Listas Lineares - Implementação

Estrutura de Dados

Professor: Henrique Viana Oliveira, henriq.viana@uece.br

Exercício: 1.

Implemente as classes **ArrayStack**, **ArrayQueue** e **ArrayDeque**.

Exercício: 2.

Execute a seguinte série de operações de pilha, assumindo uma pilha inicialmente vazia: push(5), push(3), pop(), push(2), push(8), pop(), pop(), push(9), push(1), pop(), push(7), push(6), pop(), pop(), push(4), pop(), pop().

Exercício: 3.

Implemente uma função com assinatura **transfer(S, T)** que transfira todos os elementos da pilha S para a pilha T, de modo que o elemento que começa no topo de S seja o primeiro a ser inserido em T, e o elemento na parte inferior de S termine no topo de T.

Exercício: 4.

Forneça um método recursivo para remover todos os elementos de uma pilha.

Exercício: 5.

Implemente uma função que inverte uma lista de elementos colocando-os em uma pilha em uma ordem e escrevendo-os de volta na lista na ordem inversa.

Exercício: 6.

Execute a seguinte série de operações de fila, assumindo uma fila inicialmente vazia: enqueue(5), enqueue(3), dequeue(), enqueue(2), enqueue(8), dequeue(), dequeue(), enqueue(9), enqueue(1), dequeue(), enqueue(7), enqueue(6), dequeue(), dequeue(), enqueue(4), dequeue(), dequeue().

Exercício: 7.

Execute a seguinte série de operações de deque, assumindo uma deque inicialmente vazia: add first(4), add last(8), add last(9), add first(5), back(), delete first( ), delete last( ), add last(7), first( ), last( ), add last(6), delete first( ), delete first( ).

Exercício: 8.

Escreva um programa para verificar se em uma string contendo uma expressão aritmética, os parênteses de abertura e fechamento estão bem formados ou não.

Exercício: 9.

Escreva um programa para converter uma expressão aritmética na forma prefixada para formas infixas e pós-fixadas equivalentes.

Exercício: 10.

Implemente uma calculadora aritmética de inteiros usando pilhas.

Exercício: 11.

Crie um sistema usando uma pilha e uma fila para testar se uma determinada string é um palíndromo (ou seja, se os caracteres são lidos da mesma forma, tanto para a frente quanto para trás).

Exercício: 12.

Implemente as classes **LinkedStack**, **LinkedQueue**, **CircularQueue** e **LinkedDeque**.

Exercício: 13.

Forneça um algoritmo para encontrar o penúltimo nó em uma lista encadeada simples na qual o último nó é indicado por uma próxima referência de **None**.

Exercício: 14.

Descreva um bom algoritmo para concatenar duas listas encadeadas simples L e M, dadas apenas referências ao primeiro nó de cada lista, em uma única lista L que contém todos os nós de L seguidos por todos os nós de M.

Exercício: 15.

Descreva um algoritmo recursivo que conta o número de nós em uma lista encadeada simples.

Exercício: 16.

Implemente uma função que conta o número de nós em uma lista circularmente encadeada.

Exercício: 17.

Descreva um algoritmo recursivo rápido para reverter uma lista encadeada simples.

Exercício: 18.

Uma lista encadeada contém alguns números positivos e alguns números negativos. Usando essa lista encadeada, escreva um programa para criar duas novas listas encadeadas, uma contendo todos os números positivos e a outra contendo todos os números negativos.

Exercício: 19.

Escreva um programa para excluir elementos duplicados em uma lista duplamente encadeada.

Exercício: 20.

Modifique a classe **_DoublyLinkedBase** para incluir um método reverse que inverte a ordem da lista, mas sem criar ou destruir nenhum nó.