

INSTITUTO FEDERAL

Rio Grande do Sul

Campus Porto Alegre

Instituto Federal de Educação, Ciência e Tecnologia
do Rio Grande do Sul

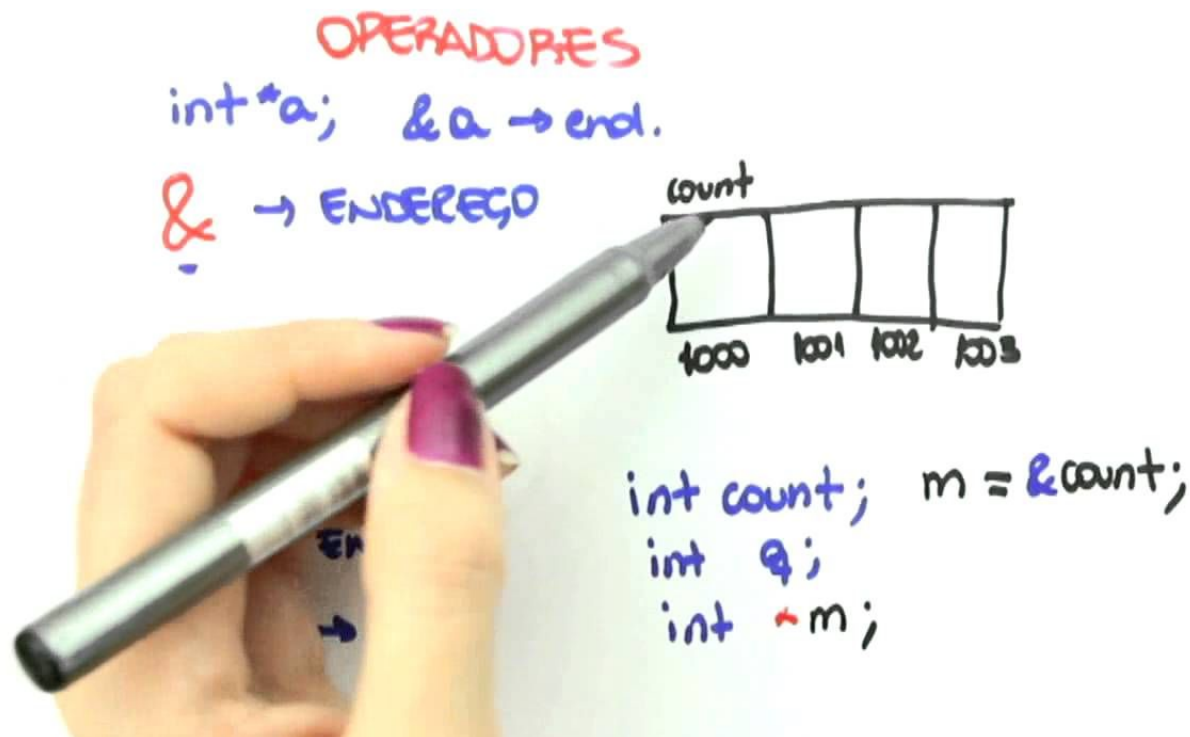
Professor Fabio Okuyama

Email: fabio.okuyama@poa.ifrs.edu.br

Ponteiros

O que são ponteiros?

- Armazenam endereços de memória
- O endereço de memória **aponta** para uma variável
- Permitem realizar alocação dinâmica de memória
- Permitem que uma função altere o conteúdo efetivo de uma variável



Para que serve uma referencia indireta?

Você tem diversas contas e deseja receber dinheiro através de PIX, de acordo com as contas que você precisa pagar. Então você pode passar sempre a mesma chave PIX e alterar para qual banco/agencia/conta você quer que ele direcione. Quem deposita não precisa saber que você possui 6 contas, só precisa saber a sua chave PIX.

VARIÁVEL CHAVE PIX
(51)9999-9999

CONTA CORRENTE

BANCO A - AGENCIA B – CONTA C

BANCO D – AGENCIA E – CONTA F

BANCO G - AGENCIA I – CONTA H

BANCO H – AGENCIA I – CONTA J

BANCO K – AGENCIA L – CONTA M

BANCO N – AGENCIA O – CONTA P

Para que serve uma referencia indireta?

Seu programa recebe vários números e precisa fazer algum processamento com estes números. O programa que preenche os números não deve ter acesso à sua variável, exceto o local onde o número será inserido.

PONTEIRO
POS_LIVRE



Endereço	Conteúdo
11EEFF	1234
22AB12	LIXO (LIVRE)
AA33DD	3456
AB13EF	12344
12ABEF	LIXO (LIVRE)
34AB5F	111111

Sintaxe para declaração de ponteiros

Sintaxe:

```
tipo *nomeDoPonteiro;
```

Onde:

tipo: Tipo de dado da variável que será apontada

nomeDoPonteiro: Nome do ponteiro.

Segue mesmo padrão de variáveis

Ex:

```
int * ponteiro;
```

```
float * ponteiro2;
```

Lembrando

```
int x=0;
```

```
int *ptrX; //cria ponteiro para inteiro
```

```
ptrX = &x; //ptrX recebe o endereço da variável x;
```

& pega o endereço da variável que vem depois;

&x é o endereço da variável x

***** pega o conteúdo apontado pelo ponteiro

*** ptrX** é o valor armazenado no endereço de memória.

Se **ptrX = &x**, então ***ptrX == x**

Exemplo

Declarando um ponteiro;

```
int x, y; //variável x, variavel y;

int *ptr; // ponteiro para variável inteira

ptr = &x; //ptr recebe o endereço da variavel x

x = 0;

y = *ptr; // y recebe o conteúdo da variável
           // apontada pelo ponteiro que
           // neste caso equivale a escrever y = x;
```


Exemplo de manipulação de ponteiros

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

```
int main() {
```

```
    int x, *px, *py;
```

```
    x = 9;
```

```
    px = &x;
```

```
    py = px;
```

```
    printf ("\nx    = %d", x);
```

```
    printf ("\n&x  = %d", &x);
```

```
    printf ("\npx   = %d", px);
```

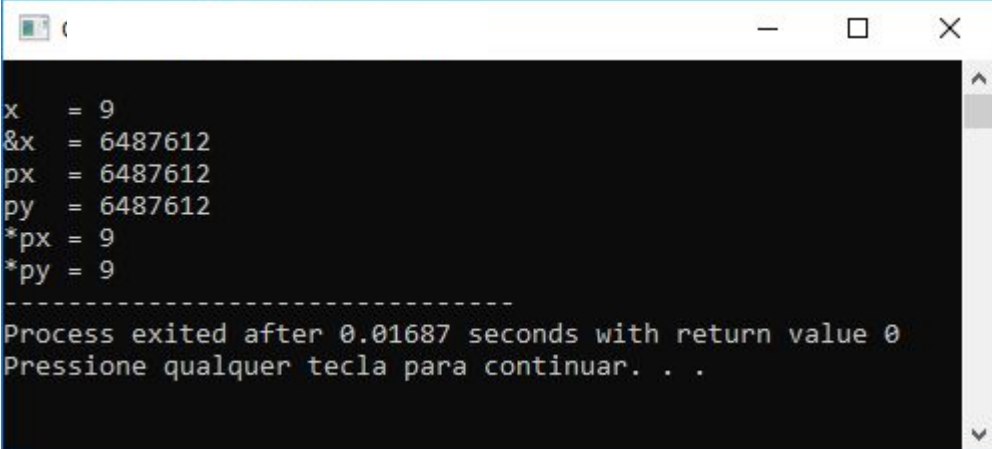
```
    printf ("\npy   = %d", py);
```

```
    printf ("\n*px  = %d", *px);
```

```
    printf ("\n*py  = %d", *py);
```

```
    return 0;
```

```
}
```



```
x    = 9
&x   = 6487612
px    = 6487612
py    = 6487612
*px   = 9
*py   = 9
-----
Process exited after 0.01687 seconds with return value 0
Pressione qualquer tecla para continuar. . .
```

Função Main

Declarando **int** e **dois ponteiros para int**;

x recebe o valor **9**

px RECEBE o endereço da variável **x**

py RECEBE o endereço armazenado em **px**.

Manipulação de ponteiros

Ponteiros são variáveis como quaisquer outras

Podem ser manipuladas da mesma forma

Um endereço de memória é um valor numérico

Este valor numérico pode ser utilizado em operações;

Expressões com ponteiros

- Se px aponta para uma variavel x
- *px pode ser utilizado em qualquer ponto em que x poderia ser utilizado;
- `int y,x,*px; px=&x; x=0;`
- `y=*px+1;` //y recebe 0+1;
- `y=*(px+1)` // y recebe o valor apontado // por px +1;
- * tem precedência sobre os operadores

Os operadores ++ e -- tem precedência sobre o *

Logo:

`*px++;` // incrementa o endereço antes

`*(px--);` // mesma coisa de `*px--`

Passagem de Parâmetros

- Passagem por valor

- O valor do parâmetro é “copiado” para uma variável dentro da função
- O valor original da variável não pode ser alterado

- Passagem por referência

- Através do uso de um ponteiro que aponta para a variável original, é possível ler e alterar o valor original;

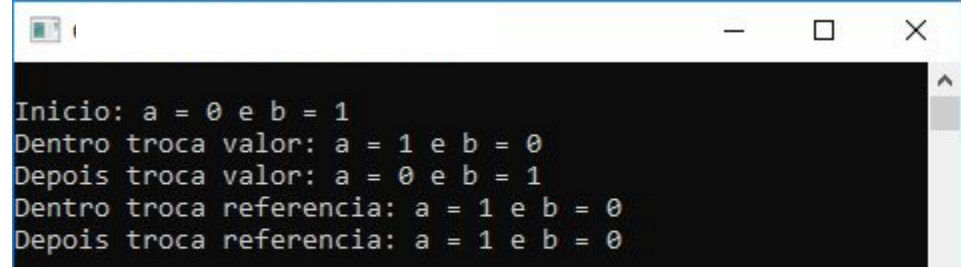
Exemplo de passagem de parâmetros

```
#include <stdio.h>

int trocaValor(int a,int b){
    int temp = a;
    a = b;
    b = temp;
    printf ("\nDentro troca valor: a = %d e b = %d", a, b);
}

int trocaReferencia(int *a,int *b){
    int temp = *a;
    *a = *b;
    *b = temp;
    printf ("\nDentro troca referencia: a = %d e b = %d", *a, *b);
}

int main(){
    int a = 0, b = 1;
    printf ("\nInicio: a = %d e b = %d", a, b);
    trocaValor(a, b);
    printf ("\nDepois troca valor: a = %d e b = %d", a, b);
    trocaReferencia(&a, &b);
    printf ("\nDepois troca referencia: a = %d e b = %d", a, b);
    return 0;
}
```



```
Inicio: a = 0 e b = 1
Dentro troca valor: a = 1 e b = 0
Depois troca valor: a = 0 e b = 1
Dentro troca referencia: a = 1 e b = 0
Depois troca referencia: a = 1 e b = 0
```

Ponteiros e Matrizes

O C possibilita a manipulação de matrizes através de ponteiros

```
int vetor[10];  
  
int *ptr; vetor[0] = 1;  
  
ptr = vetor; //mesmo que &vetor[0]  
  
printf("ptr = %x", ptr);  
  
ptr = &vetor[0];  
  
printf("ptr = %x", ptr);
```

Ponteiros e Matrizes

- O nome da matriz é um ponteiro para a primeira posição
- Um ponteiro que aponta para uma posição da matriz pode ser utilizado como matriz
- Se ptr aponta para um elemento da matriz
 - Então $(ptr+1)$ é o próximo elemento da matriz;
 - E $(ptr-1)$ é o elemento anterior da matriz;
 - E $ptr[0]$ é o elemento apontado pelo ponteiro
 - E $ptr[1]$ é o elemento seguinte.

Exemplo de ponteiros e matrizes

```
#include <stdio.h>

int main() {
    int vet[10] = {0,1,2,3,4,5,6,7,8,9};
    int *ptr;
    ptr = vet;
    printf ("%x\n", ptr);
    printf ("%x\n", &vet[0]);
    printf ("%x\n\n", vet);

    int i;
    for(i = 0; i < 10; i++){
        printf ("%d\n", *(ptr+i));
        printf ("%d\n", ptr[i]);
        printf ("%d\n\n", vet[i]);
    }
    return 0;
}
```


Ponteiros e Strings

Sendo uma string um vetor de char, este terá as mesmas características já mencionadas

```
char texto[20] = "composto";
```

```
char *text = "composto";
```

```
char txt[] = "composto";
```

```
printf("%s %s %s", texto, text, txt);
```

Matrizes de ponteiros

Assim como qualquer outro tipo de dado é possível criar vetores e matrizes de ponteiros

```
int i = 0;
```

```
int *x[10];
```

```
x[2] = &i;
```

```
*x[2] = 20;
```

```
char *erro[] = {"arquivo não encontrado\n", "erro de leitura\n"};
```

```
printf("%s", erro[0]);
```

```
printf("%s", erro[1]);
```

Exemplo de matrizes de ponteiros

```
#include <stdio.h>

int main() {
    int i;
    char *erro[3];
    erro[0] = "Arquivo nao encontrado\n";
    erro[1] = "Erro de Leitura\n";
    erro[2] = "2 - outro erro";
    for(i = 0; i < 3; i++){
        printf ("%s", erro[i]);
    }
    return 0;
}
```

```
Arquivo nao encontrado
Erro de Leitura
2 - outro erro
```

Ponteiros de ponteiros

Serve para armazenar o endereço de um ponteiro

O primeiro ponteiro aponta para o segundo, que aponta para uma variável;

Exemplos

```
int x, *ptr, **ptrptr;  
x=0;  
ptr=&x;  
ptrptr=&ptr;  
*ptr=1;  
**ptrptr=2;
```

```
main() {  
    int x, *p, **q;  
    x=10;  
    p=&x;  
    q=&p;  
    printf("%d", **q) ;  
}
```