INSTITUTO FED. DE EDUCAÇÃO, CIÊNC. E TEC. DE PERNAMBUCO
CURSO: TEC. EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS
DISCIPLINA: ALGORITMOS E ESTRUTURAS DE DADOS
PROFESSOR: RAMIDE DANTAS
ASSUNTO: PILHAS, FILAS E LISTAS

Aluno (a):			
Matrícula:		**Data:**	

Prática 04

Parte 0: Preparação

Passo 1: Crie um novo projeto chamado **Pratica4**.

> Vários arquivos fonte acompanham esta prática, cada um contendo seu próprio `main()`. No Eclipse, será necessário renomear a função `main()` (ex.: `mainPilha()`) de uma parte de forma para poder testar outra.

Parte 1: Trabalhando com pilhas

Passo 1: Adicione os arquivos **pilha.cpp** e **polonesa.cpp** que acompanham a prática ao projeto.

> **pilha.cpp:** realiza uma série de testes para validar o funcionamento da pilha.
>
> **polonesa.cpp**: implementa uma calculadora polonesa simples baseada em pilha, conforme explicado em sala de aula.

Passo 2: Crie um arquivo chamado **pilha.h** e implemente nele a classe `Pilha` como a seguir:

```
template <class T>
class Pilha {
private:
       // Atributos para array de items, capacidade e topo da pilha
public:
       Pilha(int capacidade) {
         // instancia array de items, inicializa capacidade e topo
       }

       ~Pilha() {
         // destroy array de items
       }

       void empilha(T item) {
         // empilha um item no topo da pilha; lança "Estouro da pilha" se cheia
       }

       T desempilha() {
         // remove um item do topo da pilha; lança "Pilha vazia" se vazia
       }

       int tamanho() {
         // retorna o número de elementos na pilha.
       }
};
```

> Dê implementações adequadas às funções acima.

Passo 3: Compile e teste a aplicação, verificando se o resultado é o esperado.

> Rode o **pilha.cpp** primeiro para verificar sua implementação da Pilha. Depois teste o **polonesa.cpp** para ver se o resultado gerado é o esperado.

Passo 4: (Desafio/Opcional) Tente usar a classe `std::stack` da **STL** (`<stack>`) no programa.

Parte 2: Trabalhando com Filas

Passo 1: Adicione os arquivo **fila.cpp** e **impressora.cpp** ao projeto **Pratica4**.

fila.cpp: realiza uma série de testes para validar o funcionamento da fila.

impressora.cpp: contém uma função `main()` que simula uma fila de impressão, onde um usuário submete documentos, que aguardam na fila até que a impressora possa imprimi-los. A fila atual como um buffer, permitindo ao usuário e impressora trabalharem de forma paralela. (É preciso configurar o projeto para usar C++11.)

Passo 2: Crie o arquivo **fila.h** e implemente a classe `Fila` conforme a declaração a seguir:

```
template <class T>
class Fila {
private:
       // array de itens, capacidade, tamanho, posição inicial, etc.
public:
       Fila(int cap) {
         // inicializar array de items, capacidade, tamanho, posição inicial
       }

       ~Fila() {
         // destruir array de itens
       }

       void enfileira(const T & item) {
         // adiciona um item ao final da fila; lança "Fila cheia" caso cheia
       }

       T desenfileira() {
         // remove um item do inicio da fila; lança "Fila vazia" caso vazia
       }

       int cheia() {
         // retorna 1 se cheia, 0 caso contrário
       }

       int vazia() {
         // retorna 1 se vazia, 0 caso contrário
       }

       int tamanho() {
         // retorna a quantidade de itens atualmente na fila
       }
};
```

A fila deve usar um "buffer circular" conforme explicado no material de aula. Na dúvida, implemente um array simples e progrida até obter um buffer circular.

Passo 3: Compile e teste a aplicação, verificando se o resultado é o esperado.

fila.cpp: deve rodar e exibir OK para todos os testes.

impressora.cpp: nesse código há um laço infinito: a cada volta o usuário tem uma probabilidade de 70% de submeter um novo documento (adicionar à fila) e a impressora 50% de chance de imprimir (tirar da fila). Nessa configuração, chega um momento em que a fila está cheia e o usuário não consegue adicionar mais documentos. Só depois que a impressora retira um item que um novo pode ser adicionado. Modifique esses valores de forma que o fila fique quase sempre vazia.

Passo 4: (Desafio/Opcional) Modifique o programa para usar `std::queue` da **STL** (`<queue>`).

Parte 3: Trabalhando com Listas

Passo 1: Adicione o arquivo **lista.cpp** ao projeto **Pratica4**.

Esse arquivo contém uma função `main()` que realiza testes para verificar se a lista está implementada corretamente.

Passo 2: Crie o arquivo **lista.h** e implemente a classe `Lista` conforme a declaração a seguir:

```
template <class T>
class Lista {
private:
       // itens da lista, capacidade e tamanho atual
public:
       Lista(int capacidade) {
         // inicilização do array, capacidade e tamanho
       }
       ~Lista() {
         //destruição do array
       }

       void adiciona (const T & item) {
         // adiciona um item ao final da lista; lança "Lista cheia" caso cheia
       }

       T pega(int idx) {
         // pega um item pelo indice (começa em 1);
         // lança "Item inválido" se posição inválida
       }

       void insere (int idx, const T & item) {
         // insere um item na posição indicada (a partir de 1).
         // lança "Lista cheia" caso cheia
         // lança "Item inválido" se posição inválida
         // desloca itens existentes para a direita
       }

       void remove(int idx) {
         // remove item de uma posição indicada
         // lança "Item inválido" se posição inválida
         // desloca itens para a esquerda sobre o item removido
       }

       void exibe() {
         // exibe os itens da saida padrão separados por espaços
       }

       int tamanho() {
         // retorna a quantidade de itens atualmente na lista
       }
};
```

Atenção aos métodos `insere()` e `remove()` pois eles precisam deslocar os itens existentes no array para direita (`insere()`) ou para a esquerda (`remove()`).

ATENÇÃO: A lista deve ser indexada a partir de 1, e não de 0 como em arrays. Isto é, numa lista com capacidade 10, os índices dos elementos vão de 1 a 10.

Passo 3: Compile e teste a aplicação, verificando se o resultado é o esperado.

Faça modificações nas adições, inserções e remoções para testar a lista.

Passo 4: (Desafio/opcional) Modifique o programa para usar `std::list` da **STL** (`<list>`).