Lista de Exercícios - Curso de Estruturas de Dados I (em C/C++)
Prof. Igor Machado Coelho - Tópico: Estruturas Lineares

ALUNO: CARLOS EDUARDO SANTANA AZEVEDO

1. Considere um tipo chamado Deque, que inclui manipulação de dois extremos em uma estrutura linear (como se operasse como Pilha e Fila simultaneamente).

1.a) Satisfaça as seguintes operações de um DequeTAD para o tipo 'char', utilizando uma estrutura Sequencial OU uma estrutura encadeada:

- RESPOSTA PROPOSTA -> Foram satisfeitas as condições propostas no enunciado - DequeTAD, tais como as implementações das funções: void cria(), char ver_inicio(), void insere_inicio(char dado), char remove_inicio(), char ver_fim(), void insere_fim(char dado), char remove_fim(), conforme solicitado com os devidos testes efetuados através da implementação do algoritmo no VS Code (C++).

++++++++++++++++++++ IMPLEMENTAÇÃO DOS ALGORITMOS ++++++++++++++++++++

```
#include <iostream>

constexpr int MAXN = 5; // Capacidade máxima do deque (VETOR).

using namespace std;

class Deque1 // Implementação do DequeTAD com as sentenças propostas na Atividade.
{
public:

    // Declaração de variáveis.

    char elementos[MAXN]; // Vetor de Máximo de elementos.
    int N;                // Número de elementos.
    int inicio;           // Posição do início.
    int fim;              // Posição do fim.

   // Declaração das Funções.

    void cria() // inicialização zerando o número de elementos/inicio/fim do deque.
    {
        this->N = 0;
        this->inicio = 0;
        this->fim = 0;
    }
```

```
    char ver_inicio() // Função/Método para retorna o valor do elemento
no início.
    {
        return this->elementos[inicio]; // Retorna o valor do elemento
no início.
    }

    void insere_inicio(char dado) // Função/Método para inseri no
início  do deque o elemento "dado" tipo 'char'.
    {
        this->elementos[fim] = dado; // Insere o 'dado' no elemento
fim.
        this->fim = (fim + 1) % MAXN; /* O Elemento fim é incrementado
com o resto da divisão entre o valor de fim (atualizado no laço)
                                        + o valor Máximo de elementos
MAXN, dessa forma é garantida que a Função/Método ultrapasse
                                        o número de elementos do vetor
elementos, pois se torna uma estratégia circular no Vetor,
                                        caso o número de inserções for
maior que MAXN.*/
        this->N++; // Incrementa o número de elementos.
    }

    char remove_inicio() // Função/Método para Remover no início do
deque o elemento "dado" tipo 'char'.
    {
        char r = this->elementos[inicio]; // Armazena na variável 'r' o
valor do elemento no início.
        this->inicio = (inicio + 1) % MAXN; /*Idem da Explicação
anteriopr para o %MAXN (resto da divisão entre o valor de fim
                                            (atualizado no laço) + o
valor Máximo de elementos MAXN).*/
        this->N--; // Decrementa o número de elementos.
        return r; // Retorna o valor do elemento no início.
    }
    char ver_fim() // Função/Método para retornar o elemento no fim do
Deque.
    {
        return this->elementos[N + inicio - 1]; /* N é o número atual
de elementos do deque, inicio é a posição atual do início
```

```
                                                    do deque e com o
decremento de 1 é possível determinar a posição exata do fim do
Deque.*/
    }

    void insere_fim(char dado) // Função/Método para inseri no fim do
deque o elemento "dado" tipo 'char'.
    {
        this->elementos[fim] = dado; // Insere o 'dado' no elemento
fim.
        this->fim = (fim + 1) % MAXN;/*Idem da Explicação anteriopr
para o %MAXN (resto da divisão entre o valor de fim
                                    (atualizado no laço) + o valor
Máximo de elementos MAXN).*/
        this->N++;  // Incrementa o número de elementos.
    }

    char remove_fim() // Função/Método para remover no fim do deque o
elemento "dado" tipo 'char'.
    {
        char r = this->elementos[fim]; // Armazena na variável 'r' o
valor do elemento no fim.
        this->fim = (fim + 1) % MAXN;/*Idem da Explicação anteriopr
para o %MAXN (resto da divisão entre o valor de fim
                                    (atualizado no laço) + o valor
Máximo de elementos MAXN).*/
        this->N--; // Decrementa o número de elementos.
        return r; // Retorna o valor do elemento no fim.
    }
};

int main()
{

    Deque1 *p = new Deque1();
…
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

1.b*) Implemente uma estrutura PilhaDeque para tipo ‘char’, utilizando somente um Deque como armazenamento interno e mais espaço auxiliar constante:

- RESPOSTA PROPOSTA -> Sem necessidade de apresentação conforme orientação em aula.

1.c*) Implemente uma estrutura FilaDeque para tipo ‘char’, utilizando somente um Deque como armazenamento interno e mais espaço auxiliar constante:

- RESPOSTA PROPOSTA -> Sem necessidade de apresentação conforme orientação em aula.

2) Implemente uma estrutura que satisfaz o TAD Pilha para o tipo ‘char’ e somente utiliza duas Filas como armazenamento interno (mais espaço constante):

- RESPOSTA PROPOSTA -> Foram satisfeitas as condições propostas no enunciado de PilhaTAD, a implementação foi efetuada com a utilização das Fila 1(`std::queue<char> f1;`) e Fila 2 (`std::queue<char> f2;`) e criação de funções/métodos (`char topo() | char desempilha() | void empilha(char dado) |`) de forma que foram criados laços de repetição para a movimentação dos elementos entre as Filas e posterior armazenamento constituindo a TADPilha. Conforme solicitado, os devidos testes foram efetuados através da implementação do algoritmo no VS Code (C++).

++++++++++++++++++++ IMPLEMENTAÇÃO DOS ALGORITMOS ++++++++++++++++++++

```
#include <iostream>
#include <queue>

class Pilha2F // Implementação da PilhaTAD com as sentenças propostas na Atividade.
{
public:

    // Declaração de variáveis.

    std::queue<char> f1; // Fila 1 de elementos tipo char.
    std::queue<char> f2; // Fila 2 de elementos tipo char.

    // Declaração das Funções.

    char topo() // Função/Método para retorna o valor do elemento no início da Pilha.
    {
    return this->f1.front();
    }

    char desempilha() // Função/Método para remover o valor do elemento no fim da Pilha.
    {
```

```
    char r = this->f1.front(); //Aqui é armazanado o valor do elemento
removido numa variável auxiliar tipo 'char', para um posterior retorno.
    this->f1.pop(); //Remove o 1° elemento da Fila 1.
    return r; // Variável 'r' retornada com o 1° elemento da Pilha.
    }

    void empilha(char dado) // Função/Método para inserir (criar a
Pilha) o valor do elemento na Pilha.
    {
      while (!f1.empty()) // Laço de repetição (Enquanto a Fila 1 não
estiver vazia) - o elemento tipo 'char' dado é inserido na Fila 2.
      {
        this->f2.push(f1.front()); // Fila 2 recebe o valor do 1°
elemento da Fila 1.
        this->f1.pop(); // Remove o 1° elemento da Fila 1.
      }

    f1.push(dado); // Insere outro novo elemento tipo 'char' dado na
Fila 1.

    while (!f2.empty()) // Laço de repetição (Enquanto a Fila 2 não
estiver vazia) - o elemento tipo 'char' dado é inserido na Fila 01.
    {
        this->f1.push(f2.front());// Fila 1 recebe o valor do 1°
elemento da Fila 2.
        this->f2.pop(); // Remove o 1° elemento da Fila 2.
    }
    }
};

int main()
{

    Pilha2F *p = new Pilha2F();
…
```

++

3) Implemente uma estrutura que satisfaz o TAD Fila para o tipo ‘char’ e somente utiliza duas Pilhas como armazenamento interno (mais espaço constante):

- RESPOSTA PROPOSTA -> Foram satisfeitas as condições propostas no enunciado de FilaTAD, a implementação foi efetuada com a utilização das Pilha 1 (`std::stack<char> p1;`) e Pilha 2 (`std::stack<char> p2;`) e criação de funções/métodos (`void enfileira(char dado) | char front() | char`

`desenfileira()`) de forma que foram criados laços de repetição para a movimentação dos elementos entre as pilhas e com a criação de uma Variável ‘r’ para retorno de valores dos elementos na chamada da **função ‘char front()’** e posteriores armazenamentos constituindo a TADFila. Conforme solicitado, os devidos testes foram efetuados através da implementação do algoritmo no VS Code (C++).

++++++++++++++++++++ IMPLEMENTAÇÃO DOS ALGORITMOS ++++++++++++++++++++

```
#include <iostream>
#include <stack>

class Fila2P // Implementação da FilaTAD com as sentenças propostas na Atividade.
{
public:

    // Declaração de variáveis.

    std::stack<char> p1; // Pilha 1 de elementos tipo char.
    std::stack<char> p2; // Pilha 2 de elementos tipo char.

    int N = 0;

    // Declaração das Funções.

    void enfileira(char dado) // Função/Método para inserir (criar a Fila) o valor do elemento na Fila.
    {
        p1.push(dado); // Insere os valores dos elementos na Pilha 1.
        this->N++; // Incrementa o valor da variável N.
    }

    char front() // Função/Método para retorna o valor do elemento no início da Fila.
    {
        while (!p1.empty()) // Laço de repetição (Enquanto a Pilha 1 não estiver vazia) - Pilha 2 recebe os elementos da Pilha 1.
        {
            this->p2.push(p1.top()); // Pilha 2 recebe o valor do 1° elemento da Pilha 1.
            this->p1.pop(); // Remove o 1° elemento da Pilha 1.
        }

        char r = this->p2.top(); // Variável 'r' recebe o valor do 1° elemento da Pilha 2.
```

```
        while (!p1.empty()) // Laço de repetição (Enquanto a Pilha 1
não estiver vazia) - Pilha 1 recebe o 1° elemento da Pilha 2.
        {
            this->p1.push(p2.top()); // Pilha 1 recebe o valor do 1°
elemento da Pilha 2.
        }
        return r;// Variável 'r' retornada com o 1° elemento da Fila.
    }

    char desenfileira() // Função/Método para remover o valor do
elemento no início da Fila.
    {
        while (!p1.empty()) // Laço de repetição (Enquanto a Pilha 1
não estiver vazia) - Pilha 2 recebe o 1° elemento da Pilha 1.
        {
            p2.push(p1.top()); // Pilha 2 recebe o valor do 1° elemento
da Pilha 1.
            p1.pop(); // Remove o 1° elemento da Pilha 1.
        }

        char r = p2.top(); // Variável 'r' recebe o valor do 1°
elemento da Pilha 2.

        p2.pop(); // Remove o 1° elemento da Pilha 2.

        while (!p2.empty()) //
        {
            p1.push(p2.top()); // Pilha 1 recebe o valor do 1° elemento
da Pilha 2.
            p2.pop(); // Remove o 1° elemento da Pilha 2.
            return r;  // Variável 'r' retornada com o 1° elemento da
Pilha 2.
        }
        this->N--; // Decrementa o valor da variável N.
    }
};

int main()
{
    Fila2P *p = new Fila2P();
...
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

4) Escreva um algoritmo que dada uma pilha padrão P externa passada como parâmetro, inverte o conteúdo de P. Somente utilize as estruturas extras permitidas como armazenamento externo(mais espaço constante):

a) Uma Fila:

- RESPOSTA PROPOSTA -> Foram satisfeitas as condições propostas no enunciado de PilhaTAD, a implementação foi efetuada com a utilização da Fila 1(`std::queue<char> f1;`), da Pilha p1 (`std::stack<char> p1;`) e criação da função/métodos (`void inverte(std::stack<char> *p1)`), que faz uso dos ponteiros da Pilha p1. Dessa forma, foram criados laços de repetição para a movimentação dos elementos entre a Fila e a Pilha (índices do ponteiro) e posterior armazenamento constituindo a TADPilha. Conforme solicitado, os devidos testes foram efetuados através da implementação do algoritmo no VS Code (C++).

++++++++++++++++++++ IMPLEMENTAÇÃO DOS ALGORITMOS ++++++++++++++++++++

```
#include <iostream>
#include <stack>
#include <queue>

class FilaXPilha // Implementação da PilhaTAD com as sentenças
propostas na Atividade.
{
public:

// Declaração da função.

    void inverte(std::stack<char> *p1) // Função/Método para inverter a
Pilha 'p1', com a utilização de Ponteiro da Pilha 'p1'.
    {
        std::queue<char> f1; // Declaração da Fila 1.

        while (!p1->empty()) // Laço de repetição (Enquanto a Pilha 1
não estiver vazia) - Fila 1 recebe os elementos da Pilha 1.
        {
            f1.push(p1->top()); // Fila 1 recebe o valor do 1° elemento
da Pilha 1.
            p1->pop(); // Remove o 1° elemento da Pilha 1.
        }
        while (!f1.empty()) // Laço de repetição (Enquanto a Fila 1 não
estiver vazia) - Pilha 1 recebe o 1° elemento da Fila 1.
        {
            p1->push(f1.front()); // Pilha 1 recebe o valor do 1°
elemento da Fila 1.
            f1.pop(); // Remove o 1° elemento da Fila 1.
```

```
        }
    }
};

int main()
{
    std::stack<char> p1;

    FilaXPilha *p = new FilaXPilha();
...
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

b) Duas Pilhas:

- RESPOSTA PROPOSTA -> Foram satisfeitas as condições propostas no enunciado de PilhaTAD, a implementação foi efetuada com a utilização das Pilhas A (`std::stack<char> pA;`), Pilha B (`std::stack<char> pB;`), Pilha p (`std::stack<char> p;`) e criação da função/métodos (`void inverte(std::stack<char> *p)`), que faz uso dos ponteiros da Pilha p. Dessa forma, foram criados laços de repetição para a movimentação dos elementos entre a Fila e a Pilha (índices do ponteiro) e posterior armazenamento constituindo a TADPilha. Conforme solicitado, os devidos testes foram efetuados através da implementação do algoritmo no VS Code (C++).

++++++++++++++++++++ IMPLEMENTAÇÃO DOS ALGORITMOS ++++++++++++++++++++

```
#include <iostream>
#include <stack>

class Pilhasx2 // Implementação da PilhaTAD com as sentenças propostas na Atividade.
{
public:

    // Declaração da função.

    void inverte(std::stack<char> *p) // Função/Método para inverter a Pilha 'p', com a utilização de Ponteiro da Pilha 'p'.

    {
        std::stack<char> pA; // Declaração da Pilha A.
        std::stack<char> pB; // Declaração da Pilha B.

        while (!p->empty()) // Laço de repetição (Enquanto a Pilha p não estiver vazia) - Pilha A recebe os elementos da Pilha p.
        {
```

```
            pA.push(p->top()); // Pilha A recebe o valor do 1° elemento
da Pilha p.
            p->pop(); // Remove o 1° elemento da Pilha p.
        }
        while (!pA.empty()) // Laço de repetição (Enquanto a Pilha A
não estiver vazia) - Pilha p recebe o 1° elemento da Pilha A.
        {
            pB.push(pA.top()); // Pilha B recebe o valor do 1° elemento
da Pilha A.
            pA.pop(); // Remove o 1° elemento da Pilha A.
        }
        while (!pB.empty()) // Laço de repetição (Enquanto a Pilha B
não estiver vazia) - Pilha p recebe o 1° elemento da Pilha B.
        {
            p->push(pB.top()); // Pilha p recebe o valor do 1° elemento
da Pilha B.
            pB.pop(); // Remove o 1° elemento da Pilha B.
        }
    }
};

int main()
{

    std::stack<char> p;

    Pilhasx2 *px2 = new Pilhasx2();
...
```

++

c) Uma Pilha:

- RESPOSTA PROPOSTA -> Foram satisfeitas as condições propostas no enunciado de PilhaTAD, a implementação foi efetuada com a utilização das Pilhas p1 (`std::stack<char> p1;`), Pilha f1 (`std::stack<char> f1;`), e criação da função/métodos (`void inverte (std::stack<char>* f1)`), que faz uso dos ponteiros da Pilha f1. Dessa forma, foram criados laços de repetição para a movimentação dos elementos entre a Fila e a Pilha (índices do ponteiro) e posterior armazenamento constituindo a TADPilha. Conforme solicitado, os devidos testes foram efetuados através da implementação do algoritmo no VS Code (C++).

++++++++++++++++++++ IMPLEMENTAÇÃO DOS ALGORITMOS ++++++++++++++++++++

```
#include <iostream>
#include <stack>
```

```
class PilhaXPilha // Implementação da PilhaTAD com as sentenças
propostas na Atividade.
{
public:

    // Declaração da função.

    void inverte (std::stack<char>* f1) // Função/Método para inverter
a Pilha 'f1', com a utilização de Ponteiro da Pilha 'f1'.
    {
        std::stack<char> p1; // Declaração da Pilha p1.

        while (!f1->empty()) // Laço de repetição (Enquanto a Pilha f1
não estiver vazia) - Pilha p1 recebe os elementos da Pilha f1.
        {
            p1.push(f1->top()); // Pilha P1 recebe o valor do 1°
elemento da Pilha f1.
            f1->pop(); // Remove o 1° elemento da Pilha f1.
        }
        while (!p1.empty()) // Laço de repetição (Enquanto a Pilha p1
não estiver vazia) - Pilha f1 recebe o 1° elemento da Pilha P1.
        {
            f1->push(p1.top()); // Pilha f1 recebe o valor do 1°
elemento da Pilha p1.
            p1.pop(); // Remove o 1° elemento da Pilha p1.
        }
    }

};

int main()
{
std::stack<char> f1;

PilhaXPilha *p = new PilhaXPilha();
...
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

5) Escreva um algoritmo que, dada uma fila padrão F externa passada como parâmetro, inverte o conteúdo de F. Somente utilize as estruturas extras permitidas como armazenamento externo (mais espaço constante):

a) Uma Pilha:

- RESPOSTA PROPOSTA -> Foram satisfeitas as condições propostas no enunciado de FilaTAD, a implementação foi efetuada com a utilização das Pilhas p1 (`std::stack<char> p1;`), Fila 1(`std::queue<char> f1;`), e criação da função/métodos (`void inverte(std::queue<char> *f1)`), que faz uso dos ponteiros da Fila f1. Dessa forma, foram criados laços de repetição para a movimentação dos elementos entre a Pilha e a Fila (índices do ponteiro) e posterior armazenamento constituindo a TADFila. Conforme solicitado, os devidos testes foram efetuados através da implementação do algoritmo no VS Code (C++).

++++++++++++++++++++ IMPLEMENTAÇÃO DOS ALGORITMOS ++++++++++++++++++++

```
#include <iostream>
#include <stack>
#include <queue>


class FilaxPilha // Implementação da FilaTAD com as sentenças propostas na Atividade.
{
public:

    // Declaração da função.

    void inverte(std::queue<char> *f1) // Função/Método para inverter a Fila 'f1', com a utilização de Ponteiro da Fila 'f1'.
    {
        std::stack<char> p1; // Declaração da Pilha p1.

        while (!f1->empty()) // Laço de repetição (Enquanto a Fila f1 não estiver vazia) - Pilha p1 recebe os elementos da Fila f1.
        {
            p1.push(f1->front()); // Pilha P1 recebe o valor do 1° elemento da Fila f1.
            f1->pop(); // Remove o 1° elemento da Fila f1.
        }
        while (!p1.empty()) // Laço de repetição (Enquanto a Pilha p1 não estiver vazia) - Fila f1 recebe o 1° elemento da Pilha P1.
        {
            f1->push(p1.top()); // Fila f1 recebe o valor do 1° elemento da Pilha p1.
            p1.pop(); // Remove o 1° elemento da Pilha p1.
        }
    }
};
```

```
int main()
{
    std::queue<char> f1;

    FilaxPilha *p = new FilaxPilha();
...
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

b) Duas Filas:

- RESPOSTA PROPOSTA -> Foram satisfeitas as condições propostas no enunciado de FilaTAD, a implementação foi efetuada com a utilização das Fila A(`std::queue<char> fA;`), Fila B(`std::queue<char> fB;`), Fila f(`std::queue<char> f;`) e criação da função/métodos (`void inverte(std::queue<char> *f)`), que faz uso dos ponteiros da Fila f. Dessa forma, foram criados laços de repetição - Destaque para o "for" que armazena o último elemento da Fila f no 1º "While" - para a movimentação dos elementos entre as Filas (índices do ponteiro) e posterior armazenamento constituindo a TADFila. Conforme solicitado, os devidos testes foram efetuados através da implementação do algoritmo no VS Code (C++).

++++++++++++++++++++ IMPLEMENTAÇÃO DOS ALGORITMOS ++++++++++++++++++++

```
#include <iostream>
#include <queue>

class Filasx2 // Implementação da FilaTAD com as sentenças propostas na Atividade.
{
public:

    // Declaração da função.

    void inverte(std::queue<char> *f) // Função/Método para inverter a Fila 'f', com a utilização de Ponteiro da Fila 'f'.
    {
        std::queue<char> fA; // Declaração da Fila fA.
        std::queue<char> fB; // Declaração da Fila fB.
        int n = f->size(); // Declaração da variável n, que recebe o tamanho da Fila f.

        while (--n > 0) // Laço de repetição (Enquanto --n for maior 0) serão executados os laços e comandos abaixo.
        {
            for (int i = 0; i < n; i++) // Laço de repetição (Enquanto i for menor que n) - Fila fB recebe os elementos da Fila f.
            {
```

```
                fB.push(f->front()); // Fila fB recebe o valor do 1°
elemento da Fila f.
                f->pop(); // Remove o 1° elemento da Fila f.
            }

            fA.push(f->front()); // Fila fA recebe o valor do 1°
elemento da Fila f.
            f->pop(); // Remove o 1° elemento da Fila f.

            while (!fB.empty()) // Laço de repetição (Enquanto a Fila
fB não estiver vazia) - Fila f recebe o 1° elemento da Fila fB.
            {
                f->push(fB.front()); // Fila f recebe o valor do 1°
elemento da Fila fB.
                fB.pop(); // Remove o 1° elemento da Fila fB.
            }
        }

        fA.push(f->front()); // Fila fA recebe o valor do 1° elemento
da Fila f.
        f->pop(); // Remove o 1° elemento da Fila f.

        while (!fA.empty()) // Laço de repetição (Enquanto a Fila fA
não estiver vazia) - Fila f recebe o 1° elemento da Fila fA.
        {
            f->push(fA.front()); // Fila f recebe o valor do 1°
elemento da Fila fA.
            fA.pop(); // Remove o 1° elemento da Fila fA.
        }
    }
};

int main()
{

    std::queue<char> f;

    Filasx2 *px2 = new Filasx2();
...
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

6) Criar uma implementação do TAD Pilha para o tipo ‘int’, chamada PilhaMin, que oferece os métodos do TAD e também o método obterMinimo(), que retorna o menor elemento da pilha. O método obterMinimo() deve operar em tempo constante.

- SEM RESPOSTA PROPOSTA;

7) Escreva um algoritmo que converte uma expressão aritmética parentizada usando as 4 operações para a expressão correspondente em notação polonesa reversa.

Exemplo:

Entrada: “((A+B)*(C-(F/D)))”
Saída: “AB+CFD/-*”

- SEM RESPOSTA PROPOSTA;