Lista de Exercícios - Linguagem de Programação II (C)

13 de abril de 2025

Instruções Gerais

- Implemente cada exercício em um arquivo `.c` separado.
- Compile e teste seus programas cuidadosamente. Use `gcc -Wall -Wextra -pedantic seu_arquivo.c -o seu_executavel` para compilação robusta.
- Preste atenção aos tipos de dados, escopo de variáveis e gerenciamento de memória (via ponteiros).
- Comente seu código quando necessário para explicar a lógica.
- A numeração das aulas é apenas uma referência aos tópicos abordados.

---

1. **Revisão de Estruturas e Matrizes (Aulas 4-6)**

   Tópico: Estruturas (`struct`), Matrizes.

   Descrição: Crie uma estrutura `Aluno` que armazene o nome (vetor de `char`), a matrícula (`int`) e 3 notas (vetor de `float`). Em seguida, crie um programa que:

   (a) Declare um vetor (array) de 5 `Alunos`.

   (b) Preencha os dados de cada aluno (pode ser direto no código ou lendo do teclado via `scanf`).

   (c) Calcule e imprima a média das notas de *cada* aluno.

   (d) Encontre e imprima o nome e a matrícula do aluno com a maior média geral.

2. **Funções Básicas e Escopo (Aulas 7-9)**

   Tópico: Funções (sem/com parâmetros, sem/com retorno), Escopo de Variáveis.

   Descrição: Implemente as seguintes funções:

   (a) `void imprimeTabuleiro(char tabuleiro[3][3])`: Recebe um tabuleiro de Jogo da Velha (matriz `char 3x3`) e o imprime na tela de forma organizada (ex: com separadores `|`).

(b) `int verificaGanhador(char tabuleiro[3][3])`: Recebe o tabuleiro e verifica se houve um ganhador (`'X'` ou `'O'`). Retorna `1` se `'X'` ganhou, `2` se `'O'` ganhou, e `0` se ninguém ganhou ainda (ou deu empate). Verifique linhas, colunas e as duas diagonais principais.

(c) Na função `main`, declare um tabuleiro `3x3`, preencha-o com um cenário de jogo (ex: vitória de `'X'`), chame `imprimeTabuleiro` e depois chame `verificaGanhador`, imprimindo uma mensagem indicando o resultado (Ex: "Jogador X venceu!", "Jogador O venceu!", "Ninguém venceu ainda.").

(d) *Questão teórica (responder em comentário no código):* Se você declarar uma variável `int contador` dentro da função `verificaGanhador`, ela pode ser acessada diretamente na função `main`? Explique o conceito de escopo local.

3. **Ponteiros - Introdução (Aulas 10-12)**

Tópico: Ponteiros (declaração, `&` - operador de endereço, `*` - operador de derreferência).

Descrição: Crie um programa que:

(a) Declare três variáveis inteiras: `int a = 10;`, `int b = 20;`, `int c = 30;`.

(b) Declare três ponteiros para inteiros: `int *ptrA;`, `int *ptrB;`, `int *ptrC;`.

(c) Faça com que cada ponteiro aponte para a variável correspondente (ex: `ptrA = &a;`).

(d) Imprima o *endereço* e o *valor* de cada variável usando:

- A própria variável (ex: `printf("Valor de a: %d, Endereço de a: %p n", a, &a);`).
- O ponteiro correspondente (ex: `printf("Valor apontado por ptrA: %d, Endereço armazenado em ptrA: %p n", *ptrA, ptrA);`).

(e) Modifique o valor de `b` para `50` *usando apenas o ponteiro* ***ptrB*** (ex: `*ptrB = 50;`).

(f) Imprima o novo valor de `b` (usando a variável `b`) para confirmar a alteração.

4. **Aritmética de Ponteiros e Vetores (Aulas 10-12, 16-18)**

Tópico: Ponteiros, Aritmética de Ponteiros, Vetores.

Descrição: Crie um programa que:

(a) Declare um vetor de inteiros `int numeros[5] = {10, 20, 30, 40, 50};`.

(b) Declare um ponteiro para inteiro `int *ptrNumeros` e faça-o apontar para o *início* do vetor `numeros`.

(c) Usando um loop (`for` ou `while`) e *apenas aritmética de ponteiros* (incremento do ponteiro `ptrNumeros++` ou acesso via `*(ptrNumeros + i)`) e o operador de derreferência (`*`), imprima todos os elementos do vetor. **Não use** a sintaxe de colchetes (`numeros[i]`) dentro do loop para acessar os elementos.

(d) Imprima também o endereço de memória de cada elemento usando o ponteiro.

5. **Funções com Parâmetros por Referência (Aulas 13-15)**

   Tópico: Funções, Passagem por Referência (usando ponteiros).

   Descrição: Implemente uma função chamada `calculaOperacoes` que:

   (a) Recebe dois números inteiros `a` e `b` como parâmetros (por valor).

   (b) Recebe dois *ponteiros* para inteiros, `int *ptrSoma` e `int *ptrProduto`, como parâmetros.

   (c) Dentro da função, calcula a soma (`a + b`) e o produto (`a * b`).

   (d) Armazena o resultado da soma no local de memória apontado por `ptrSoma` (usando `*ptrSoma = ...;`).

   (e) Armazena o resultado do produto no local de memória apontado por `ptrProduto` (usando `*ptrProduto = ...;`).

   (f) A função *não* deve ter retorno (`void`).

   (g) Na função `main`, declare duas variáveis inteiras `resultadoSoma` e `resultadoProduto`. Declare outras duas para os operandos (ex: `num1 = 7`, `num2 = 4`). Chame a função `calculaOperacoes` passando `num1`, `num2`, e os *endereços* de `resultadoSoma` e `resultadoProduto` (usando `&resultadoSoma` e `&resultadoProduto`).

   (h) Após a chamada da função, imprima os valores de `resultadoSoma` e `resultadoProduto` na `main` para verificar se foram alterados corretamente pela função.

6. **Integração - Ponteiros, Vetores e Funções (Aulas 16-18)**

   Tópico: Ponteiros, Vetores, Funções.

   Descrição: Escreva uma função `inverteVetor` que:

   (a) Recebe um ponteiro para o início de um vetor de inteiros (`int *vetor`) e o tamanho do vetor (`int tamanho`) como parâmetros.

   (b) Inverte a ordem dos elementos *dentro* do próprio vetor original (modifica o vetor passado como argumento). Use ponteiros e/ou aritmética de ponteiros para acessar e trocar os elementos. Dica: você pode usar dois ponteiros (ou índices), um começando do início e outro do fim, movendo-se em direção ao centro e trocando os elementos apontados/indexados a cada passo. Use uma variável auxiliar para a troca.

   (c) A função não retorna nada (`void`).

   (d) Na função `main`, declare um vetor (ex: `int v[6] = {10, 20, 30, 40, 50, 60};`), imprima o vetor original (pode criar uma função auxiliar `imprimeVetor`), chame a função `inverteVetor`, e depois imprima o vetor novamente para mostrar que ele foi invertido.

7. **Ponteiro para Ponteiro (Stack) - Conceito (Aulas 19-21)**

   Tópico: Ponteiro para Ponteiro (`**`), Indireção Dupla.

   Descrição: Crie um programa que:

   (a) Declare uma variável inteira `int valor = 100;`.

(b) Declare um ponteiro para inteiro `int *ptr1;` e faça-o apontar para `valor` (`ptr1 = &valor;`).

(c) Declare um ponteiro para ponteiro de inteiro `int **ptr2;` e faça-o apontar para `ptr1` (`ptr2 = &ptr1;`).

(d) Imprima na tela, de forma clara e identificada:

- O valor de `valor` usando a variável diretamente.
- O valor de `valor` usando `*ptr1`.
- O valor de `valor` usando `**ptr2`.
- O endereço de `valor` usando `&valor`.
- O endereço de `valor` (que está armazenado em `ptr1`) usando `ptr1`.
- O endereço de `valor` (acessado via `ptr2`) usando `*ptr2`.
- O endereço de `ptr1` usando `&ptr1`.
- O endereço de `ptr1` (que está armazenado em `ptr2`) usando `ptr2`.
- O endereço de `ptr2` usando `&ptr2`.

(e) Modifique o valor de `valor` para `200` usando *apenas o ponteiro para ponteiro `ptr2`* (ex: `**ptr2 = 200;`).

(f) Imprima o novo valor de `valor` (usando a variável `valor`) para confirmar a modificação.

8. **Treinamento com Ponteiro para Ponteiro e Funções (Aulas 22-24)**

Tópico: Ponteiro para Ponteiro (`**`), Funções, Passagem por Referência de Ponteiros.

Descrição: O objetivo é criar uma função que possa alterar *para onde* um ponteiro (declarado na `main`) está apontando. Imagine que você tem duas variáveis inteiras, `x = 10` e `y = 20`. Você também tem um ponteiro `p` que inicialmente aponta para `x`. Escreva uma função `trocaAlvoPonteiro` que:

(a) Recebe um *ponteiro para ponteiro* de inteiro (`int **pp`). Este `pp` conterá o endereço do ponteiro `p` da `main`.

(b) Recebe um *ponteiro* para inteiro (`int *novoAlvo`). Este conterá o endereço da variável para a qual queremos que `p` passe a apontar (neste caso, o endereço de `y`).

(c) Dentro da função, faça a mágica: modifique o conteúdo do endereço `pp` para que ele (que é o ponteiro `p` original) passe a conter o endereço armazenado em `novoAlvo`. (Use `*pp = novoAlvo;`).

(d) A função não retorna nada (`void`).

(e) Na função `main`:

- Declare `int x = 10;`, `int y = 20;`.
- Declare `int *p = &x;`.
- Imprima o valor apontado por `p` (`printf("Antes: p aponta para valor %d\n", *p);`). Deve ser 10.

- Chame a função `trocaAlvoPonteiro`, passando o *endereço* do ponteiro `p` (`&p`) como primeiro argumento e o endereço da variável `y` (`&y`) como segundo argumento.
- Após a chamada da função, imprima novamente o valor apontado por `p` (`printf("Depois: p aponta para valor %d n", *p);`). Agora deve ser 20, mostrando que `p` foi redirecionado para apontar para `y`.

Objetivo: Entender como usar `**` para permitir que uma função modifique um ponteiro declarado em outra função (passagem de ponteiro por referência).