Universidade Federal da Paraíba
Centro de Informática
Disciplina: Programação Estruturada
Professor: Gilberto Farias / Luciano Costa

Lista de Exercícios - Funções e Ponteiros

Observações

- A lista é composta por questões teóricas e questões práticas.
- As questões teóricas devem ser respondidas **de forma manuscrita**, de maneira organizada, depois digitalizadas, combinadas em um único arquivo PDF e enviadas pelo SIGAA.
- As questões práticas deverão ser respondidas com as implementações dos códigos.
- As implementações deverão estar sob controle de versão (`git`), hospedadas em um servidor público (`GitHub`) ou privado (`BitBucket`).
- Cada exercício deve ser implementado em um arquivo separado com o número do problema. Por exemplo: "p1.c", "p12.c". Coloque todos os arquivos na mesma pasta chamada `lista2`.
- No campo de observação, na hora do envio da lista, coloque o link para os códigos no repositório.

Universidade Federal da Paraíba
Centro de Informática
Disciplina: Programação Estruturada
Professor: Gilberto Farias / Luciano Costa

Questões Teóricas

1. Explique o que é um ponteiro em C, descrevendo o conceito de endereço de memória e por que ponteiros são importantes na linguagem.
2. Defina o que é um ponteiro nulo. Explique seu propósito e por que tentar acessar o conteúdo de um ponteiro nulo leva a erros de execução.
3. Explique a diferença entre o uso do operador `*` na declaração de um ponteiro e o uso do operador `*` para desreferenciar um ponteiro. Inclua exemplos de uso correto e incorreto.
4. Explique as vantagens da modularização na programação. Discuta como a modularização influencia a clareza, manutenção e reutilização de código.
5. Descreva os elementos que compõem uma função em C, incluindo tipo de retorno, nome, parâmetros formais, corpo da função e o papel da instrução `return`.
6. Diferencie parâmetros formais e parâmetros reais. Explique como ocorre o casamento entre eles durante a chamada de função e apresente um exemplo de chamada incorreta, justificando o erro.
7. Explique como funciona a passagem por valor em C e por que modificações feitas dentro da função não afetam as variáveis originais no escopo da função chamadora.
8. Explique como a passagem por referência pode ser simulada em C usando ponteiros. Descreva o que ocorre na memória durante a execução de uma função que recebe ponteiros como parâmetros.
9. Defina os conceitos de caso base e caso recursivo em uma função recursiva. Justifique por que ambos são necessários para o correto funcionamento da recursão.
10. Descreva o que acontece quando uma função recursiva não possui um caso base adequado. Inclua em sua resposta uma explicação sobre o estouro de pilha (*stack overflow*).
11. Explique os três modos de parâmetros: entrada, saída e entrada/saída. Dê um exemplo para cada modo, indicando quando seu uso é apropriado.
12. Explique como o uso de funções promove reutilização de código e reduz a ocorrência de erros. Cite uma função simples que poderia ser reutilizada em diversos programas.
13. Explique como funções promovem encapsulamento, ocultando detalhes internos de implementação e deixando o código principal mais organizado.

Universidade Federal da Paraíba
Centro de Informática
Disciplina: Programação Estruturada
Professor: Gilberto Farias / Luciano Costa

Questões Práticas

1. Implemente uma função recursiva `somaDigitos(int n)` que retorne a soma dos dígitos de um número inteiro. Exemplo: entrada `483` deve produzir saída `15`.

2. Implemente a função `somaMultiplos(int inicio, int fim, int k)` que retorna a soma de todos os múltiplos de `k` no intervalo fechado $[inicio, fim]$.

3. Um número é perfeito quando a soma de todos os seus divisores próprios é igual a ele. Implemente: `int somaDiv(int n, int d)` para somar recursivamente os divisores de `n` menores que ele; e `int ehPerfeito(int n)` para indicar se o número é perfeito.

4. Escreva uma função `minMax(int a, int b, int c, int *min, int *max)` que receba três valores inteiros e escreva, nos ponteiros `min` e `max`, respectivamente, o menor e o maior dos três números.

5. Implemente a função recursiva `passos(int n)` que retorna o número de operações necessárias para reduzir `n` a 1 segundo as regras: Se `n` é par, divida por 2. Se `n` é ímpar, subtraia 1. Conte quantos passos foram realizados até atingir o valor 1.

6. Dois números são amigos se a soma dos divisores próprios de um é igual ao outro, e vice-versa. Implemente: `int somaDivisores(int n, int d)` para somar divisores de modo recursivo; `int amigos(int a, int b)` para determinar se são números amigos.

7. Um número é palíndromo quando permanece igual ao ser lido de trás para frente. Implemente duas funções: `int inverte(int n, int acumulador)` que inverte o número recursivamente; `int ehPalindromo(int n)` que retorna 1 se o número for palíndromo e 0 caso contrário.

8. Leia seis valores numéricos. Crie uma função que conta quantos são positivos e calcule a média entre eles.

9. Dados dois números inteiros X e Y, crie uma função que calcule a soma dos números ímpares estritamente entre eles. Se $X > Y$, troque-os.

10. Implemente uma função recursiva que imprima a decomposição de um número inteiro N em fatores primos: $N = p_1 \cdot p_2 \cdot \cdots \cdot p_k$. Use a função: `void fatoresPrimos(int n, int divisor);` A função deve imprimir cada fator assim que ele for encontrado, sem armazenar resultados em vetores.

11. Implemente uma função: `void reduz(int *a, int *b);` que aplica repetidamente as regras:

$$\begin{cases} a = a - b, & \text{se } a > b, \\ b = b - a, & \text{se } b > a. \end{cases}$$

    O processo termina quando $a = b$. O programa deve imprimir quantas operações foram realizadas até a estabilização.