IP
Funções - Parte II

Hebert Coelho

Instituto de Informática
Universidade Federal de Goiás

Roteiro

- Relembrando Ponteiros

Roteiro

- Relembrando Ponteiros
- Relembrando Funções

Roteiro

- Relembrando Ponteiros
- Relembrando Funções
- Protótipos de Funções

Roteiro

- Relembrando Ponteiros
- Relembrando Funções
- Protótipos de Funções
- Passagem de argumentos por valor e por referência

Roteiro

- Relembrando Ponteiros
- Relembrando Funções
- Protótipos de Funções
- Passagem de argumentos por valor e por referência
- Vetores em funções

Roteiro

- Relembrando Ponteiros
- Relembrando Funções
- Protótipos de Funções
- Passagem de argumentos por valor e por referência
- Vetores em funções
- Exemplos

Roteiro

- Relembrando Ponteiros
- Relembrando Funções
- Protótipos de Funções
- Passagem de argumentos por valor e por referência
- Vetores em funções
- Exemplos
- Exercícios

Ponteiros

Definição:

Um *ponteiro* é uma variável que contém um endereço de memória.

Ponteiros

Definição:

Um *ponteiro* é uma variável que contém um endereço de memória.

Declaração:

Uma declaração de ponteiro consiste no tipo de base, um $*$ e o nome da variável.

Ponteiros

Definição:

Um *ponteiro* é uma variável que contém um endereço de memória.

Declaração:

Uma declaração de ponteiro consiste no tipo de base, um $*$ e o nome da variável.

Sintaxe:

```
tipo *nome;
```

Ponteiros

Definição:

Um *ponteiro* é uma variável que contém um endereço de memória.

Declaração:

Uma declaração de ponteiro consiste no tipo de base, um $*$ e o nome da variável.

Sintaxe:

```
tipo *nome;
```

Exemplos:

```
int *valor;
char *letra;
```

Ponteiros

Operadores: * e &.

O operador & devolve o endereço na memória. & pode ser lido como "o endereço de";
O operador * devolve o valor da variável localizada no endereço que segue.

Ponteiros

Operadores: * e &.

O operador & devolve o endereço na memória. & pode ser lido como "o endereço de";
O operador * devolve o valor da variável localizada no endereço que segue.

Exemplo:

```
int a;          int *p;          p = &a;
```

Ponteiros

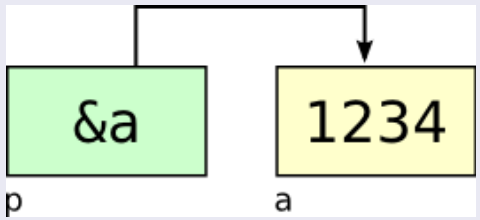
Operadores: * e &.

O operador & devolve o endereço na memória. & pode ser lido como "o endereço de";
O operador * devolve o valor da variável localizada no endereço que segue.

Exemplo:

```
int a;          int *p;          p = &a;
```

O que ocorre na memória:



Inicialização de Ponteiros

- Assim como as variáveis, os ponteiros tem que ser inicializados pois podem conter um valor desconhecido.

Inicialização de Ponteiros

- Assim como as variáveis, os ponteiros tem que ser inicializados pois podem conter um valor desconhecido.
- É comum o uso do valor nulo para inicializar ponteiros.

Inicialização de Ponteiros

- Assim como as variáveis, os ponteiros tem que ser inicializados pois podem conter um valor desconhecido.
- É comum o uso do valor nulo para inicializar ponteiros.

Exemplos

```
int y = 0;
int *p1 = &y;          // ponteiro p1 inicializado com endereço de y
int *p2 = NULL;        // ponteiro inicializado com valor nulo
char *p = "Este ponteiro aponta para o inicio desta frase";
```

Inicialização de Ponteiros

- Assim como as variáveis, os ponteiros tem que ser inicializados pois podem conter um valor desconhecido.
- É comum o uso do valor nulo para inicializar ponteiros.

Exemplos

```
int y = 0;
int *p1 = &y;          // ponteiro p1 inicializado com endereço de y
int *p2 = NULL;        // ponteiro inicializado com valor nulo
char *p = "Este ponteiro aponta para o inicio desta frase";
```

Exercício

1-Faça um programa que imprima o conteúdo do ponteiro *p criado acima.

Função

Definição

São estruturas que agrupam um conjunto de comandos, que são executados quando a função é chamada, e podem retornar um valor ao final de sua execução.

Função

Definição

São estruturas que agrupam um conjunto de comandos, que são executados quando a função é chamada, e podem retornar um valor ao final de sua execução.

Exemplo:

```
x=sqrt(4);
scanf(“%d”, &x);
```

Por que utilizar funções?

- Evitar que os blocos do programa fiquem grandes demais;

Por que utilizar funções?

- Evitar que os blocos do programa fiquem grandes demais;
- Dividir o programa em partes mais fáceis de serem compreendidas;

Por que utilizar funções?

- Evitar que os blocos do programa fiquem grandes demais;
- Dividir o programa em partes mais fáceis de serem compreendidas;
- Permitir o reaproveitamento de código já construído;

Por que utilizar funções?

- Evitar que os blocos do programa fiquem grandes demais;
- Dividir o programa em partes mais fáceis de serem compreendidas;
- Permitir o reaproveitamento de código já construído;
- Evitar que um trecho de código seja repetido várias vezes.

Declarando uma função

Uma função é declarada da seguinte forma:

```
tipo nome (tipo parâmetro1, ..., tipo parâmetroN)
     comandos;
     return valor de retorno;
}
```

Exemplos:

A função abaixo soma dois valores, passados como parâmetro:

```
int soma (int a, int b) {
     return (a + b);
}
```

O resultado da chamada de uma função é uma expressão:

Exemplos:

A função abaixo soma dois valores, passados como parâmetro:

```
int soma (int a, int b) {
     return (a + b);
}
```

O resultado da chamada de uma função é uma expressão:

```
x = soma(4, 2);
printf(“Soma de a e b:  %d \ n”, soma(a, b));
```

Variáveis globais e locais

- Variável local é declarada dentro de uma função.

Variáveis globais e locais

- Variável local é declarada dentro de uma função.
- Variável global é declarada fora de qualquer função.

Variáveis globais e locais

- Variável local é declarada dentro de uma função.
- Variável global é declarada fora de qualquer função.
- As variáveis locais são visíveis apenas na função onde foram declaradas.

Variáveis globais e locais

- Variável local é declarada dentro de uma função.
- Variável global é declarada fora de qualquer função.
- As variáveis locais são visíveis apenas na função onde foram declaradas.
- As variáveis globais são visíveis por todas as funções.

Protótipo de funções

Tente executar o código abaixo. Funcionou? Porque?

```
#include<stdio.h>
int var=10;
int main () {
     naoMudaVar();
     muda();
     printf(“\n%d”,var);
}

void naoMudaVar(){  int var=100; }

void muda(){  var=var+200; }
```

Protótipo de funções

- Para organizar melhor um programa podemos declarar uma função sem implementá-la ou defini-la.

Protótipo de funções

- Para organizar melhor um programa podemos declarar uma função sem implementá-la ou defini-la.
- Para declarar uma função sem a sua implementação. Substituímos as chaves e seu conteúdo por ponto-e-virgula.

Protótipo de funções

- Para organizar melhor um programa podemos declarar uma função sem implementá-la ou defini-la.
- Para declarar uma função sem a sua implementação. Substituímos as chaves e seu conteúdo por ponto-e-virgula.

```
tipo nome (tipo parâmetro1, ..., tipo parâmetroN);
```

Protótipo de funções

- Para organizar melhor um programa podemos declarar uma função sem implementá-la ou defini-la.
- Para declarar uma função sem a sua implementação. Substituímos as chaves e seu conteúdo por ponto-e-virgula.

```
tipo nome (tipo parâmetro1, ..., tipo parâmetroN);
```

1. A declaração de uma função deve vir sempre antes do seu uso.

Protótipo de funções

- Para organizar melhor um programa podemos declarar uma função sem implementá-la ou defini-la.
- Para declarar uma função sem a sua implementação. Substituímos as chaves e seu conteúdo por ponto-e-virgula.

```
tipo nome (tipo parâmetro1, ..., tipo parâmetroN);
```

1. A declaração de uma função deve vir sempre antes do seu uso.
2. A sua definição pode aparecer em qualquer lugar do programa.

Passagem de argumentos por valor

- Quando passamos argumentos a uma função, os valores fornecidos são copiados para os parâmetros da função. Este processo é idêntico a uma atribuição.

Passagem de argumentos por valor

- Quando passamos argumentos a uma função, os valores fornecidos são copiados para os parâmetros da função. Este processo é idêntico a uma atribuição.
- Desta forma, alterações nos parâmetros dentro da função não alteram os valores que foram passados.

Passagem de argumentos por valor

- Quando passamos argumentos a uma função, os valores fornecidos são copiados para os parâmetros da função. Este processo é idêntico a uma atribuição.
- Desta forma, alterações nos parâmetros dentro da função não alteram os valores que foram passados.

Exemplo

```
void nao_troca(int x, int y) {
      int aux;
      aux = x;
      x = y;
      y = aux;
}
```

Passagem de argumentos por referência

- Existe uma forma de alterarmos as variáveis passadas como argumento, ao invés de usarmos apenas o seu valor.

Passagem de argumentos por referência

- Existe uma forma de alterarmos as variáveis passadas como argumento, ao invés de usarmos apenas o seu valor.
- O artifício é passarmos como argumento o endereço da variável, e não o seu valor.

Passagem de argumentos por referência

- Existe uma forma de alterarmos as variáveis passadas como argumento, ao invés de usarmos apenas o seu valor.
- O artifício é passarmos como argumento o endereço da variável, e não o seu valor.
- Para indicarmos que será passado o endereço do argumento, usamos o mesmo tipo que usamos para declarar um variável que guarda um endereço.

Passagem de argumentos por referência

- Existe uma forma de alterarmos as variáveis passadas como argumento, ao invés de usarmos apenas o seu valor.
- O artifício é passarmos como argumento o endereço da variável, e não o seu valor.
- Para indicarmos que será passado o endereço do argumento, usamos o mesmo tipo que usamos para declarar um variável que guarda um endereço.

Exemplo

```
tipo nome (tipo *parâmetro1, ..., tipo *parâmetroN) {
      comandos;
}
```

Passagem de argumentos por referência

- Um endereço de uma variável passado como parâmetro não é muito útil. Para acessarmos o valor de uma variável apontada por um endereço, usamos o operador *;

Passagem de argumentos por referência

- Um endereço de uma variável passado como parâmetro não é muito útil. Para acessarmos o valor de uma variável apontada por um endereço, usamos o operador *;
- Ao precedermos uma variável que contém um endereço com este operador, obtemos o equivalente a variável armazenada no endereço em questão;

Passagem de argumentos por referência

- Um endereço de uma variável passado como parâmetro não é muito útil. Para acessarmos o valor de uma variável apontada por um endereço, usamos o operador *;
- Ao precedermos uma variável que contém um endereço com este operador, obtemos o equivalente a variável armazenada no endereço em questão;

Exemplo

```
void troca(int *x, int *y) {
      int aux;
      aux = *x;
      *x = *y;
      *y = aux;
}
```

Passagem de argumentos por referência

- Uma outra forma de conseguirmos alterar os valores de variáveis externas a funções é usando variáveis globais;

Passagem de argumentos por referência

- Uma outra forma de conseguirmos alterar os valores de variáveis externas a funções é usando variáveis globais;
- Nesta abordagem usamos variáveis globais no lugar de parâmetros e de valores de retorno;

Passagem de argumentos por referência

- Uma outra forma de conseguirmos alterar os valores de variáveis externas a funções é usando variáveis globais;
- Nesta abordagem usamos variáveis globais no lugar de parâmetros e de valores de retorno;
- Porém, ao usar esta técnica estamos negando uma das principais vantagens de se usar funções, reaproveitamento de código.

Vetores em funções

- Vetores têm um comportamento diferente quando usados como parâmetros ou valores de retorno de funções;

Vetores em funções

- Vetores têm um comportamento diferente quando usados como parâmetros ou valores de retorno de funções;
- Por padrão, um vetor é interpretado pelo compilador como o endereço do primeiro elemento do vetor;

Vetores em funções

- Vetores têm um comportamento diferente quando usados como parâmetros ou valores de retorno de funções;
- Por padrão, um vetor é interpretado pelo compilador como o endereço do primeiro elemento do vetor;
- Desta forma, sem precisarmos usar uma notação especial, os vetores são sempre passados por referência.

Vetores em funções

- Vetores têm um comportamento diferente quando usados como parâmetros ou valores de retorno de funções;
- Por padrão, um vetor é interpretado pelo compilador como o endereço do primeiro elemento do vetor;
- Desta forma, sem precisarmos usar uma notação especial, os vetores são sempre passados por referência.

Maneiras para declarar um parâmetro que receberá um vetor

```
void display(int num[10])

void display(int num[])

void display(int *num) // Forma mais usada
```

Vetores em funções

- Ao passar um vetor como parâmetro não é necessário fornecer o seu tamanho na declaração da função. Porém, é importante lembrar que o vetor tem um tamanho que deve ser considerado.

Vetores em funções

- Ao passar um vetor como parâmetro não é necessário fornecer o seu tamanho na declaração da função. Porém, é importante lembrar que o vetor tem um tamanho que deve ser considerado.
- Quando o vetor é multi-dimensional a possibilidade de não informar o tamanho na declaração se restringe apenas a primeira dimensão.

Vetores em funções

- Ao passar um vetor como parâmetro não é necessário fornecer o seu tamanho na declaração da função. Porém, é importante lembrar que o vetor tem um tamanho que deve ser considerado.
- Quando o vetor é multi-dimensional a possibilidade de não informar o tamanho na declaração se restringe apenas a primeira dimensão.

Maneiras para declarar um parâmetro que receberá um vetor

```
void display(int num[10])

void display(int num[])

void display(int *num) // Forma mais usada
```

Exercícios

Faça um programa que ordene um vetor, 1 em ordem crescente, 2 em ordem decrescente. O seu programa deverá fazer uso de duas funções, a função troca(int *x, int*y) e a função int ordena(int *vetor, int tam, int opcao). A função troca(int *x, int *y) troca os valores das variáveis apontadas por x e y. A função int ordena(int *vetor, int tam, int opcao), ordena o vetor com dimensão tam na opção passada como parâmetro, se opcao=1 crescente, se opcao=2 decrescente. A função troca retorna 0 caso consiga ordenar corretamente, 1 caso a opção seja inválida.

Faça um programa para armazenar os dados de distâncias entre cidades e mostrar a distância entre duas cidades desejadas. O seu programa deverá fazer uso de uma função para armazenar as distâncias entre cidades, e também uma função para imprimir a distância entre as cidades desejadas. Suponha que existem no máximo 10 cidades e que o usuário irá digitar as distâncias.
Exemplo de entrada:
Digite a quantidade de cidade desejadas: 4
Digite a distância da cidade 0 para a cidade 0: 0
Digite a distância da cidade 0 para a cidade 1: 56
Digite a distância da cidade 0 para a cidade 2: 12
Digite a distância da cidade 0 para a cidade 3: 49
...