Sumário

Introdução

Esta lista de exercícios foi proposta pelo Prof. Dr. Igor Machado Coelho, como atividade continuada do tópico Estruturas Lineares na disciplina de Estrutura de Dados e Algoritmos, do Programa de Pós-graduação em Ciência da Computação (PGC) da Universidade Federal Fluminense (UFF), Campus Niterói.

O autor é servidor público no Instituto Federal de Rondônia (IFRO) - *Campus* Vilhena e atua como docente nas disciplinas da área de Engenharia Elétrica, também é aluno, em nível de doutorado, do PGC-UFF e participa do Projeto de Cooperação entre Instituições para Qualificação de Profissionais de Nível Superior (PCI), firmado entre o IFRO e a UFF.

Esta lista de exercícios serviu de oportunidade para que o autor tivesse uma introdução à orientação à objetos e estrutura de projeto utilizando boas práticas de programação. Foi um desafio sair do pensamento de programação estruturada, com o código em apenas um arquivo e passar a separar as funções em arquivos e subpastas.

A decisão de utilizar um tipo genérico, na resolução do 1º exercício, levou o autor a buscar entendimento de *makefiles*. Uma vez que

Exercício 1

A fim de responder essa questão implementei tanto o deque sequencial quanto o encadeado. Porém, me limitei a comentar apenas no deque encadeado porque foi o que trouxe maior aprendizado. Antes de implementá-lo, a alocação dinâmica era bem nebulosa para mim. Preferi utilizar o tipo genérico, de modo que a implementação pudesse ser reaproveitada em exercícios posteriores, a exemplo das alternativas dessa questão e do Exercício 7, que foi em ordem cronológica o 2º exercício a ser implementado.

Preferi, também, destinar apenas uma pasta para os arquivos Os arquivos de cabeçalho e foram A escolha pelo tipo genérico, associado à estrutura de diretórios

Foram implementados os arquivos de cabeçalho

```
#ifndef _DEQUE_ENCADEADO_HPP_
#define _DEQUE_ENCADEADO_HPP_

template<typename TipoGenerico>
class NoDeque
{
public:
    TipoGenerico dado;
    NoDeque<TipoGenerico> *noSeguinte;
    NoDeque<TipoGenerico> *noAnterior;
};

template <typename TipoGenerico>
class Deque
{

private:
    NoDeque<TipoGenerico> *inicio;
    NoDeque<TipoGenerico> *fim;
    int numeroElementos;

public:
    Deque();
    ~Deque();
    void insereInicio(TipoGenerico v);
    void insereFim(TipoGenerico v);
    int tamanho();
```

```
28      TipoGenerico buscaInicio();
29      TipoGenerico buscaFim();
30      TipoGenerico removeInicio();
31      TipoGenerico removeFim();
32  };
33
34  #include "dequeEncadeado.cpp"
35
36  #endif
```

Com relação à complexidade, exceto o método destrutor `~Deque()`, que tem crescimento linear, ou seja, complexidade $O(N)$, todos os demais métodos possuem tempo constante, isto é, complexidade $O(1)$.

e de implementação. No arquivo de

Exercício 2

Exercício 3

Exercício 4

Exercício 5

Exercício 6

Exercício 7

Para resolver este exercício, uma pilha armazenou os operadores e um vetor, a saída.

De modo que um laço percorre a entrada e armazena letras (variáveis) diretamente na saída e sinais (operações) na pilha até encontrar: 1. um parêntese fechado; ou 2. um operador de precedência inferior ao último encontrado.

Assim que uma das duas condições é satisfeita, duas coisas podem ocorrer, respectivamente: 1. o último operador é desempilhado, inserido no vetor de saída, e o um parêntese aberto também é desempilhado; 2. o operador de maior precedência é desempilhado, inserido na saída e o novo operador é empilhado.

Ao chegar ao final do laço, todos os operadores são desempilhados e inseridos na saída.

Com o objetivo de determinar as precedências, utilizou-se o código ASCII do caractere subtraído de 41 em módulo 6. Esta transformação faz com que o caractere '*' (decimal 42), torne-se 1; '+' (decimal 43), torne-se 2; '-' (decimal 45), torne-se 4; e, por fim, o caractere '/' (decimal 47), torne-se 0.