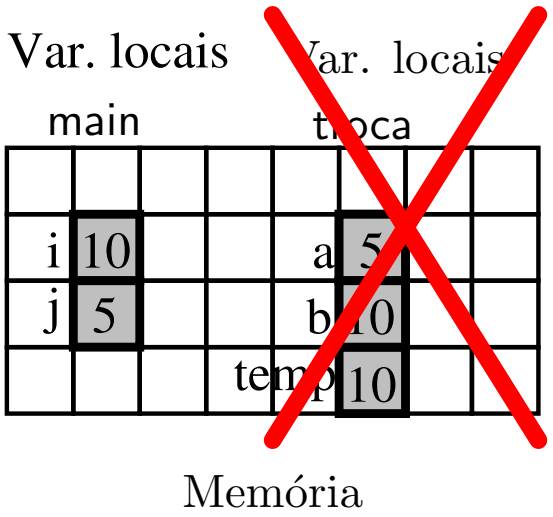
```
void troca(int a, int b)
{
  int temp;
  temp = a;
  a = b;
  b = temp;
}
main()
{
   int i=10; int j=5;
   troca(i,j);
   printf("i=%d j=%d\n",i,j);
}
```



Memória

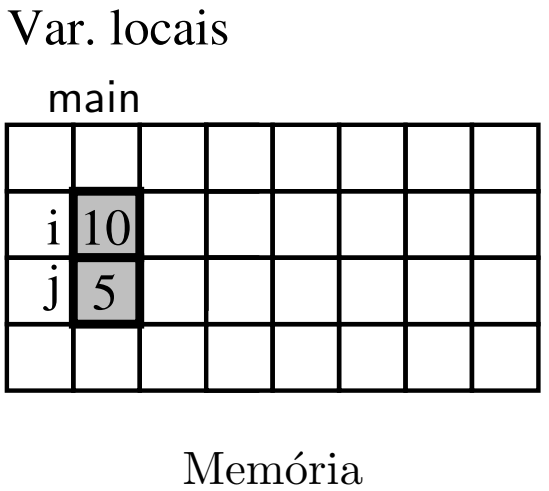
Tela

```
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
```

Memória até agora...

- Memória: gigante gaveteiro;
- Cada gaveta tem um identificador (endereço);
- Declarar uma variável é "colar" uma etiqueta numa(s) gaveta(s);
- As etiquetas ficam coladas na gaveta durante a execução inteira do programa/da função.

Definição: Ponteiro

- Um ponteiro (`pointer`) é uma variável que armazena um endereço da memória;
- Ou seja, atribuir um valor a um ponteiro corresponde a armazenar numa gaveta o endereço de uma outra gaveta.

Sintaxe: Declaração

```
tipo *nomePonteiro;
```

cria uma variável chamada `nomePonteiro` apontando para variáveis de tipo `tipo`. O tipo pode ser qualquer, inclusive um tipo definido pelo programador. Ex:

```
int *p;
```

Sintaxe: Uso

- `&` : Operador de endereço:
  Retorna o endereço da variável associada. Ex:

  ```
  p = &i;
  ```

- `*` : Operador de conteúdo:
  Acessa o valor apontado pelo ponteiro. Ex:

  ```
  *p = 2;
  ```

- `NULL` : Ponteiro nulo.

Tipo de ponteiro

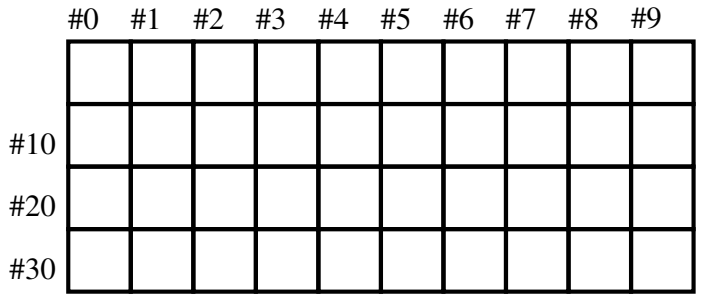
- Para acessar um valor apontado por um ponteiro, precisamos saber o tamanho do dado na memória;
  $\Rightarrow$Um ponteiro precisa ter um tipo.
- Respeitar os tipos!:
  Um ponteiro de inteiros deve apontar para uma variável inteira!

Impressão

Usar o formato `%p` para imprimir um endereço:

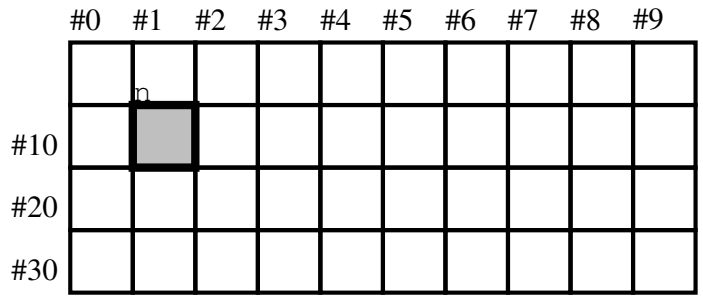
```
printf("Endereco de n=%p\n",&n);
```

```
int n;
int *p;
n = 1;
p = &n;
*p = 2;
printf("n=%d\n",n);
printf("*p=%d\n",*p);
printf("&n=%p p=%p\n",&n,p);
```



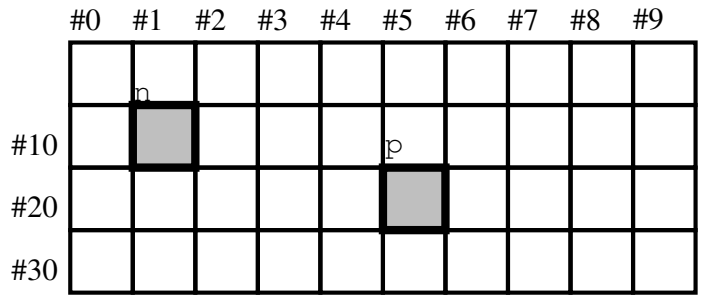
Tela

```
int n;
int *p;
n = 1;
p = &n;
*p = 2;
printf("n=%d\n",n);
printf("*p=%d\n",*p);
printf("&n=%p p=%p\n",&n,p);
```



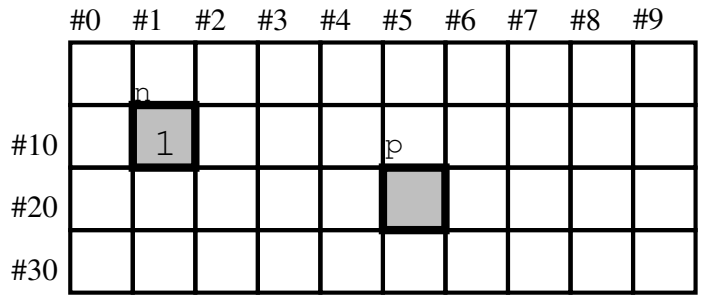
Tela

```
int n;
int *p;
n = 1;
p = &n;
*p = 2;
printf("n=%d\n",n);
printf("*p=%d\n",*p);
printf("&n=%p p=%p\n",&n,p);
```



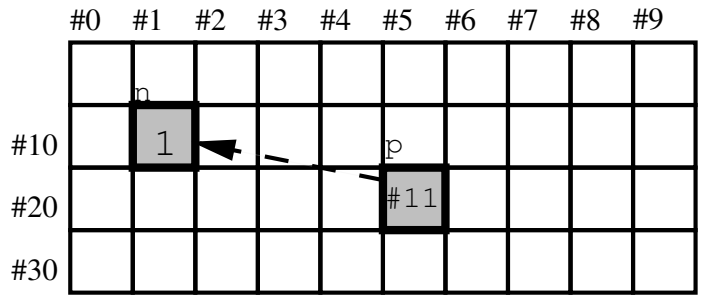
Tela

```
int n;
int *p;
n = 1;
p = &n;
*p = 2;
printf("n=%d\n",n);
printf("*p=%d\n",*p);
printf("&n=%p p=%p\n",&n,p);
```



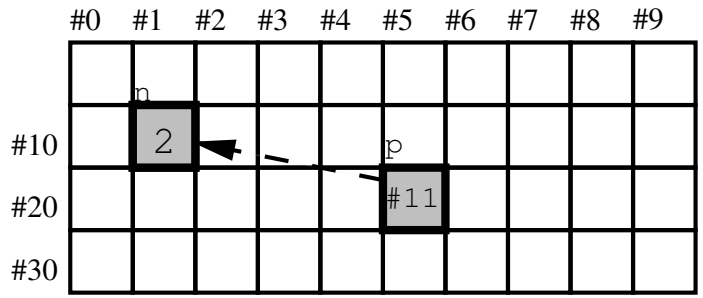
Tela

```
int n;
int *p;
n = 1;
p = &n;
*p = 2;
printf("n=%d\n",n);
printf("*p=%d\n",*p);
printf("&n=%p p=%p\n",&n,p);
```



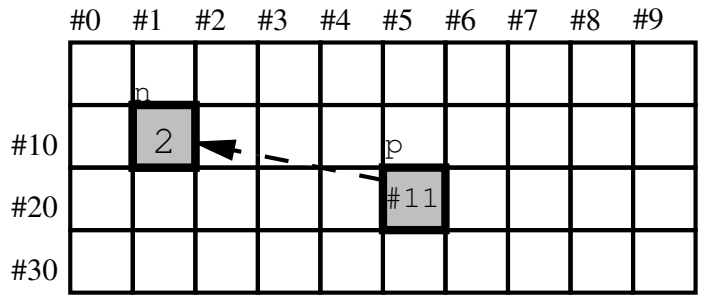
Tela

```
int n;
int *p;
n = 1;
p = &n;
*p = 2;
printf("n=%d\n",n);
printf("*p=%d\n",*p);
printf("&n=%p p=%p\n",&n,p);
```



Tela

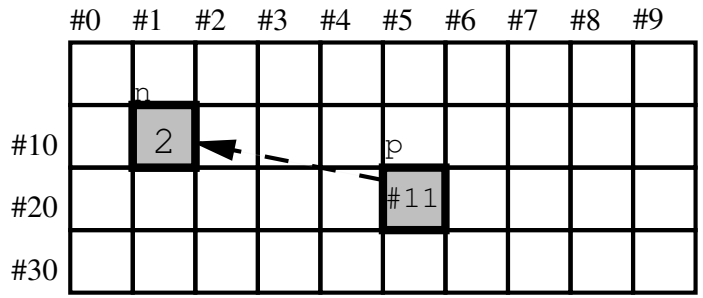
```
int n;
int *p;
n = 1;
p = &n;
*p = 2;
printf("n=%d\n",n);
printf("*p=%d\n",*p);
printf("&n=%p p=%p\n",&n,p);
```



Tela

```
n=2
```

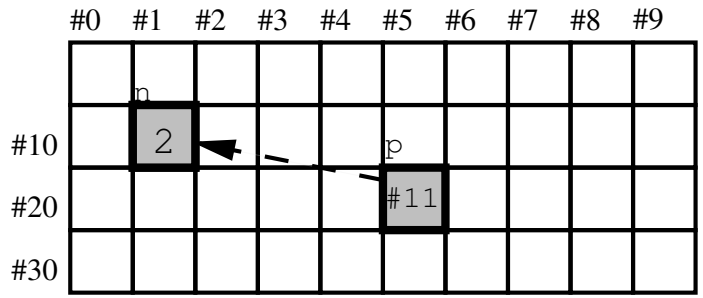
```
int n;
int *p;
n = 1;
p = &n;
*p = 2;
printf("n=%d\n",n);
printf("*p=%d\n",*p);
printf("&n=%p p=%p\n",&n,p);
```



Tela

```
n=2
*p=2
```

```
int n;
int *p;
n = 1;
p = &n;
*p = 2;
printf("n=%d\n",n);
printf("*p=%d\n",*p);
printf("&n=%p p=%p\n",&n,p);
```



Tela

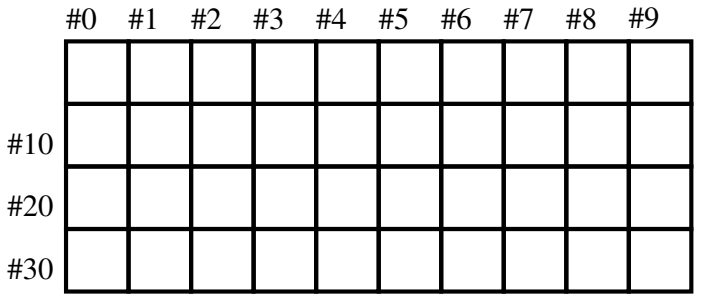
```
n=2
*p=2
&n=#11 p=#11
```

Operações

Podemos

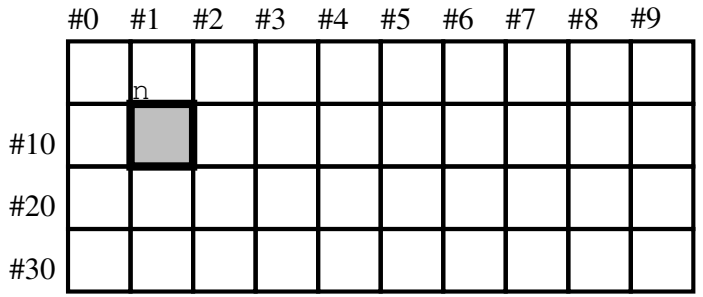
- Copiar ponteiros
- Comparar ponteiros
- Usar ponteiros como parâmetros
- Usar ponteiros como retorno de função

```
int n;
int *p,*q;
n = 1;
p = &n;
q = p;
n = 2;
*q = n + *p;
printf("n=%d\n",n);
printf("*p=%d *q=%d\n",*p,*q);
printf("&n=%p p=%p q=%p\n",&n,p,q);
```



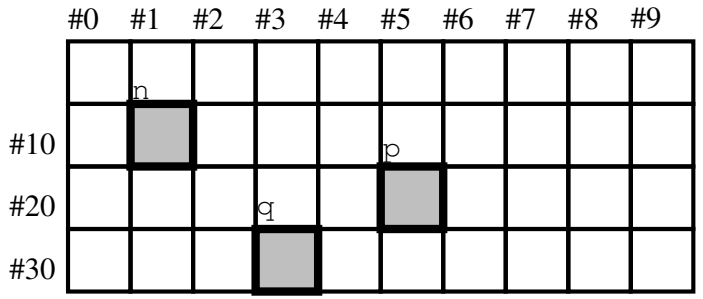
Tela

```
int n;
int *p,*q;
n = 1;
p = &n;
q = p;
n = 2;
*q = n + *p;
printf("n=%d\n",n);
printf("*p=%d *q=%d\n",*p,*q);
printf("&n=%p p=%p q=%p\n",&n,p,q);
```



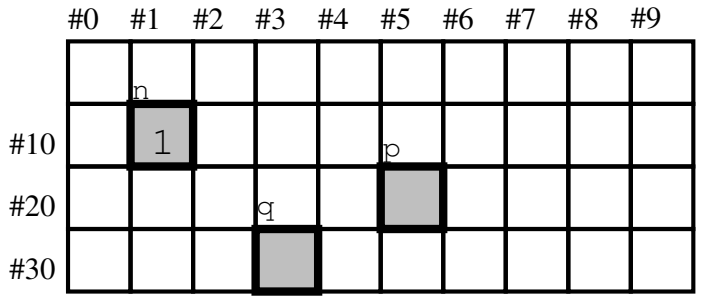
Tela

```
int n;
int *p,*q;
n = 1;
p = &n;
q = p;
n = 2;
*q = n + *p;
printf("n=%d\n",n);
printf("*p=%d *q=%d\n",*p,*q);
printf("&n=%p p=%p q=%p\n",&n,p,q);
```



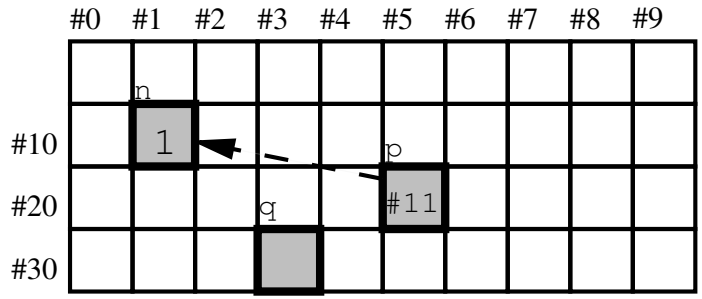
Tela

```
int n;
int *p,*q;
n = 1;
p = &n;
q = p;
n = 2;
*q = n + *p;
printf("n=%d\n",n);
printf("*p=%d *q=%d\n",*p,*q);
printf("&n=%p p=%p q=%p\n",&n,p,q);
```



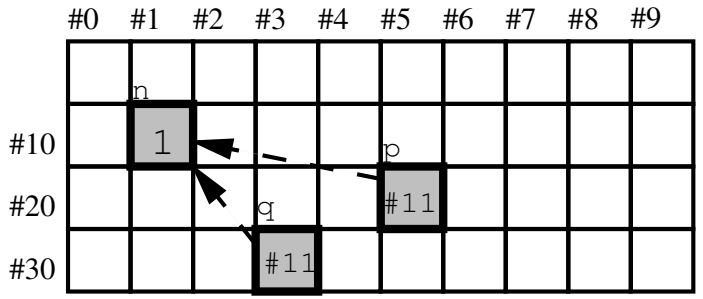
Tela

```
int n;
int *p,*q;
n = 1;
p = &n;
q = p;
n = 2;
*q = n + *p;
printf("n=%d\n",n);
printf("*p=%d *q=%d\n",*p,*q);
printf("&n=%p p=%p q=%p\n",&n,p,q);
```



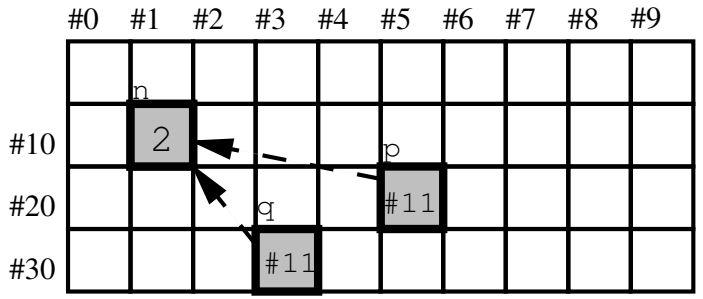
Tela

```
int n;
int *p,*q;
n = 1;
p = &n;
q = p;
n = 2;
*q = n + *p;
printf("n=%d\n",n);
printf("*p=%d *q=%d\n",*p,*q);
printf("&n=%p p=%p q=%p\n",&n,p,q);
```



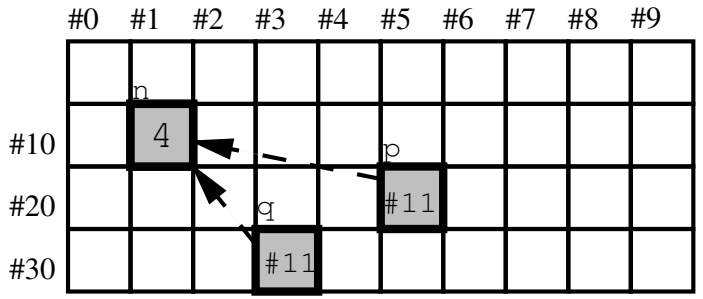
Tela

```
int n;
int *p,*q;
n = 1;
p = &n;
q = p;
n = 2;
*q = n + *p;
printf("n=%d\n",n);
printf("*p=%d *q=%d\n",*p,*q);
printf("&n=%p p=%p q=%p\n",&n,p,q);
```



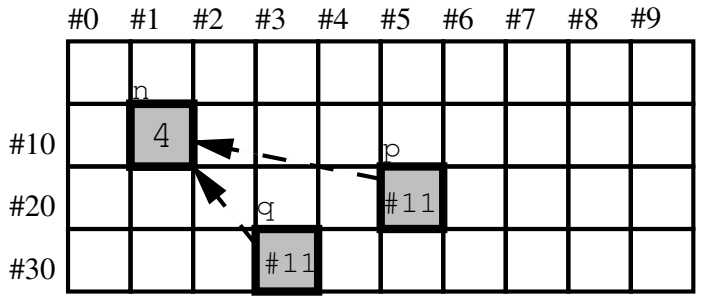
Tela

```
int n;
int *p,*q;
n = 1;
p = &n;
q = p;
n = 2;
*q = n + *p;
printf("n=%d\n",n);
printf("*p=%d *q=%d\n",*p,*q);
printf("&n=%p p=%p q=%p\n",&n,p,q);
```



Tela

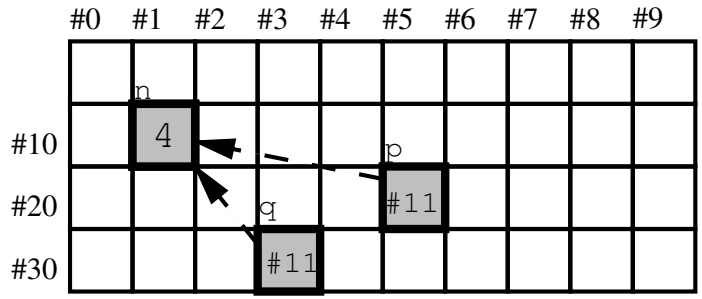
```
int n;
int *p,*q;
n = 1;
p = &n;
q = p;
n = 2;
*q = n + *p;
printf("n=%d\n",n);
printf("*p=%d *q=%d\n",*p,*q);
printf("&n=%p p=%p q=%p\n",&n,p,q);
```



Tela

```
n=4
```

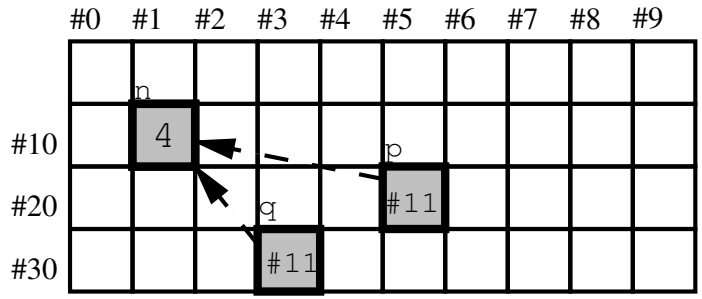
```
int n;
int *p,*q;
n = 1;
p = &n;
q = p;
n = 2;
*q = n + *p;
printf("n=%d\n",n);
printf("*p=%d *q=%d\n",*p,*q);
printf("&n=%p p=%p q=%p\n",&n,p,q);
```



Tela

```
n=4
*p=4 *q=4
```

```
int n;
int *p,*q;
n = 1;
p = &n;
q = p;
n = 2;
*q = n + *p;
printf("n=%d\n",n);
printf("*p=%d *q=%d\n",*p,*q);
printf("&n=%p p=%p q=%p\n",&n,p,q);
```



Tela

```
n=4
*p=4 *q=4
&n=#11 p=#11 q=#11
```

Cuidado!

Com ponteiros, acessamos diretamente a memória.
⇒Garantir que os ponteiros sempre apontam para um lugar que tem sentido.

Exemplo: Atribuição errada

```
int* prepCrash()
{
    int n;
    n=2;
    printf("n=%d", n);
    return &n;
}
int main(void)
{
    int *p;
    p = prepCrash();
    *p = 12; //Errado! O n nao existe mais
    printf("*p=%d\n", *p);
    return 0;
}
```

Exercício: Troca do valor de `a` e `b`

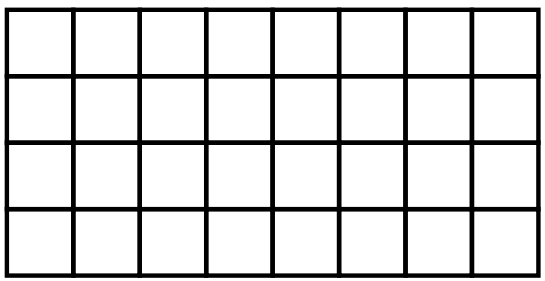
Escrever uma função que troca o valor dos parâmetros inteiros `a` e `b`.
Escrever um programa em C que testa se a função funcionou corretamente.

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);
}
```

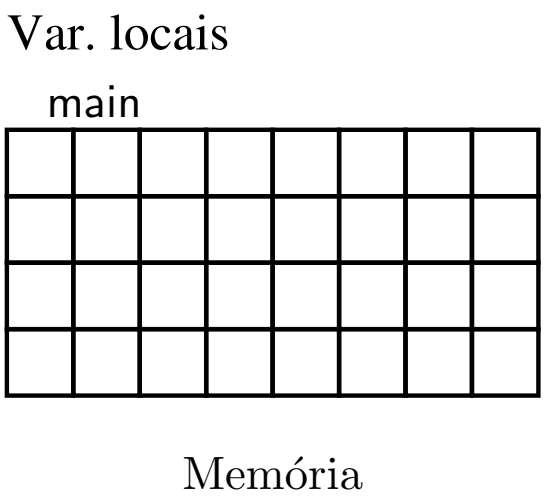
```
void troca(int *a, int *b)
{
  int temp;
  temp = *a;
  *a = *b;
  *b = temp;
}
main()
{
   int i=10; int j=5;
   troca(&i,&j);
   printf("i=%d j=%d\n",i,j);
}
```



Memória

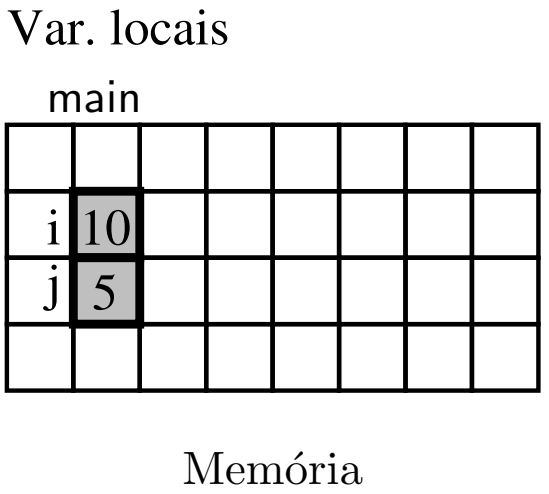
Tela

```
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);
}
```



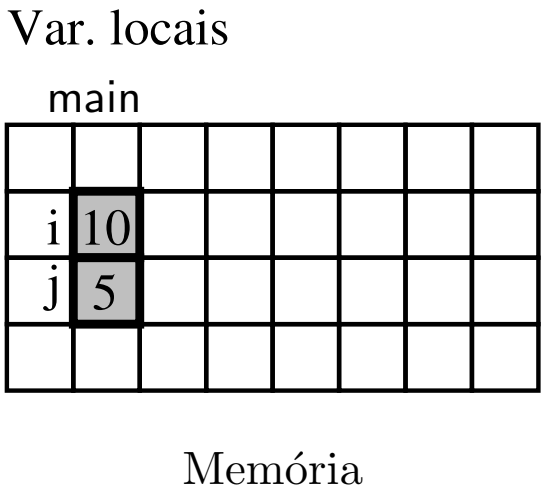
Tela

```
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);
}
```



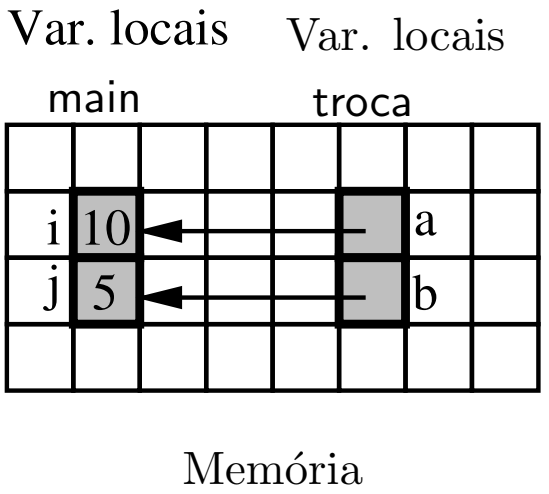
Tela

```
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

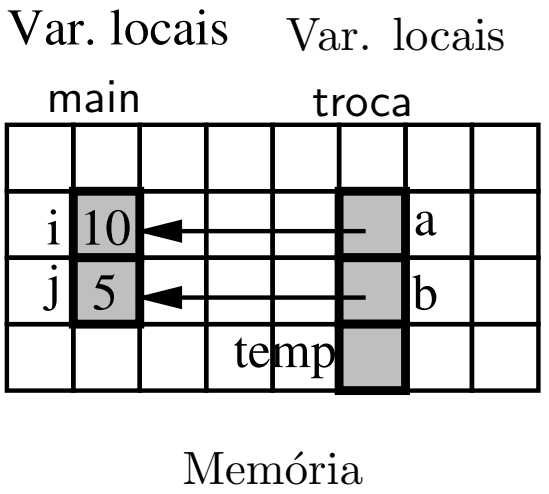
```
void troca(int *a, int *b)
{
  int temp;
  temp = *a;
  *a = *b;
  *b = temp;
}
main()
{
   int i=10; int j=5;
   troca(&i,&j);
   printf("i=%d j=%d\n",i,j);
}
```



Memória

Tela

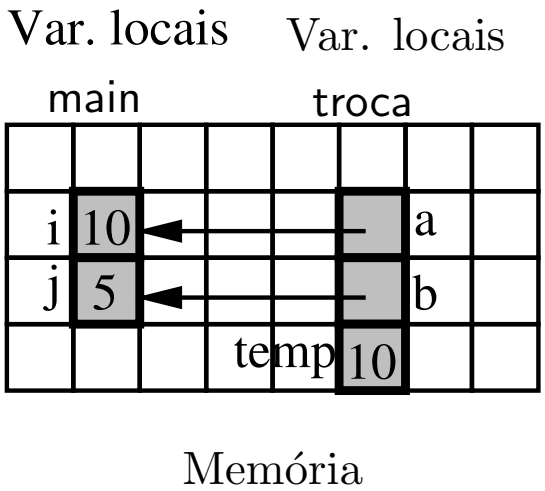
```
void troca(int *a, int *b)
{
  int temp;
  temp = *a;
  *a = *b;
  *b = temp;
}
main()
{
   int i=10; int j=5;
   troca(&i,&j);
   printf("i=%d j=%d\n",i,j);
}
```



Memória

Tela

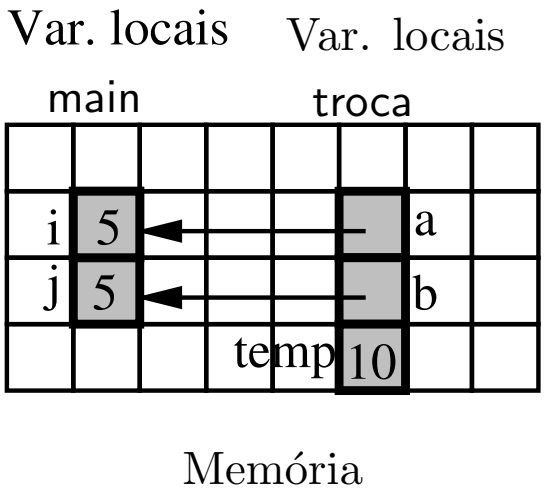
```
void troca(int *a, int *b)
{
  int temp;
  temp = *a;
  *a = *b;
  *b = temp;
}
main()
{
   int i=10; int j=5;
   troca(&i,&j);
   printf("i=%d j=%d\n",i,j);
}
```



Memória

Tela

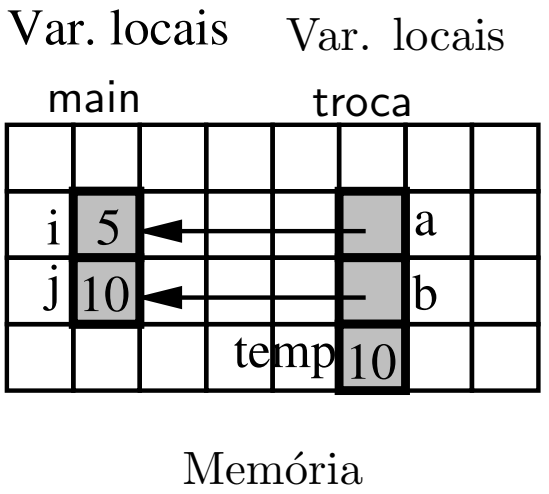
```
void troca(int *a, int *b)
{
  int temp;
  temp = *a;
  *a = *b;
  *b = temp;
}
main()
{
   int i=10; int j=5;
   troca(&i,&j);
   printf("i=%d j=%d\n",i,j);
}
```



Memória

Tela

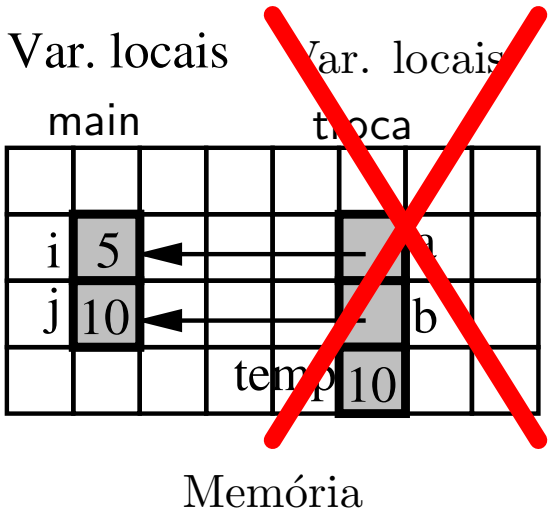
```
void troca(int *a, int *b)
{
  int temp;
  temp = *a;
  *a = *b;
  *b = temp;
}
main()
{
   int i=10; int j=5;
   troca(&i,&j);
   printf("i=%d j=%d\n",i,j);
}
```



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);
}
```

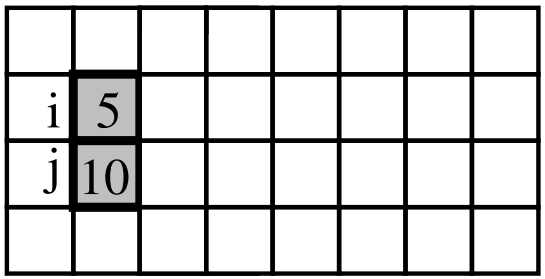


Tela

```
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);
}
```

Var. locais

main



Memória

Tela

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

Sintaxe: Passagem de parâmetro por referência

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

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.

Ponteiro sobre vetor

Variável vetor

```
int tab[5];
```

- `tab` contém o endereço do primeiro elemento do vetor
- Podemos atribuir este endereço para a um ponteiro

```
int* ptr = tab;
```

- O ponteiro se comporta como o vetor:
  `ptr[3]` retorna o elemento de índice 3 do vetor
- A recíproca não é válida:

```
tab = ptr; //Errado
```

Exemplo de ponteiro sobre vetor

```
int *p, tab[5], i;
for (i=0; i<5; i++)
{
   tab[i] = i;
}
p=tab;
for (i=0; i<5; i++)
{
   printf("%d ", p[i]);
}
printf("\n");
```

Custo do uso das estruturas

Tamanho de uma estrutura

- Agregação de dados
- Potencialmente: Grande quantidade de dados

Passagem de parâmetro por valor

- Copia o valor da variável
- Com estrutura: Copia todos os dados da estrutura

Custo alto (tempo, memória)

Passagem de parâmetro por referência

- Cópia somente o endereço

Custo baixo

Exemplos

Passagem por valor

```
void imprimirAluno(struct TAluno a)
{
   printf("Nome do aluno: %s\n", a.nome);
   printf("Matricula: %d\n",a.mat);
   ...
```

Passagem por referência

```
void imprimirAluno(struct TAluno *a)
{
   printf("Nome do aluno: %s\n", (*a).nome);
   printf("Matricula: %d\n",(*a).mat);
   ...
```

operador ->

Observação

- as parenteses ao redor do `*a` são necessárias: Prioridade do operador `.` sobre `*`
- Operador `->`
  `a->` equivale a `(*a).`

Observação

```
void imprimirAluno(struct TAluno *a)
{
   printf("Nome do aluno: %s\n", a->nome);
   printf("Matricula: %d\n",a->mat);
   ...
```