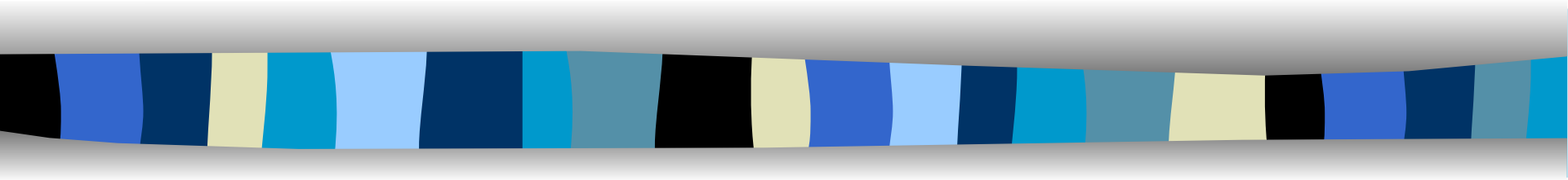
Ponteiros



João Frederico Roldan Viana

jfredrv@gmail.com

(85)99231.2777

Agenda

- Ponteiros
  - Definições
  - Declaração de Ponteiros
  - Operadores de Ponteiros
    - Operador de endereço
    - Operador de conteúdo
  - Atribuição
  - Operações elementares
  - Ponteiros e Vetores
  - Ponteiros e Funções
  - Alocação Dinâmica de Memória

Ponteiros

- Definições
  - Ponteiros são variáveis que contém endereços. Neste sentido, estas variáveis apontam para algum determinado endereço da memória.
  - Em geral, o ponteiro aponta para o endereço de alguma variável de tipo conhecido declarada no programa.

Ponteiros

- Declaração de Ponteiros
  - Quando declaramos um ponteiro, devemos declará-lo com o mesmo tipo (int, char, etc.) do bloco a ser apontado. Por exemplo, se queremos que um ponteiro aponte para uma variável int (bloco de 2 bytes) devemos declará-lo como int também.
  - Sintaxe:

    *tipo_ptr *nome_ptr;*

    tipo_ptr nome_ptr1, nome_ptr2, nome_ptr3 . . . ;*

Ponteiros

- Declaração de Ponteiros
  - Exemplos:

    ```
    int *ptr1;

    int* ptr2, ptr3;
    ```

  - A primeira instrução declara um ponteiro chamado *ptr1* que aponta para um inteiro. Este ponteiro aponta para o primeiro endereço de um bloco de dois bytes.
  - A segunda instrução declara dois ponteiros (*ptr2* e *ptr3*) do tipo *int*. Observe que o * está justaposto ao tipo, assim todos os elementos da lista serão declarados ponteiros.

Ponteiros

- Operadores de Ponteiros
  - Quando trabalhamos com ponteiros, queremos, basicamente, fazer duas coisas:
    - Conhecer o endereço de uma variável (operador de endereço &)
    - Conhecer o conteúdo de um endereço (operador de conteúdo *)

Ponteiros

- Operador de endereço (&)
  - Determina o endereço de uma variável (o primeiro byte do bloco ocupado pela variável).
  - Por exemplo, *&val* determina o endereço do bloco ocupado pela variável *val*.
  - O operador de endereço (&) somente pode ser usado em uma única variável. Não pode ser usado em expressões como, por exemplo, *&(a+b)*.

Ponteiros

- Operador de endereço (&)
  - Quando escreve-se a instrução scanf("%d", &*num*), estamos nos referimos ao endereço do bloco ocupado pela variável *num*.
  - A instrução acima significa: “leia o buffer do teclado, transforme o valor lido em um valor inteiro (2 bytes) e o armazene no bloco localizado no endereço da variável *num*”.

Ponteiros

- Operador de conteúdo (*)
  - Determina o conteúdo (valor) do dado armazenado no endereço de um bloco apontado por um ponteiro.
  - Por exemplo, **p* determina conteúdo do bloco apontado pelo ponteiro *p*.
  - De forma resumida: o operador (*) determina o conteúdo de um endereço.
  - O operador de conteúdo (*) somente pode ser usado em variáveis do tipo ponteiros.

Ponteiros

Exercício 1

- Atribuição
  - Para se atribuir a um ponteiro o endereço de uma variável escreve-se:

```
int *p, val = 5; // declaração de ponteiro e variável
p = &val;        // atribuição
```

  - Para se atribuir a uma variável o conteúdo de um endereço escreve-se:

```
int *p = 0x3f8, val; // declaração de ponteiro e variável
val = *p;            // atribuição
```

Ponteiros

- Operações Elementares
  - A um ponteiro pode ser atribuído o endereço de uma variável comum.

```
...
int *p;
int s;
p = &s;   // p recebe o endereço de s
...
```

Ponteiros

Exercício 2

- Operações Elementares
  - Um ponteiro pode receber o valor de outro ponteiro, isto é, pode receber o endereço apontado por outro ponteiro, desde que os ponteiros sejam de mesmo tipo.

```
...
int *p1, *p2, num = 10;
p1 = &num; // p1 recebe o endereço de num
p2 = p1;      // p2 recebe o conteúdo de p1
              //  (endereço de num)
...
```

Ponteiros

Exercício 3

- Operações Elementares
  - Um ponteiro pode receber um endereço de memória diretamente. Um endereço é um numero inteiro. Em geral, na forma hexadecimal (0x....).
  - Para usar o valor atribuido devemos, em geral, forçar uma conversão de tipo usando casting do tipo de ponteiro declarado para o inteiro.

```
...
float *p1;
p1 = 0x03F8;            // endereço da porta serial COM1
printf("%d", (int)p1); // casting
...
```

Ponteiros

Exercício 4

- Operações Elementares
  - A um ponteiro pode ser atribuído o valor nulo usando a constante simbólica *NULL* (declarada na biblioteca stdlib.h).
  - Um ponteiro com valor *NULL* não aponta para lugar nenhum.

```
#include <stdlib.h>
...
char *p;
p = NULL;
...
```

Ponteiros

Exercício 5

- Operações Elementares
  - Uma quantidade inteira pode ser adicionada ou subtraída de um ponteiro. A adição de um inteiro *n* a um ponteiro *p* fará com que ele aponte para o endereço do *n-ésimo* bloco seguinte.

```
...
double *p1, *p2, num;
p1 = &num;
p2 = p1 + 5;  // p2 aponta para o quinto bloco
                  // posterior a p1

...
p1 = p2++;   // p1 aponta para o p2 e p2 aponta para
             // o bloco posterior a ele

...
```

Ponteiros

- Operações Elementares
  - Dois ponteiros podem ser comparados (usando-se operadores lógicos) desde que sejam de mesmo tipo.

```
...
if(px == py){
    // se px aponta para o mesmo bloco que py
if(px > py){
    // se px aponta para um bloco posterior a py
if(px != py){
    // se px aponta para um bloco diferente de py
if(px == NULL){
    // se px é nulo
...
```

Ponteiros

Exercícios 6 e 7

- Ponteiros e Vetores
  - Em C, o nome de um vetor é tratado como o endereço de seu primeiro elemento.
  - Por exemplo, se vet é um vetor, então vet e &vet[0] representam o mesmo endereço.
  - Podemos acessar o endereço de qualquer elemento do vetor da seguinte forma: &vet[i] que é equivalente a (vet + i). Vale ressaltar que (vet + i) não representa uma adição aritmética normal mas o endereço do *i-ésimo* elemento do vetor vet.
  - Do mesmo modo, podemos acessar o conteúdo de qualquer elemento do vetor da seguinte forma:
    *(vet + i) que é equivalente a vet[i].

Ponteiros

- Ponteiros e Funções
  - Passagem de parâmetros por referência
    - Significa que passamos como parâmetro para uma função o endereço de uma variável, isto é, a função chamada recebe a localização na memória da variável através de um ponteiro.
    - Assim qualquer alteração no conteúdo apontado pelo do ponteiro será uma alteração no conteúdo da variável original. O valor original é alterado.
    - Permite que (formalmente) uma função retorne quantos valores se desejar.

Ponteiros

- Ponteiros e Funções
  - Passagem de parâmetros por referência
    - Sintaxe na função chamada:

      tipof nomef(*tipop nomep*){ . . . }

      onde,

      tipof: tipo de retorno da função.

      nomef: nome da função a ser chamada.

      tipop: tipo do ponteiro (igual ao tipo da variável passada).

      nomep: nome do ponteiro.

Ponteiros

- Ponteiros e Funções
  - Passagem de parâmetros por referência
    - Sintaxe para chamar a função:

      nomef(*end_var*);

      onde,

      nomef: nome da função chamada.

      end_var: endereço da variável passada como argumento.

Ponteiros

Exercício 8

- Ponteiros e Funções
  - Passagem de parâmetros por referência
    - Exemplo:

```
void troca(int *p1, int *p2){
    . . .
}
void main(){
    . . .
    troca(&a, &b);
    . . .
}
```

Ponteiros

- Alocação Dinâmica de Memória
  - A linguagem C permite alocar dinamicamente (em tempo de execução), blocos de memória usando ponteiros.
  - Isto é desejável caso queiramos poupar memória, isto é não reservar mais memória que o necessário para o armazenamento de dados.
  - Dada a relação entre ponteiros e vetores, isto significa que podemos declarar dinamicamente vetores de tamanho variável.

Ponteiros

- Alocação Dinâmica de Memória
  - Para a alocação de memória usamos a função malloc() – *memory allocation.*
  - Para liberar (desalocar) o espaço de memória se usa a função free().
  - *Para a realocação de memória usamos a função realloc() – memory reallocation.*

Ponteiros

- Alocação Dinâmica de Memória
  - malloc()
    - Sintaxe: *pont* = (*tipo**)malloc(*tam*);

      pont: nome do ponteiro que recebe o endereço do espaço de memória alocado;

      tipo: tipo do endereço apontado (tipo do ponteiro);

      tam: tamanho do espaço alocado, ou seja,é o numero de *bytes*.
    - Caso não seja possível alocar o espaço requisitado a função malloc() retorna a constante simbólica *NULL*.

Ponteiros

- Alocação Dinâmica de Memória
  - malloc()
    - Exemplo:

```
. . .
int num;
int *vet;
scanf("%d", &num);
vet = (int*) malloc(num * 4);
. . .
```

Ponteiros

Exercício 9

- Alocação Dinâmica de Memória
  - malloc()
    - Exemplo:

```
. . .
int num;
int *vet;
scanf("%d", &num);
vet = (int*) malloc(num * sizeof(int));
. . .
```

Ponteiros

- Alocação Dinâmica de Memória
  - free()
    - Sintaxe: free(*pont*);

      pont: nome do ponteiro que contem o endereço do inicio do espaço de memória reservado.

    - Exemplo:

```
int = *vet;
. . .
vet = (int*) malloc(num * sizeof(int));
. . .
free(vet);
```

Ponteiros

- Realocação Dinâmica de Memória
  - realloc()
    - Sintaxe: *pont* = (*tipo**)realloc(*pont*, *tam*);

      pont: nome do ponteiro que recebe o endereço do espaço de memória alocado;

      tipo: tipo do endereço apontado (tipo do ponteiro);

      tam: tamanho do espaço alocado, ou seja,é o numero de *bytes*.
    - Caso não seja possível alocar o espaço requisitado a função realloc() retorna a constante simbólica *NULL*.

Ponteiros

- Realocação Dinâmica de Memória
  - realloc()
    - Exemplo:

```
int = *vet;
. . .
vet = (int*) malloc(num * sizeof(int));
. . .
vet = (int*) realloc(vet, num * sizeof(int));
. . .
free(vet);
```