

# Ponteiros e funções

## Aula 10

Diego Padilha Rubert

Faculdade de Computação  
Universidade Federal de Mato Grosso do Sul

Algoritmos e Programação II

# Conteúdo da aula

- 1 Introdução
- 2 Parâmetros de entrada e saída?
- 3 Devolução de ponteiros
- 4 Exercícios

- ▶ aprendemos algumas “regras” de como construir funções que têm parâmetros de entrada e saída ou argumentos passados por referência
- ▶ esses argumentos/parâmetros são na verdade ponteiros
- ▶ o endereço de uma variável é passado como argumento para uma função
- ▶ o parâmetro correspondente que recebe o endereço é então um ponteiro
- ▶ qualquer alteração realizada no conteúdo do parâmetro tem reflexos externos à função, no argumento correspondente

- ▶ aprendemos algumas “regras” de como construir funções que têm parâmetros de entrada e saída ou argumentos passados por referência
- ▶ esses argumentos/parâmetros são na verdade ponteiros
- ▶ o endereço de uma variável é passado como argumento para uma função
- ▶ o parâmetro correspondente que recebe o endereço é então um ponteiro
- ▶ qualquer alteração realizada no conteúdo do parâmetro tem reflexos externos à função, no argumento correspondente

- ▶ aprendemos algumas “regras” de como construir funções que têm parâmetros de entrada e saída ou argumentos passados por referência
- ▶ esses argumentos/parâmetros são na verdade ponteiros
- ▶ o endereço de uma variável é passado como argumento para uma função
- ▶ o parâmetro correspondente que recebe o endereço é então um ponteiro
- ▶ qualquer alteração realizada no conteúdo do parâmetro tem reflexos externos à função, no argumento correspondente

- ▶ aprendemos algumas “regras” de como construir funções que têm parâmetros de entrada e saída ou argumentos passados por referência
- ▶ esses argumentos/parâmetros são na verdade ponteiros
- ▶ o endereço de uma variável é passado como argumento para uma função
- ▶ o parâmetro correspondente que recebe o endereço é então um ponteiro
- ▶ qualquer alteração realizada no conteúdo do parâmetro tem reflexos externos à função, no argumento correspondente

- ▶ aprendemos algumas “regras” de como construir funções que têm parâmetros de entrada e saída ou argumentos passados por referência
- ▶ esses argumentos/parâmetros são na verdade ponteiros
- ▶ o endereço de uma variável é passado como argumento para uma função
- ▶ o parâmetro correspondente que recebe o endereço é então um ponteiro
- ▶ qualquer alteração realizada no conteúdo do parâmetro tem reflexos externos à função, no argumento correspondente

# Parâmetros de entrada e saída?

```
#include <stdio.h>

void troca(int *a, int *b)
{
    int aux;

    aux = *a;
    *a = *b;
    *b = aux;
}

int main(void)
{
    int x, y;

    scanf("%d%d", &x, &y);
    printf("Antes da troca : x = %d e y = %d\n", x, y);
    troca(&x, &y);
    printf("Depois da troca: x = %d e y = %d\n", x, y);
    return 0;
}
```



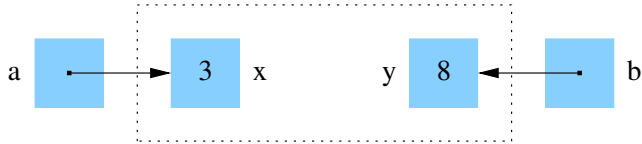
# Parâmetros de entrada e saída?



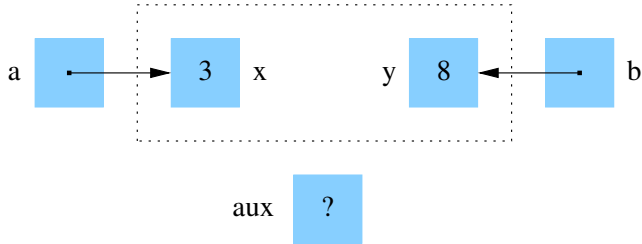
# Parâmetros de entrada e saída?

3 x y 8

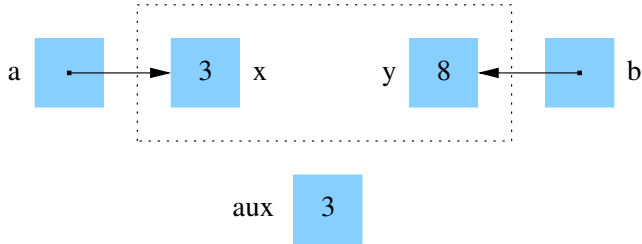
# Parâmetros de entrada e saída?



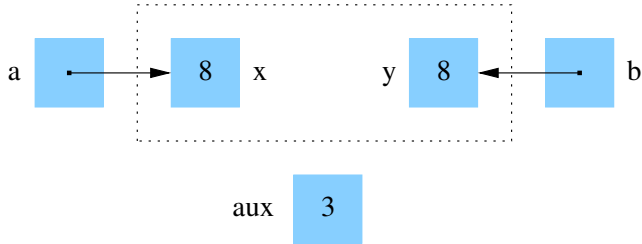
# Parâmetros de entrada e saída?



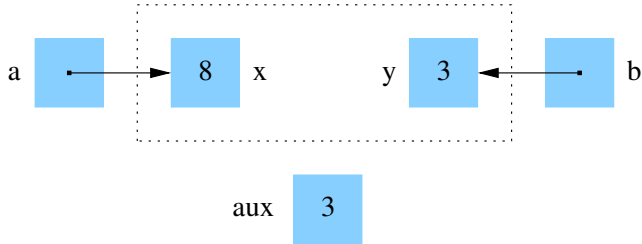
# Parâmetros de entrada e saída?



# Parâmetros de entrada e saída?



# Parâmetros de entrada e saída?



# Parâmetros de entrada e saída?

8 x y 3



# Parâmetros de entrada e saída?

- ▶ esse exemplo destaca que são realizadas **cópias** do valores dos argumentos – que nesse caso são endereços das variáveis da função **main** – para os parâmetros respectivos da função **troca**
- ▶ no corpo dessa função, sempre que usamos o operador de indireção para acessar algum valor, estamos na verdade acessando o conteúdo da variável correspondente dentro da função **main**, que chamou a função **troca**
- ▶ isso ocorre com as variáveis **x** e **y** da função **main**, quando copiamos seus endereços nos parâmetros **a** e **b** da função **troca**

# Parâmetros de entrada e saída?

- ▶ esse exemplo destaca que são realizadas **cópias** do valores dos argumentos – que nesse caso são endereços das variáveis da função **main** – para os parâmetros respectivos da função **troca**
- ▶ no corpo dessa função, sempre que usamos o operador de indireção para acessar algum valor, estamos na verdade acessando o conteúdo da variável correspondente dentro da função **main**, que chamou a função **troca**
- ▶ isso ocorre com as variáveis **x** e **y** da função **main**, quando copiamos seus endereços nos parâmetros **a** e **b** da função **troca**

# Parâmetros de entrada e saída?

- ▶ esse exemplo destaca que são realizadas **cópias** do valores dos argumentos – que nesse caso são endereços das variáveis da função **main** – para os parâmetros respectivos da função **troca**
- ▶ no corpo dessa função, sempre que usamos o operador de indireção para acessar algum valor, estamos na verdade acessando o conteúdo da variável correspondente dentro da função **main**, que chamou a função **troca**
- ▶ isso ocorre com as variáveis **x** e **y** da função **main**, quando copiamos seus endereços nos parâmetros **a** e **b** da função **troca**

# Parâmetros de entrada e saída?

## ► cópia???

- aprendemos que parâmetros passados desta mesma forma são parâmetros de entrada e saída, ou seja, são parâmetros passados por referência e não por cópia
- só há passagem de argumentos por cópia na linguagem C
- não há passagem de argumentos por referência na linguagem C
- simulamos a passagem de um argumento por referência usando ponteiros

# Parâmetros de entrada e saída?

- ▶ cópia???
- ▶ aprendemos que parâmetros passados desta mesma forma são parâmetros de entrada e saída, ou seja, são parâmetros passados por referência e não por cópia
- ▶ só há passagem de argumentos por cópia na linguagem C
- ▶ não há passagem de argumentos por referência na linguagem C
- ▶ simulamos a passagem de um argumento por referência usando ponteiros

# Parâmetros de entrada e saída?

- ▶ cópia???
- ▶ aprendemos que parâmetros passados desta mesma forma são parâmetros de entrada e saída, ou seja, são parâmetros passados por referência e não por cópia
- ▶ só há passagem de argumentos por cópia na linguagem C
- ▶ não há passagem de argumentos por referência na linguagem C
- ▶ simulamos a passagem de um argumento por referência usando ponteiros

# Parâmetros de entrada e saída?

- ▶ cópia???
- ▶ aprendemos que parâmetros passados desta mesma forma são parâmetros de entrada e saída, ou seja, são parâmetros passados por referência e não por cópia
- ▶ só há passagem de argumentos por cópia na linguagem C
- ▶ não há passagem de argumentos por referência na linguagem C
- ▶ simulamos a passagem de um argumento por referência usando ponteiros

# Parâmetros de entrada e saída?

- ▶ cópia???
- ▶ aprendemos que parâmetros passados desta mesma forma são parâmetros de entrada e saída, ou seja, são parâmetros passados por referência e não por cópia
- ▶ só há passagem de argumentos por cópia na linguagem C
- ▶ não há passagem de argumentos por referência na linguagem C
- ▶ simulamos a passagem de um argumento por referência usando ponteiros



# Parâmetros de entrada e saída?

- ▶ ao passar (por cópia) o endereço de uma variável como argumento para uma função, o parâmetro correspondente deve ser um ponteiro e, mais que isso, um ponteiro para a variável correspondente cujo endereço foi passado como argumento
- ▶ qualquer modificação indireta realizada no corpo dessa função usando esse ponteiro será realizada na verdade no conteúdo da variável apontada pelo parâmetro, que é simplesmente o conteúdo da variável passada como argumento na chamada da função
- ▶ não há nada de errado com o que aprendemos nas aulas anteriores sobre argumentos de entrada e saída, isto é, passagem de argumentos por referência
- ▶ passagem de argumentos por referência é um tópico conceitual quando falamos da linguagem de programação C

# Parâmetros de entrada e saída?

- ▶ ao passar (por cópia) o endereço de uma variável como argumento para uma função, o parâmetro correspondente deve ser um ponteiro e, mais que isso, um ponteiro para a variável correspondente cujo endereço foi passado como argumento
- ▶ qualquer modificação indireta realizada no corpo dessa função usando esse ponteiro será realizada na verdade no conteúdo da variável apontada pelo parâmetro, que é simplesmente o conteúdo da variável passada como argumento na chamada da função
- ▶ não há nada de errado com o que aprendemos nas aulas anteriores sobre argumentos de entrada e saída, isto é, passagem de argumentos por referência
- ▶ passagem de argumentos por referência é um tópico conceitual quando falamos da linguagem de programação C

# Parâmetros de entrada e saída?

- ▶ ao passar (por cópia) o endereço de uma variável como argumento para uma função, o parâmetro correspondente deve ser um ponteiro e, mais que isso, um ponteiro para a variável correspondente cujo endereço foi passado como argumento
- ▶ qualquer modificação indireta realizada no corpo dessa função usando esse ponteiro será realizada na verdade no conteúdo da variável apontada pelo parâmetro, que é simplesmente o conteúdo da variável passada como argumento na chamada da função
- ▶ não há nada de errado com o que aprendemos nas aulas anteriores sobre argumentos de entrada e saída, isto é, passagem de argumentos por referência
- ▶ passagem de argumentos por referência é um tópico conceitual quando falamos da linguagem de programação C

# Parâmetros de entrada e saída?

- ▶ ao passar (por cópia) o endereço de uma variável como argumento para uma função, o parâmetro correspondente deve ser um ponteiro e, mais que isso, um ponteiro para a variável correspondente cujo endereço foi passado como argumento
- ▶ qualquer modificação indireta realizada no corpo dessa função usando esse ponteiro será realizada na verdade no conteúdo da variável apontada pelo parâmetro, que é simplesmente o conteúdo da variável passada como argumento na chamada da função
- ▶ não há nada de errado com o que aprendemos nas aulas anteriores sobre argumentos de entrada e saída, isto é, passagem de argumentos por referência
- ▶ passagem de argumentos por referência é um tópico conceitual quando falamos da linguagem de programação C

# Devolução de ponteiros

- ▶ além de passar ponteiros como argumentos para funções também podemos fazê-las devolver ponteiros
- ▶ comuns, por exemplo, quando tratamos de cadeias de caracteres
- ▶ não é possível que uma função devolva o endereço de uma variável local sua, já que ao final de sua execução, essa variável será destruída

# Devolução de ponteiros

- ▶ além de passar ponteiros como argumentos para funções também podemos fazê-las devolver ponteiros
- ▶ comuns, por exemplo, quando tratamos de cadeias de caracteres
- ▶ não é possível que uma função devolva o endereço de uma variável local sua, já que ao final de sua execução, essa variável será destruída

# Devolução de ponteiros

- ▶ além de passar ponteiros como argumentos para funções também podemos fazê-las devolver ponteiros
- ▶ comuns, por exemplo, quando tratamos de cadeias de caracteres
- ▶ não é possível que uma função devolva o endereço de uma variável local sua, já que ao final de sua execução, essa variável será destruída

# Devolução de ponteiros

```
int *max(int *a, int *b)
{
    if (*a > *b)
        return a;
    else
        return b;
}
```

```
int i, j, *p;
:
:
p = max(&i, &j);
```



# Devolução de ponteiros

```
int *max(int *a, int *b)
{
    if (*a > *b)
        return a;
    else
        return b;
}
```

```
int i, j, *p;
:
:
p = max(&i, &j);
```

1. (a) Escreva uma função com a seguinte interface:

```
void min_max(int n, int v[MAX], int *max, int *min)
```

que receba um número inteiro  $n$ , com  $1 \leq n \leq 100$ , e um vetor  $v$  com  $n > 0$  números inteiros e devolva um maior e um menor dos elementos desse vetor.

- (b) Escreva um programa que receba  $n > 0$  números inteiros, armazene-os em um vetor e, usando a função do item (a), mostre na saída um maior e um menor elemento desse conjunto. Simule no papel a execução de seu programa antes de implementá-lo.

2. (a) Escreva uma função com a seguinte interface:

```
void dois_maiores(int n, int v[MAX], int *pr, int *seg)
```

que receba um número inteiro  $n$ , com  $1 \leq n \leq 100$ , e um vetor  $v$  com  $n > 0$  números inteiros e devolva um maior e um segundo maior elementos desse vetor.

- (b) Escreva um programa que receba  $n > 0$  números inteiros, armazene-os em um vetor e, usando a função do item (a), mostre na saída um maior e um segundo maior elemento desse conjunto. Simule no papel a execução de seu programa antes de implementá-lo.

3. (a) Escreva uma função com a seguinte interface:

```
void soma_prod(int a, int b, int *soma, int *prod)
```

que receba dois números inteiros  $a$  e  $b$  e devolva a soma e o produto destes dois números.

- (b) Escreva um programa que receba  $n$  números inteiros, com  $n > 0$  par, calcule a soma e o produto deste conjunto usando a função do item (a) e determine quantos deles são maiores que esta soma e quantos são maiores que o produto. Observe que os números na entrada podem ser negativos. Simule no papel a execução de seu programa antes de implementá-lo.

4. (a) Escreva uma função com a seguinte interface:

```
int *maximo(int n, int v[MAX])
```

que receba um número inteiro  $n$ , com  $1 \leq n \leq 100$ , e um vetor  $v$  de  $n$  números inteiros e devolva o endereço do elemento de  $v$  onde reside um maior elemento de  $v$ .

- (b) Escreva um programa que receba  $n > 0$  números inteiros, armazene-os em um vetor e, usando a função do item (a), mostre na saída um maior elemento desse conjunto. Simule no papel a execução de seu programa antes de implementá-lo.

