Lista de Exercícios N2 - 2025

Estrutura de Dados II

Este trabalho deve ser feito em equipe de até **três integrantes**. O trabalho é divido em duas seções: teórica e prática. Ambas devem ser adicionadas ao repositório da equipe. Todos os integrantes devem ter contribuições (*commits*) no repositório para a atribuição da nota.

Aproveite a lista para pesquisar e assimilar o conteúdo, além de aprimorar suas habilidades de lógica e programação. Recomendo pesquisar na internet, mas evitar, a princípio, utilizar IA para resolver as questões.

Todos os integrantes devem submeter o link do repositório da equipe no portal digital.

Parte 1 – Teoria

1. O que significa alocação estática de memória para um conjunto de elementos?
2. Qual a diferença entre alocação estática e alocação dinâmica?
3. O que é um ponteiro?
4. O que é uma estrutura de dados homogêneos?
5. O que é uma estrutura de dados heterogêneos?
6. Qual a vantagem das listas sobre os vetores em termos de consumo de memória? Exemplifique.
7. O que é uma lista simplesmente encadeada? Apresente um diagrama para ilustrar essa estrutura de dados.
8. O que é uma lista duplamente encadeada? Apresente um diagrama para ilustrar essa estrutura de dados.
9. O que é uma lista duplamente encadeada? Apresente um diagrama para ilustrar essa estrutura de dados.
10. Explique o funcionamento do algoritmo de busca binária e sequencial.
11. Explique o funcionamento dos seguintes algoritmos de ordenação: *Insertion sort, Selection sort, Merge sort, Count sort, Quicksort*.

Parte 2 – Prática

Questão 01

Em uma agenda telefônica os contatos são cadastrados com os seguintes dados:

- Nome – nome do contato (máximo 40 caracteres);
- Telefone – cadeia de caracteres com o número do telefone do contato (máximo 15 caracteres);
- Celular – cadeia de caracteres com o número do celular do contato (máximo 15 caracteres);
- Email – cadeia de caracteres com o email do contato (máximo 40 caracteres);
- DataAniversario – data de aniversário do contato (contendo dia e mês);

Essas informações podem ser representadas em dois tipos estruturados: Data e Contato.

Data
Dia
Mês

Contato
Nome
Telefone
Celular
Email
DataAniversario

Utilizando listas encadeadas simples, escreva um programa que permita o cadastro, edição, remoção, busca e impressão de contatos desta agenda telefônica. Os elementos da lista encadeada para armazenar contatos são representados pela seguinte estrutura:

```
1. struct elemento {
2. Contato info;
3. struct elemento* prox;
4.  };
5. typedef struct elemento Elemento;
```

O seu programa deve implementar as seguintes funções:

cria_agenda – cria uma lista encadeada retornando um ponteiro para NULL;

insere_contato – insere um novo contato na lista encadeada no final;

lista_contatos – exibe na tela todos os dados dos contatos existentes na agenda;

busca_contato – busca um contato na agenda com base em um determinado nome informado pelo usuário. A função retorna o endereço de memória do elemento encontrado ou NULL caso o contato não seja encontrado;

remove_contato – deleta um determinado contato existente na agenda. A função deve permitir ao usuário buscar por um contato na agenda (utilizando a função busca_contato previamente criada) e em seguida remover da lista o contato. Se o contato buscado não for encontrado, o programa deve exibir uma mensagem informando o usuário sobre esse fato;

atualiza_contato – modifica os dados de um contato já existente na agenda. A função deve permitir ao usuário buscar por um contato na agenda (utilizando a função busca_contatopreviamente criada) e em seguida alterar os dados do contato encontrado com base nas novas informações fornecidas pelo usuário. Se o contato buscado não for encontrado, o programa deve exibir uma mensagem informando o usuário sobre esse fato;

Em seguida, você deve implementar a função principal do programa que permita ao usuário realizar todas as operações da agenda: cadastro, impressão, edição, remoção, busca e remover duplicados.

O programa deve exibir um menu para o usuário escolher as operações desejadas.

1. Inserir Contato
2. Listar Contatos
3. Buscar Contato

4. Editar Contato
5. Remover Contato
6. Sair

O programa deve permitir que o usuário realize operações na agenda de contatos até o momento que ele desejar sair do programa (escolhendo a opção 6).

Questão 02

Construa um programa em C com as seguintes funções:

- Uma função que receba um valor n e crie dinamicamente um vetor de n elementos e retorne um ponteiro.
- Uma função que receba um ponteiro para um vetor e um valor n e imprima os n elementos desse vetor.
- Construa também uma função que receba um ponteiro para um vetor e libere esta área de memória.

Ao final, crie uma função principal que leia um valor n e chame a função criada acima. Depois, a função principal deve ler os n elementos desse vetor. Então, a função principal deve chamar a função de impressão dos n elementos do vetor criado e, finalmente, liberar a memória alocada através da função criada para liberação.

Questão 03

Você foi contratado como estagiário em um laboratório de automação industrial. Um dos sistemas legados da empresa controla um braço robótico que mistura componentes químicos. Para evitar ambiguidades com parênteses e precedência de operadores, as fórmulas de mistura são enviadas para a máquina em **notação posfixa**, também conhecida como Notação Polonesa Reversa (RPN).

Por exemplo, para misturar 20ml de um reagente com o resultado da soma de 10ml e 5ml de outro, a fórmula infixa 20 + (10 + 5) seria representada em RPN como 10 5 + 20 +.

O software atual que interpreta essas fórmulas está apresentando falhas e precisa ser reescrito. Sua tarefa é desenvolver, em C, um programa robusto que receba uma expressão matemática em RPN como uma string, a avalie e retorne o resultado final.

O Desafio

Implementar um programa em C que avalia expressões matemáticas na notação posfixa. O programa deve receber a expressão como um único argumento de linha de comando.

Requisitos:

1. **A Estrutura da Pilha:**
    - Você deve implementar uma pilha dinâmica (usando lista ligada) para armazenar os operandos.
    - A pilha deve ser capaz de armazenar números de ponto flutuante (double).

- Defina as structs necessárias para um nó (Node) e para o controle da pilha (Stack).

2. **Funções da Pilha:**
   - Implemente as operações fundamentais da pilha:
     - Stack* createStack(): Aloca e inicializa uma nova pilha vazia.
     - void push(Stack* stack, double value): Adiciona um valor ao topo da pilha.
     - double pop(Stack* stack): Remove e retorna o valor do topo da pilha. Se a pilha estiver vazia, o programa deve indicar um erro e terminar.
     - int isEmpty(Stack* stack): Retorna 1 se a pilha estiver vazia, 0 caso contrário.
     - void freeStack(Stack* stack): Libera toda a memória alocada para a pilha.
3. **O Avaliador:**
   - Crie a função principal da lógica, double evaluateRPN(char* expression).
   - Esta função deve percorrer a string da expressão, token por token (números e operadores são separados por espaços).
   - **Se o token for um número:** Converta-o para double e o empilhe (push).
   - **Se o token for um operador (+, -, *, /):**
     - Desempilhe (pop) os dois últimos operandos. **Atenção:** A ordem é importante para subtração e divisão! O primeiro valor desempilhado é o segundo operando.
     - Realize a operação.
     - Empilhe (push) o resultado de volta na pilha.
   - Ao final da expressão, o único valor restante na pilha será o resultado final.
   - **Tratamento de Erros:** O programa deve lidar com expressões malformadas (ex: operadores demais, operandos de menos) e com a divisão por zero. Em caso de erro, exiba uma mensagem clara.
4. **Programa Principal (main):**
   - A função main deve verificar se um argumento de linha de comando foi fornecido.
   - Ela passará esse argumento para a função evaluateRPN e imprimirá o resultado formatado.

Exemplos de Uso e Saída Esperada

1. **Expressão simples:**

- **Comando:** ./avaliador "10 5 +"
- **Saída Esperada:** Resultado: 15.00

2. **Expressão complexa:**
   - **Comando:** ./avaliador "5 1 2 + 4 * + 3 -" (Equivalente a 5 + ((1 + 2) * 4) - 3)
   - **Saída Esperada:** Resultado: 14.00
3. **Expressão com divisão:**
   - **Comando:** ./avaliador "10 4 2 / +" (Equivalente a 10 + (4 / 2))
   - **Saída Esperada:** Resultado: 12.00
4. **Expressão malformada (operandos insuficientes):**
   - **Comando:** ./avaliador "10 5 + *"
   - **Saída Esperada:** Erro: Expressao malformada (operandos insuficientes para o operador '*').
5. **Erro de divisão por zero:**
   - **Comando:** ./avaliador "10 0 /"
   - **Saída Esperada:** Erro: Tentativa de divisao por zero.

Dicas e Observações

- Utilize a função strtok(str, delim) da biblioteca <string.h> para dividir a string da expressão em tokens usando o espaço como delimitador.
- Utilize atof(str) da biblioteca <stdlib.h> para converter uma string (como "12.5") para um double.
- Para verificar se um token é um operador, você pode simplesmente comparar o primeiro caractere: if (token[0] == '+').
- Lembre-se da ordem correta para subtração e divisão: val2 = pop(s); val1 = pop(s); result = val1 - val2;.
- Não se esqueça de gerenciar a memória! Libere todos os nós e a própria estrutura da pilha ao final do programa com freeStack()