Ponteiros, Funções e parâmetros

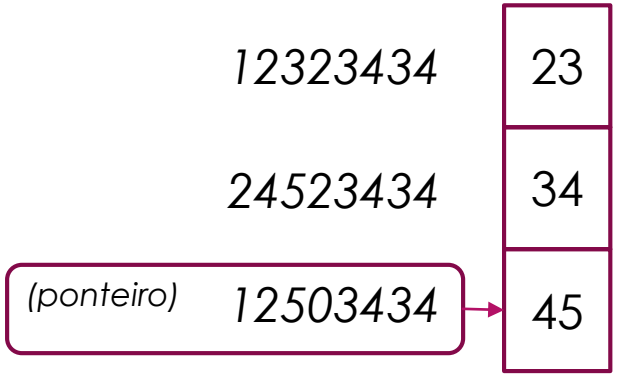
Ponteiros

- O nome de uma variável indica o que está armazenado nela.
- O endereço de uma variável é um ponteiro.
- Seu valor indica em que parte da memória do computador a variável está alocada.
- Ponteiros proporcionam um modo de acesso à variável sem referenciá-la diretamente (modo indireto de acesso).

http://www.redes.unb.br/~udias/cpe/aulas18--.pdf

Ponteiros

- Um **Ponteiro** ou apontador é um tipo de dado de uma linguagem de programação cujo valor se refere diretamente a um outro valor alocado em outra área da memória, através de seu endereço.
- Um **ponteiro** é simplesmente uma variável que armazena o endereço de outra variável.



Ponteiros

- Existem dois operadores especiais para trabalhar com ponteiros. São eles:

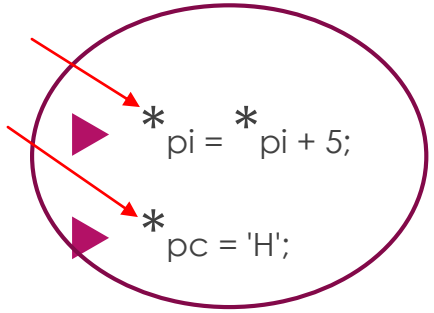
- * - conteúdo do endereço apontado por
- & - endereço de

Observe o exemplo abaixo:

```
1   main( ) {
2   int v1, v2, * p, * q
3   v1 = 3;
4   v2 = 12;
5   p  = &v1;       // p recebe o endereço de memória da variável v1
6   q = p;         // copia o endereço guardado em p para q
7   *q = 44;      // altera o valor armazenado no endereço apontado por q
8   }
```

O operador de dereferência: *

- Quando um ponteiro aponta para um endereço de memória, a operação para acessar o conteúdo do endereço apontado é chamado de **dereferência**. O operador unário * é usado para fazer a **dereferência**. Note que este uso do símbolo * não tem relação com o símbolo de multiplicação. Usando os exemplos anteriores, *pi é o objeto apontado por pi.
- *pi tem valor 5
- *pc tem valor 'G'
- Como um ponteiro dereferenciado (tais como *pi ou *pc) refere-se a um objeto na memória, ele pode ser usado não só como valor, mas também como um lvalue. Isto significa que um ponteiro dereferenciado pode ser usado no lado esquerdo de uma atribuição. Veja alguns exemplos:
- printf("Valor= %d, Char = %c\n", *pi, *pc);



http://www.inf.ufpr.br/cursos/ci067/Docs/NotasAula/notas-28_Ponteiros.html

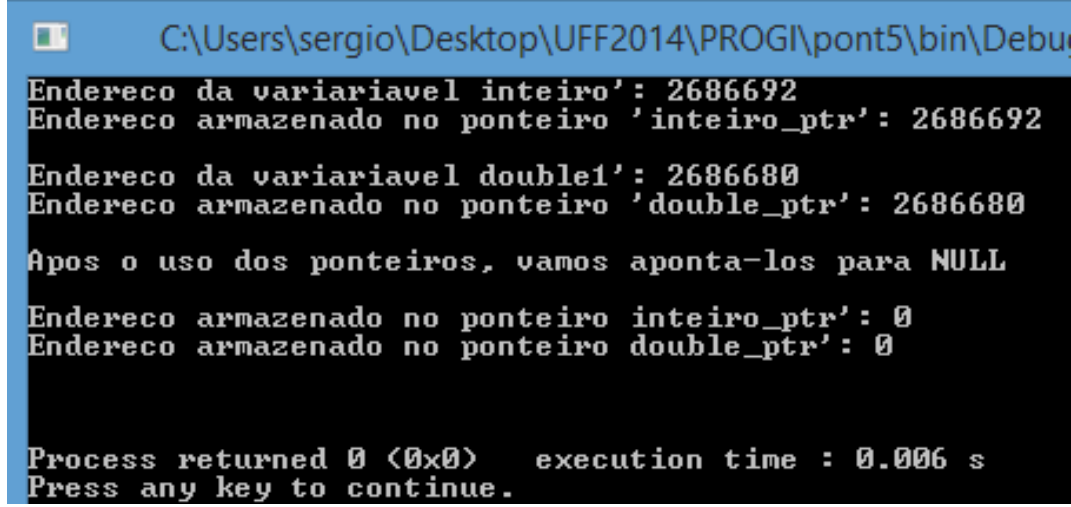
Como inicializar um ponteiro em C – A constante NULL

- É sempre bom inicializarmos os ponteiros, pois senão eles podem vir com lixo e você se esquecer, posteriormente, de inicializar. Então, quando for usar, pensará que está usando o ponteiro de modo correto, mas estará usando o ponteiro com ele apontando para um lixo (endereço qualquer de memória).

- Uma boa prática é apontar os ponteiros para a primeira posição de memória, que é conhecida como **NULL**. Sempre que terminar de usar um ponteiro, coloque ele pra apontar para a posição NULL. Para fazer isso, faça:
- tipo *nome_do_ponteiro = NULL;

http://www.cprogressivo.net/2013/03/Como-declarar-inicializar-e-usar-ponteiros-e-a-constante-NULL-em-C.html

```c
#include <stdio.h>

int main(void)

{
    int inteiro;
    int *inteiro_ptr = &inteiro;

    double double1;
    double *double_ptr = &double1;

    printf("Endereco da variariavel inteiro': %d\n", &inteiro);
    printf("Endereco armazenado no ponteiro 'inteiro_ptr': %d\n\n", inteiro_ptr);

    printf("Endereco da variariavel double1': %d\n", &double1);
    printf("Endereco armazenado no ponteiro 'double_ptr': %d\n\n", double_ptr);

    printf("Apos o uso dos ponteiros, vamos aponta-los para NULL\n\n");

    inteiro_ptr = NULL;
    double_ptr = NULL;

    printf("Endereco armazenado no ponteiro inteiro_ptr': %d\n", inteiro_ptr);
    printf("Endereco armazenado no ponteiro double_ptr': %d\n\n\n", double_ptr);
    return 0;
}
```



http://www.cprogressivo.net/2013/03/Como-declarar-inicializar-e-usar-ponteiros-e-a-constante-NULL-em-C.html

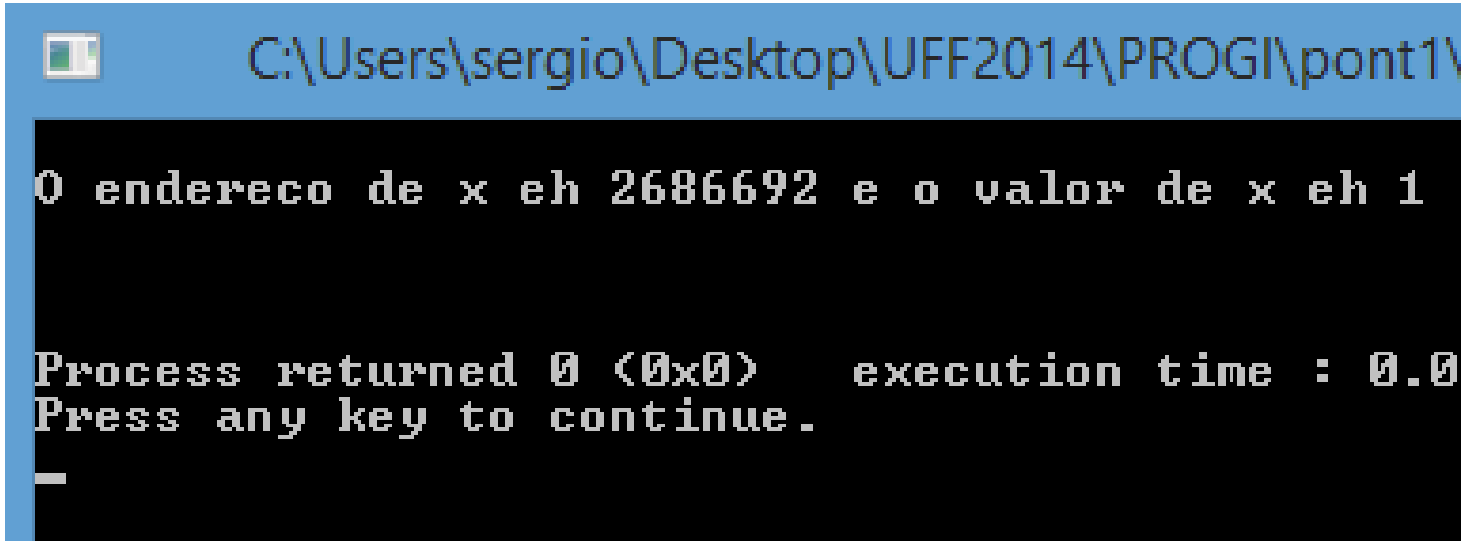
Atribuições envolvendo ponteiros

- Um ponteiro pode ter atribuído a si um valor que seja o endereço de memória onde está armazenado um valor do mesmo tipo do ponteiro. Isto ocorre quando se usa o operador de endereço visto acima, ou quando se usa o valor de um outro ponteiro que aponte para um objeto do mesmo tipo do primeiro ponteiro. Observe-se o exemplo abaixo:

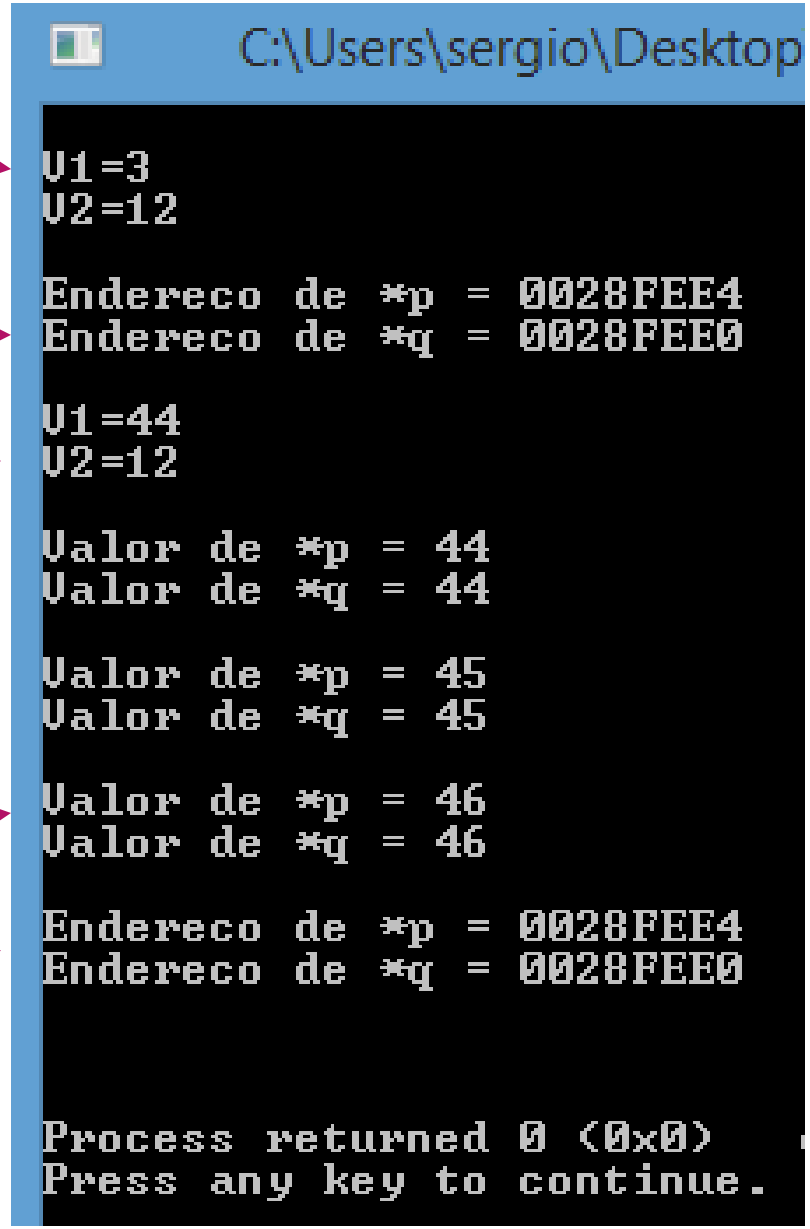
```
int *p1, *p2, x;
float *p3;

p1 = &x;      /* Correto */
p2 = p1;      /* Correto */
p3 = p1;      /* Incorreto. Compilador acusa "Warning". */
```

http://www.inf.ufpr.br/cursos/ci067/Docs/NotasAula/notas-28_Ponteiros.html

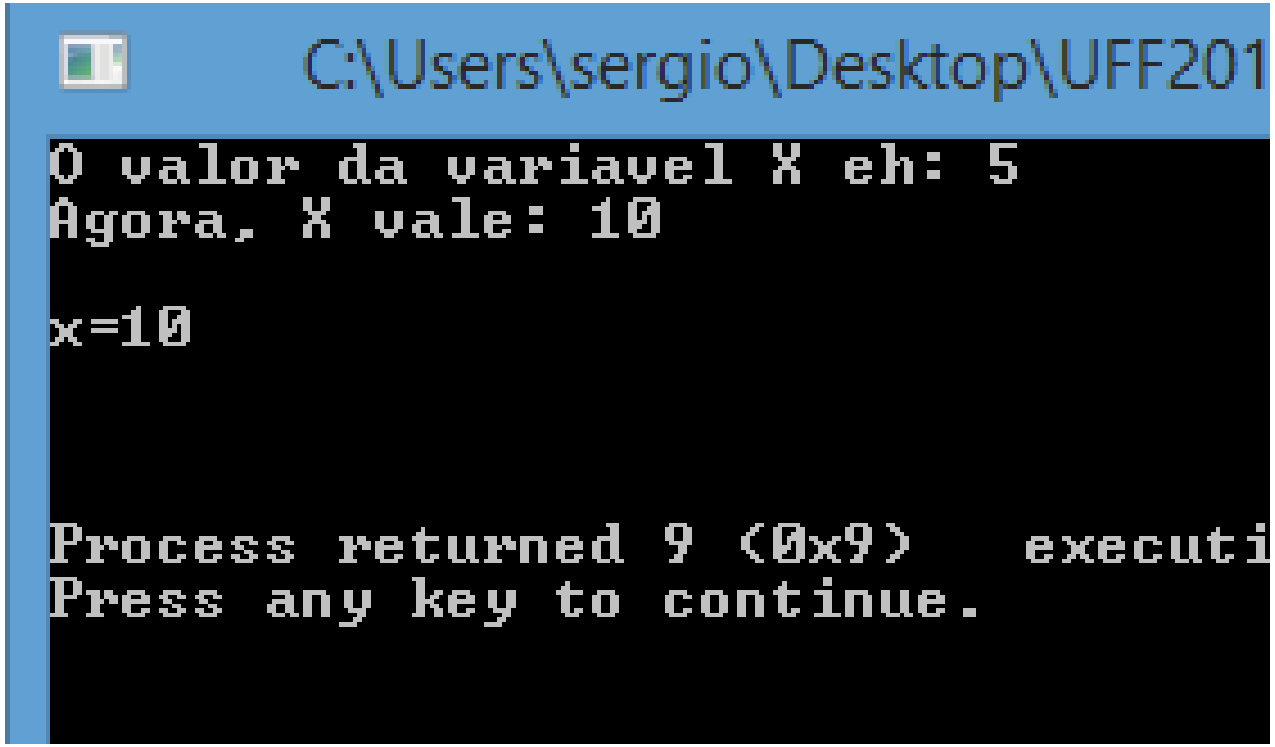
```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
   int x,y;
   int *px,*py;

   x = 1;      /* atribui à variável x o valor 1. */
   y = 2;      /* atribui à variável y o valor 2. */

   px= &x;    /* atribui ao ponteiro px o endereco da variavel x. */
   py= &y;    /* atribui ao ponteiro py o endereco da variavel y. */

   printf("\nO endereco de x eh %d e o valor de x eh %d\n\n\n",px,*px);

   return 0;
}
```



```c
int main() {
   int v1, v2, *p, *q;
   v1 = 3;
   v2 = 12;

   printf("\nV1=%d", v1);
   printf("\nV2=%d", v2);

   p  = &v1;       // p recebe o endereço de memória da variável v1
   q = p;          // copia o endereço guardado em p para q
   *q = 44;        // altera o valor armazenado no endereço apontado por q

   printf("\n\nEndereco de *p = %p", &p);
   printf("\nEndereco de *q = %p", &q);

   printf("\n\nV1=%d", v1);
   printf("\nV2=%d", v2);

   printf("\n\nValor de *p = %d", *p);
   printf("\nValor de *q = %d", *q);

   *p = *p +1;
   printf("\n\nValor de *p = %d", *p);
   printf("\nValor de *q = %d", *q);

   *q = *q +1;
   printf("\n\nValor de *p = %d", *p);
   printf("\nValor de *q = %d", *q);

   printf("\n\nEndereco de *p = %p", &p);
   printf("\nEndereco de *q = %p", &q);


   printf("\n\n\n");
   return 0;
}
```



```c
#include <stdio.h>
void main()
{
  int x;
  int *ptr;
  x = 5;
  ptr = &x;
  printf("O valor da variável X eh: %d\n", *ptr);
  *ptr = 10;
  printf("Agora, X vale: %d\n", *ptr);
  printf("\nx=%d",x);
}
```



Funções com ponteiros

Passagem por Valor

- É a forma **mais comum utilizada** para passagem de parâmetros. Por exemplo, considere a família de funções trigonométricas, como seno, cosseno, etc. A função seno, por exemplo, recebe o valor de um ângulo (um número real) e devolve o seno desse ângulo.
- Se tivermos as funções seno e cosseno, podemos facilmente definir uma função tangente.
- Em projetos grandes de desenvolvimento de software, grupos de programadores podem trabalhar no desenvolvimento de funções distintas e juntar os seus trabalhos uma vez que tenham suas funções prontas. Para isso, basta que cada grupo conheça o protótipo das funções que precisa utilizar.

Passagem por Referência

- Se possível EVITE USAR!

- Sempre que possível é recomendável utilizar a forma de passagem por valor, para evitar "efeitos colaterais", mas há situações onde esses efeitos são desejáveis, por exemplo, **quando desejamos criar uma função que retorne mais de um valor**.

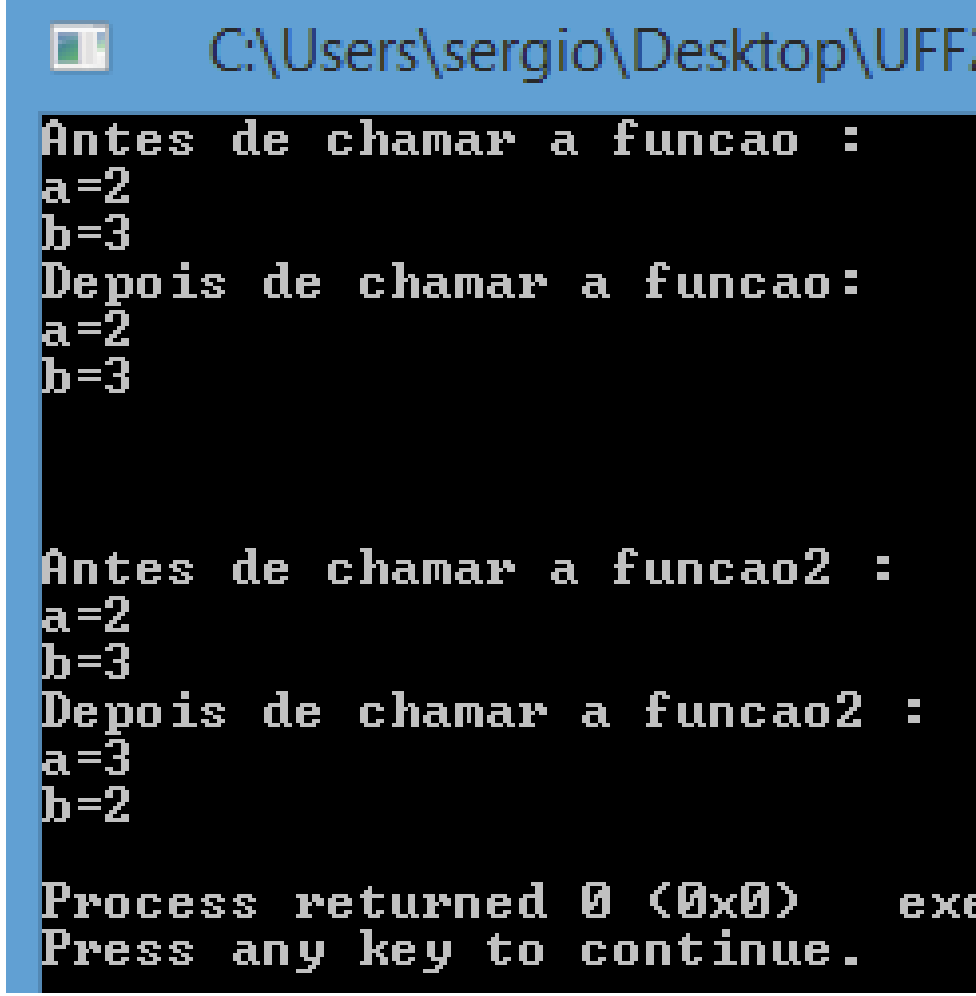
O que elas tem de diferente??

```c
#include<stdio.h>

void troca(int a, int b){
        int temp;
        temp=a;
        a=b;
        b=temp;
}
void troca2(int *a, int *b){
        int temp;
        temp=*a;
        *a=*b;
        *b=temp;
}
int main(){
        int a=2,b=3;
        printf("Antes de chamar a funcao :\na=%d\nb=%d\n",a,b);
        troca(a,b);
        printf("Depois de chamar a funcao:\na=%d\nb=%d\n",a,b);

        printf("\n\n\nAntes de chamar a funcao2 :\na=%d\nb=%d\n",a,b);
        troca2(&a,&b);
        printf("Depois de chamar a funcao2 :\na=%d\nb=%d\n",a,b);
        return 0;
}
```

```c
#include<stdio.h>

void troca(int a, int b){
        int temp;
        temp=a;
        a=b;
        b=temp;
}
void troca2(int *a, int *b){
        int temp;
        temp=*a;
        *a=*b;
        *b=temp;
}
int main(){
        int a=2,b=3;
        printf("Antes de chamar a funcao :\na=%d\nb=%d\n",a,b);
        troca(a,b);
        printf("Depois de chamar a funcao:\na=%d\nb=%d\n",a,b);

        printf("\n\n\nAntes de chamar a funcao2 :\na=%d\nb=%d\n",a,b);
        troca2(&a,&b);
        printf("Depois de chamar a funcao2 :\na=%d\nb=%d\n",a,b);
        return 0;
}
```

Passando parâmetros por valor

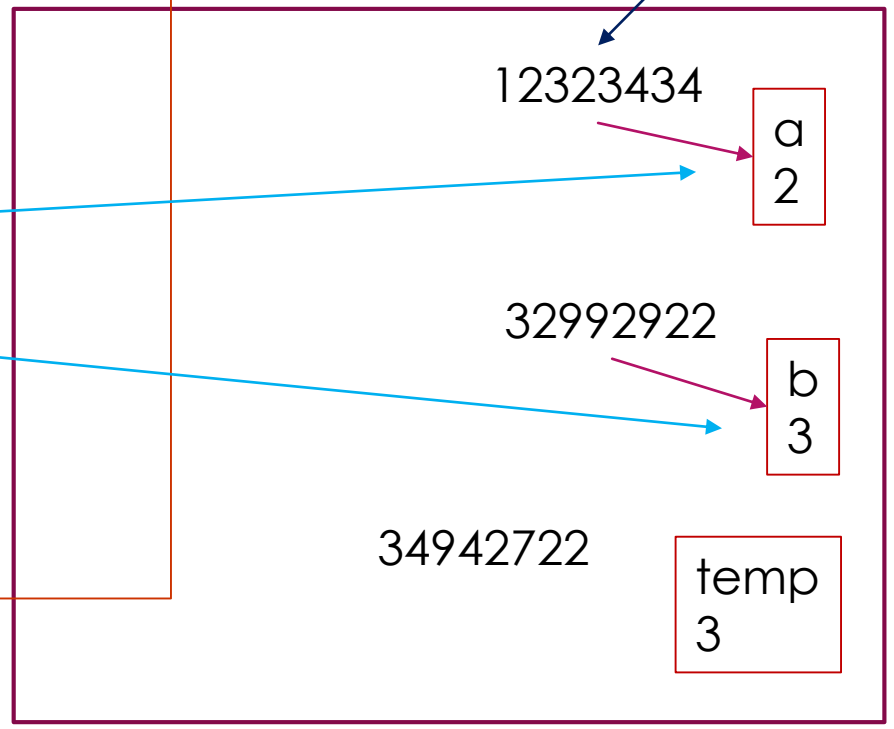
Passando parâmetros por referência



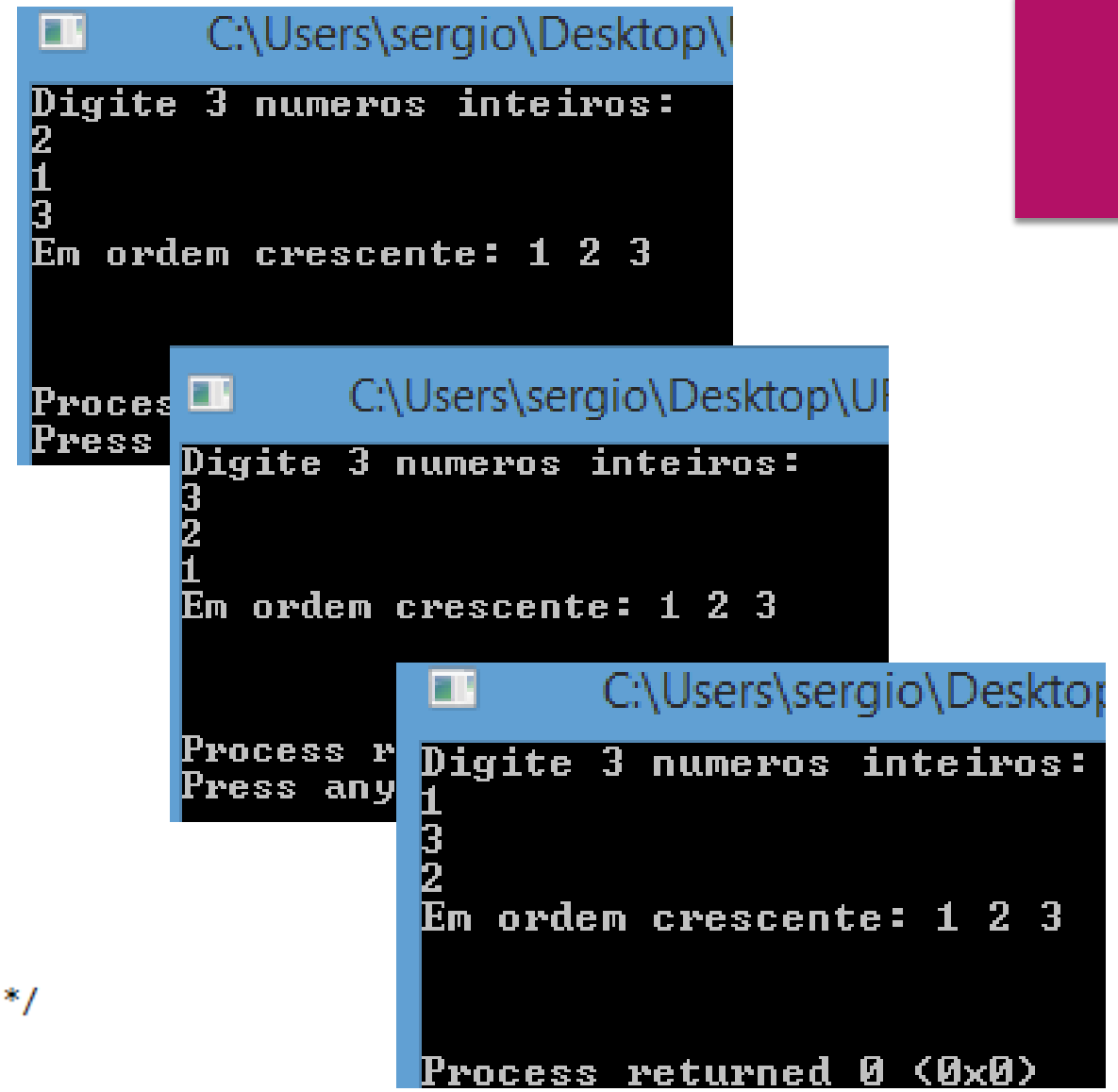
Nome de variáveis → Temp b a

Programa em C

```
#include<stdio.h>

void troca(int a, int b){
	int temp;
	temp=a;
	a=b;
	b=temp;
}
void troca2(int *a, int *b){
	int temp;
	temp=*a;
	*a=*b;
	*b=temp;
}
int main(){
	int a=2,b=3;
	printf("Antes de chamar a funcao :\na=%d\nb=%d\n",a,b);
	troca(a,b);
	printf("Depois de chamar a funcao:\na=%d\nb=%d\n",a,b);

	printf("\n\n\nAntes de chamar a funcao2 :\na=%d\nb=%d\n",a,b);
	troca2(&a,&b);
	printf("Depois de chamar a funcao2 :\na=%d\nb=%d\n",a,b);
	return 0;
}
```



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

/* prototipo */
void Troca(int *px, int *py);

int main ()
{
   int a, b, c;

   printf ("Digite 3 numeros inteiros: \n");
   scanf("%d %d %d", &a, &b, &c);

   if (a > b) Troca (&a, &b);
   if (b > c) Troca (&b, &c);
   if (a > b) Troca (&a, &b);

   printf("Em ordem crescente: %d %d %d\n\n\n", a, b, c);

   getchar();
   return 0;
}

void Troca(int *px, int *py)
/* Troca os valores das variaveis apontadas por px e py. */
{
   int n;

   n= *py;
   *py= *px;
   *px= n;
}
```



http://www.ime.usp.br/~elo/IntroducaoComputacao/Funcoes%20passagem%20de%20parametros.htm

Só entende ponteiros
quem usa ponteiros,
quem testa, quem usa,
quem testa, quem usa...