

Passagem de Parâmetros

- Duas formas de passagem de parâmetros
- Por valor
- Por referência
 - Ponteiros!

Exemplo 1: passagem por valor

- Fazer uma função que receba 2 inteiros x e y e troque seus valores.

```
void troca(int x, int y) {  
    int aux;  
  
    aux = x;  
    x = y;  
    y = aux;  
}
```

```
int main() {  
    int a=5, b=10;  
  
    printf("a=%d b=%d\n",a,b);  
    troca(a,b);  
    printf("a=%d b=%d\n",a,b);  
  
    return 0;  
}
```

- O que será impresso na tela?

Passagem de parâmetro por referência

- Não passa uma cópia do valor
- Passa uma referência ao valor na memória do computador
 - Alterações feitas no parâmetro afetam o conteúdo **apontado** pela referência.
- Para isso, usamos **ponteiros!**

Endereço de memória

- Variáveis são guardadas em posições da memória
- Cada posição da memória é **endereçada**

```
int main(){  
    int x = 1;  
    int y = 10;  
  
    return 0;  
}
```

“Endereço de x é 1000”

Memória:

[endereço] conteúdo

[1000]

[1001]

[1002]

[1003]

[1004]

⋮

⋮

“Endereço de y é 1001”

Obtendo endereços: operador &

- Usado para obter o endereço de uma variável
- Uso: `&<variável>`

```
int main(){
    int x = 1;
    int y = 10;

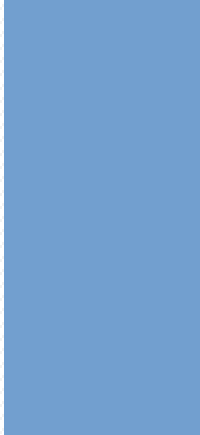
    print("O end de x eh %p \n", &x);

    print("O end de y eh %p \n", &y);

    return 0;
}
```

```
0 end de x eh 0x1000
0 end de y eh 0x1001
```

M em ó ria :

[endereço]	conteúdo
[1000]	
[1001]	
[1002]	
[1003]	
[1004]	
⋮	⋮

O que são ponteiros?

- São **variáveis** que armazenam **posições de memória**
- Declaração de variáveis ponteiro

```
< tipo> * < nome_variável_ponteiro> ;
```

- Tipo: tipo da variável apontada (int, float, char, ...)
- * indica que essa variável é um ponteiro

Exemplos de ponteiros

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

```
int main(){  
    int x = 1; //variável int  
    float y = 10.0; //variável float  
  
    int * p1;      //ponteiro para variável int  
    float * p2; //ponteiro para variável float  
  
    p1 = &x;      //p1 aponta para x  
    p2 = &y;      //p2 aponta para y  
  
    print("O end de x eh %p \n", p1);  
  
    print("O end de y eh %p \n", p2);  
  
    return 0;  
}
```

x:[1000]

y:[1001]

p1:[1002]

p2:[1003]

[1004]

⋮

⋮

```
0 end de x eh 0x1000  
0 end de y eh 0x1001
```

Acessando ponteiros: operador *

- Se temos um ponteiro, podemos conteúdo apontado com o operador unário *

```
int main(){
    int x = 31;
    int y;

    int * p1;

    p1 = &x;

    y = *p1;

    printf("O valor de y eh %d", y);

    return 0;
}
```

x:[1000]

y:[1001]

p1:[1002]

[1003]

[1004]

⋮

⋮

O valor de y eh 31

Passagem por referência e ponteiros

- Passagem por referência utiliza ponteiros
- Ao invés de cópias dos valores, são passadas referências (ponteiros) para estes valores
- Assim, esses valores podem ser modificados dentro da função

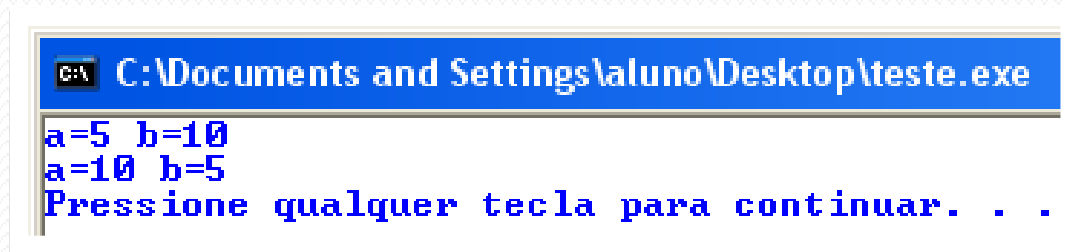
Exemplo 2: passagem por referências

- Fazer uma função que receba 2 inteiros x e y e troque seus valores

```
void troca(int *x, int *y){  
    int aux;  
  
    aux = *x;  
    *x = *y;  
    *y = aux;  
}
```

```
int main() {  
    int a=5, b=10;  
  
    printf("a=%d b=%d\n",a,b);  
    troca(&a, &b);  
    printf("a=%d b=%d\n",a,b);  
  
    return 0;  
}
```

- Passagem de parâmetros:
 - x e y agora recebem os endereços de a e b
 - Valores são buscados e armazenados nas posições de memória apontadas por x e y
 - Ou seja, em a e b !



A screenshot of a Windows command prompt window. The title bar is blue and contains the text "C:\Documents and Settings\aluno\Desktop\teste.exe". The command prompt area is white and contains the following text in blue: "a=5 b=10", "a=10 b=5", and "Pressione qualquer tecla para continuar. . .".

- Os valores foram trocados, pois a passagem de **parâmetros** foi feita por **referência**
- Dentro do subprograma, **x** e **y** receberam os **endereços de memória** de **a** e **b**, que são alterados

Vetores, parâmetros e ponteiros

- Vetores não podem ser passados por valor
 - Imagine copiar um vetor com 2M posições!!
- **Vetores** são **sempre** passados por **referência**
- Vetores são (uma espécie de) ponteiros!
 - Declaração de um vetor apenas reserva uma quantidade de memória proporcional ao tamanho do vetor
 - Variável do **vetor** aponta para a **posição** de memória do **primeiro elemento** do vetor

Passagem de vetor como parâmetro

- Declaração de um parâmetro **vetor** em uma **função**

< tipo> < nome> []

< tipo> *< nome>

- Onde
 - **tipo**: corresponde ao tipo dos elementos do vetor
 - **nome**: é o nome atribuído ao vetor
 - **[]**: indica que a variável é do tipo vetor
 - pode ser utilizado sem um valor, pois em C não interessa qual a dimensão do vetor que é passado a uma função, mas sim o tipo dos seus elementos
 - *****: indica que é um ponteiro para o tipo primitivo de dado do vetor

Exemplo 3

- Crie uma função `inic()` que inicializa os elementos de um vetor com zeros e faça um programa principal que a teste

```
#include <stdio.h>

void inic(int *s, int n) {
    int i;
    for (i=0; i<n; i++){
        s[i] = 0;
    }
}
```

```
int main() {
    int v[10], i;

    inic(v, 10);

    for(i=0; i<10; i++) {
        printf("Elemento %d = %d \n", i, v[i]);
    }
    return 0;
}
```

Exercício 1

Implemente e teste a função

```
float max(float v[], int n)
```

- que recebe um vetor de números reais e o número de elementos a considerar e cujo retorno é o maior número dentre os ***n*** primeiros elementos do vetor.

Exemplo 4

- Faça uma função que receba uma string como parâmetro e retorne o número de vogais desta.

Resumo

- Passagem por valor
 - Tipos simples (ou *primitivos*)
 - Quando não queremos modificar o valor dos parâmetros
- Passagem por referência
 - Tipos simples quando queremos modificar os seus valores
 - Ou **sempre** em tipos complexos (e.g. arranjos)
- Argumentos na passagem de parâmetros por referência:
 - **Se a variável for um vetor**, seu nome corresponde ao endereço do seu primeiro elemento
 - **Se a variável não for um vetor**, então ela deve ser precedida de & na chamada da função (se quisermos alterá-la na função)