Curso: Ciência da Computação

Disciplina: Estrutura de Dados 1

Professor: Clayton Zambon

centro universitário unifacvest

1. Introdução
1.1. Ponteiros

1. Introdução
1.1. Ponteiros

A utilização de ponteiros em linguagem C é uma das características que tornam a linguagem tão flexível e poderosa.

Ponteiros ou apontadores, são variáveis que armazenam o endereço de memória de outras variáveis.

Dizemos que um ponteiro “aponta” para uma variável quando contém o endereço da mesma.

Os ponteiros podem apontar para qualquer tipo de variável. Portanto temos ponteiros para int, float, double, char, etc .

1. Introdução
1.1. Ponteiros

Por que usar ponteiros?

Ponteiros são muito úteis quando uma variável tem que ser acessada em diferentes partes de um programa.

Neste caso, o código pode ter vários ponteiros espalhados por diversas partes do programa, "apontando" para a variável que contém o dado desejado.

Caso este dado seja alterado, não há problema algum, pois todas as partes do programa tem um ponteiro que aponta para o endereço onde reside o dado atualizado.

Existem várias situações onde ponteiros são úteis, por exemplo:

Alocação dinâmica de memória:

- Manipulação de arrays;
- Para retornar mais de um valor em uma função;
- Referência para listas, pilhas, árvores e grafos.

1. Introdução
1.1. Ponteiros

Analogia de Ponteiros

Pense em ponteiros como sendo aquele colega de trabalho “sacana” que não sabe nada a não ser apontar para você quando alguma pergunta é feita para ele. Quando seu chefe pergunta qualquer coisa o seu colega aponta para você responder, afinal é você quem têm na memória as informações.

Podemos dizer que ponteiros ou apontadores (também podemos nos referir a eles desta forma), são variáveis que armazenam endereços de memória.

1. Introdução
1.1. Ponteiros

1. Introdução
1.1. Ponteiros

Sintaxe de declaração de ponteiro

```
tipo *nome_ponteiro;
```

Onde temos:

tipo : é o tipo de dado da variável cujo endereço o ponteiro armazena.

*nome_ponteiro : O nome da variável ponteiro.

O asterisco * neste tipo de declaração determina que a variável será um ponteiro.

Exemplo de declaração de ponteiro:

```
int *ptr;
```

1. Introdução
1.1. Ponteiros

Operadores para ponteiros

&

Utilizamos este símbolo para capturar o endereço de memória da variável desejada.

- Exemplo: Int x = 10

  printf("Valor da variável X: %i\n", x);

  printf("Endereço de memória da variável X: %i", &x);

*****

Utilizamos este símbolo para capturar o valor do ponteiro.

- Exemplo: int *ponteiro = x;

  printf("Valor do ponteiro: %i\n", *ponteiro);

  printf("Endereço para onde aponta o ponteiro: %i", ponteiro;

1. Introdução
1.1. Ponteiros

Conceitos Ponteiros

1 bit = recebe valor ZERO ou UM

8 bits = 1 byte

1024 bytes = 1 kilobyte

1024 kilobytes = 1 megabytes

1024 megabytes = 1 gigabyte

1. Introdução
1.1. Ponteiros

Conceitos de Ponteiros

Int: para números inteiros entre -2147483648 e 2147483647: 4 bytes;

Char: para caracteres individuais do padrão ASCII: 1 byte;

Float: para reais entre (aproximadamente) 10-38 e 1038: 6 bytes, precisão de 8 dígitos;

Double: para reais entre (aproximadamente) 10-4932 e 104932: 8 bytes, precisão de 15 dígitos;

Bool: para indicar true (verdadeiro) ou false (falso): 2 bytes;

1. Introdução
1.1. Ponteiros

Mostrando Ponteiros no Código em C

```
//VALOR: é a variável que será apontada pelo ponteiro
int valor = 10;
//declaração de variável ponteiro
int *ptr;
 //atribuindo o endereço da variável valor ao ponteiro
ptr = &valor;
```

1. Introdução
1.2. Modularização

Exemplo:

Posso ter um programa com uma função para somar dois números e retornar a soma desses dois números.

1. Introdução
1.2. Modularização

Programar em C utilizando funções é muito importante para modularizar o código.

Desta forma o programa fica mais organizado, mais fácil de dar manutenção e reutilizar os códigos.

1. Introdução
1.2. Modularização

Revisando:

Um programa em C deve ter obrigatoriamente uma e somente uma função Principal.

Sempre a execução do programa começa pela função MAIN.

1. Introdução
1.2. Modularização

Revisando:

Uma peculiaridade da linguagem C é que você não pode definir uma função dentro de outra função.

Obs.: Na linguagem Python é permitido mas em C não.

1. Introdução
1.2. Modularização

Revisando:

Em C, assim como em outras linguagem, nós devemos reservar uma área na memória para armazenar cada dado.

Como é feito isso em C?

- Através da declaração de variáveis

  Exemplo:

  int resultado = 30;

Uma boa prática é inicializar a variável logo após sua declaração.

Isto evita o armazenamento de "lixo" na variável.

1. Introdução
1.2. Modularização

Revisando:

Variável local:

- Declarada dentro de uma função;

Variável Global

- Declarada fora de uma função;

- Como boa prática é declarada no arquivo do programa principal;

Em C, todas as variáveis devem ser explicitamente declaradas:

- Exemplo: Tipo e nome

  int soma; float salario; char nome;

1. Introdução
1.2. Modularização

Revisando:

USO DE BIBLIOTECAS

#include <stdio.h>

- Esta é uma biblioteca da linguagem C que permite a captura de dados pelo teclado e a saída de dados para a tela.

- Existem várias outras bibliotecas;

1. Introdução
1.2. Modularização

Revisando:

Comentários no código.

Comentários são ignorados pelo compilador.

É uma boa prática comentar seu código para futuras consultas. Se você está trabalhando em equipe, a comunicação fica facilitada.

1. Introdução
1.2. Modularização

Comandos da linguagem C

No avançar da disciplina iremos vendo os comandos e operadores utilizados e serão explicados gradativamente.

Operador sizeof:

Este operador permite saber o número de bytes ocupado por um determinado tipo de variável. É muito usado na alocação dinâmica de memória.

- Exemplo: sizeof(int);

1. Introdução
1.2. Modularização

Comandos da linguagem C

Estruturas de decisão:

IF () { … } ELSE

While (…) { … }

For (…) { … }

1. Introdução
1.2. Modularização

Funções em C

Utilizamos funções com o intuito de dividir um problema grande.

Desta forma fica mais fácil resolver e visualizar a solução.

Criamos as funções para codificar tarefas específicas.

Então, se você tem um problemas específicos, você cria funções para resolvê-los.

1. Introdução
1.2. Modularização

Módulos

- O uso de vários módulos pode não se justificar para programas pequenos mas é essencial para os de médio e grande porte.

- Facilita a divisão de uma tarefa maior;

- Quebra a tarefa maior em outra menores ficando mais fácil de implementar e testar;

- Pode ser usado para compor vários programas.

1. Introdução
1.2. Modularização

Módulos

- Todo módulo em C costuma ter um arquivo associado que contém apenas os protótipos das funções oferecidas pelo módulo;

- Estes arquivos de protótipo são a interface do módulo e em geral possui o mesmo nome do módulo associado, a diferença é que a extensão dele é .h;

1. Introdução
1.2. Modularização

Configurando a variável de ambiente no Windows

- Para ser possível compilar um programa em C, pode ser necessário baixar e instalar o programa MinGW para o Windows;

- 1º Faça o download deste programa e descompacte os arquivos na raiz do C:\

- Copie o endereço C:\MinGW\bin e inclua nas variáveis de ambiente do Windows na variável Path;

1. Introdução
1.2. Modularização

Módulos

- Comando para compilar um arquivo em C:

- gcc -c <nomedoarquivo>.c

  Este comando apenas compila o arquivo.

- Comando para gerar um executável em C:

- Após compilar será gerado um arquivo com a extensão .o
- gcc -o <nomedoarquivo>.exe <nomedodarquivo>.o

- Comando para compilar e gerar um executável em C:

- gcc <arquivo01>.c <arquivo02>.c **-o** <NomeExecutavel>.exe

- Executando o arquivo executável gerado:

- Navegue até o diretório onde o arquivo foi gerado;
- Digite: .\<NomeArquivo>.exe

Obrigado!