UFRJ

INSTITUTO DE COMPUTAÇÃO UFRJ

Programação de Computadores II

Ponteiros e Funções

Profa. Giseli Rabello Lopes

Sumário

- Ponteiros
- Funções
  - Passagem de parâmetros por referência
- Exemplos
- Exercícios

Adaptado de slides preparados por Profa. Silvana Rossetto e Profa. Cora H. F. Pinto Ribeiro

Exercício

- O programa abaixo, quando foi executado, retornou com o erro *Segmentation fault (core dumped)* imediatamente após o usuário digitar o número inteiro solicitado.
  a) O que causou o erro?
  b) Como corrigi-lo?

```
#include <stdio.h>
int main ()
{
        int *i, j;
        puts("Digite um numero inteiro:");
        scanf("%d", i);
        printf("%d\n", *i);
        return 0;
}
```

Endereço **NULL**

- Existe um valor especial para ponteiros conhecido como **valor nulo** ou **NULL**
- O valor **NULL** indica que um ponteiro, intencionalmente, não aponta para nenhuma variável
- Vale lembrar que um ponteiro, quando não inicializado, não necessariamente aponta para **NULL**

Exercício

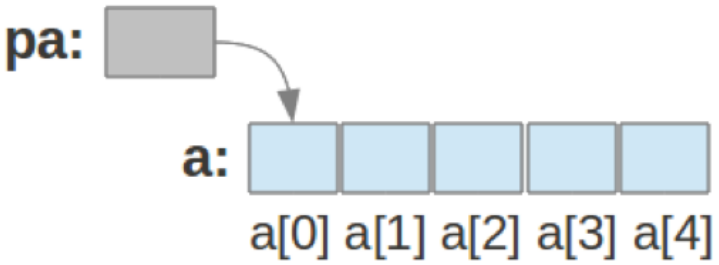
- O que será impresso pelo programa abaixo?

```
#include <stdio.h>
int main () {
        int i, j, *ip, *jp;
        char c, *cp;
        cp = &c;
        c = 'a';
        *cp = 'z';
        printf("c = %c\n", c);
        i = 10;
        ip = &i;
        printf("ip=%p\n", ip);
        printf("*ip=%d\n", *ip);
        jp = NULL;
        *jp = i; //incorreto pois jp não aponta  para uma variável!!!
        return 0;
}
```

Ponteiros e vetores

- Em C, ponteiros e vetores estão fortemente relacionados:
  - Qualquer operação que pode ser feita com vetores também pode ser feita usando ponteiros

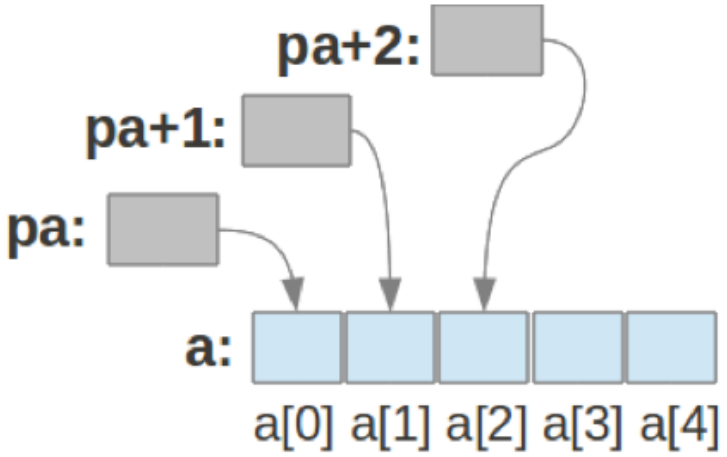
```
//exemplo de definição de vetor
int a[5]; //definiu um vetor a de tamanho 5
int *pa;
pa = &a[0];  //fez pa apontar para o primeiro elemento de a
int x;
x = *pa; //copia conteúdo de a[0] para x
```



Aritmética de ponteiros

- Se **pa** aponta para um elemento de um vetor, então, **pa+1** aponta para o próximo elemento do vetor
- **pa+i** aponta para o **i**-ésimo elemento a partir de **pa**
- **pa-i** aponta para o **i**-ésimo elemento antes de **pa**

```
//..continuando..se pa aponta para a[0], então
x = *(pa+1); //x recebe o valor de a[1]
x = *(pa+i); //x recebe o valor de a[i]
```



Ponteiros e vetores

- Por definição, o valor de uma variável ou expressão do tipo vetor é o endereço do elemento de índice zero do vetor
  - Então fazer **`pa = &a[0];`** é equivalente a fazer **`pa = a;`**
- Quando C avalia **`a[i]`**, a conversão para **`*(a+i)`** é feita imediatamente (i.e., as duas formas são equivalentes)
- Da mesma forma, **`&a[i]`** é equivalente a **`a+i`**

O que não é equivalente entre vetores e ponteiros

- Um ponteiro é uma variável, então fazer **`pa=a`** ou **`pa++`** é permitido!
- O nome de um vetor não é uma variável, então fazer **`a=pa`** ou **`a++`** não é permitido!
- Exemplos:

```
int a[2], *pa, b[2];
a[0]=1; a[1]=2;
pa = a;
printf("a[0]=%d a[1]=%d\n", *(pa), *(pa+1));
pa++;
printf("a[1]=%d\n", *(pa));
//a++; //error: lvalue required as increment operand
//b = pa;  //error: incompatible types when assigning
           //...to type 'int[2]' from type 'int *'
```

Formas de passagem de valores de parâmetros

- Por valor:
  - Passa somente um valor
  - Para cada parâmetro, é alocada uma área de uso local da função e o valor do argumento é copiado para esta área (valor inicial)
  - Alterações deste valor, durante o processamento da função, não alteram o valor da variável enviada pelo programa que chamou a função (argumento ou parâmetro real), apenas a cópia local

Exemplo 1: passagem por valor

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

```
#include <stdio.h>
void troca(int x, int y)
{
      int temp;

      temp = x;
      x = y;
      y = temp;
}
```

```
int main()
{
     int x=5, y=10;

     printf("x=%d y=%d\n",x,y);
     troca(x,y);
     printf("x=%d y=%d\n",x,y);
     return 0;
}
```

O que será impresso na tela?

```
x=5 y=10
x=5 y=10
```

- Os valores de **x** e **y** (globais) não foram trocados, porque a passagem de parâmetros foi feita por valor.
- Passagem de parâmetro por valor:
  - Variável local **x** é inicializada com conteúdo de **x** (argumento);
  - Variável local **y** é inicializada com conteúdo de **y** (argumento);
  - Dentro do subprograma, os valores de **x** e **y** são trocados, mas esta mudança é feita sobre as cópias locais dos valores e não é propagada para fora do subprograma e **x** e **y** permanecem com os valores originais.

Tipos de Funções

- Funções **`void`**: não retornam valor associado à função
  1. Sem parâmetros (parâmetros **`void`**)
  2. Com parâmetros
     1. Passados por valor
     2. Passados por referência (endereço)

Veremos na aula de hoje!

- Funções tipadas ou com retorno: devolvem um único valor associado a execução da função, usando o comando **`return`**
  1. Sem parâmetros (parâmetros **`void`**)
  2. Com parâmetros

- **`void`** é um termo que indica ausência. Em linguagem C é um tipo de dados.

Formas de passagem de valores de parâmetros

- Por referência:
  - Passa um endereço de memória, o qual é associado ao parâmetro, localmente
  - Alterações feitas no parâmetro afetam o conteúdo do endereço referido na chamada (argumento ou parâmetro real)
  - Para passar parâmetros por referência precisamos usar ponteiros

Ponteiros e argumentos de função

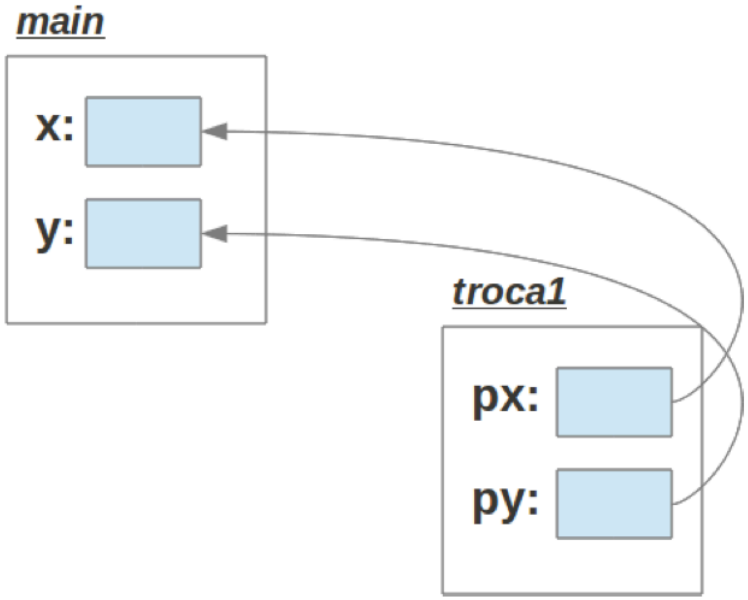
- Usando ponteiros, é possível alterar os argumentos de uma função (**passagem de argumento por referência**) (oposto a passagem de argumento por valor)

```
//exemplo de passagem de argumento por valor!
void troca (int x, int y);

//exemplo de passagem de argumento por referência!
void troca1 (int *px, int *py);
```

Passagem de argumento por referência

- Argumentos do tipo ponteiro permitem que a função acesse (e **modifique**) diretamente os objetos/variáveis da função que a chamou



Exemplo 2: passagem por referência

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

```
#include <stdio.h>
void troca1(int *px, int *py)
{ // conteúdo dos endereços
      int temp;

      temp=*px;
      *px = *py;
      *py = temp;
}
```

```
int main()
{
     int x=5, y=10;

     printf("x=%d y=%d\n",x,y);
     troca1(&x,&y); //aqui, endereços
     printf("x=%d y=%d\n",x,y);
     return 0;
}
```

Passagem de parâmetros:

- Agora **px** e **py** recebem os endereços de **x** e **y**
- Na função, valores são buscados e armazenados no endereço referido, isto é, em **x** e **y**!

```
x=5 y=10
x=10 y=5
```

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