



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

OBS: Essa prática faz uso de conhecimentos exercitados nas Práticas 1, 2 e 3.

Parte 0: Preparação

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

Esta prática usa vários códigos que acompanham a práticas, cada um contendo sua própria função `main()`. Pode ser necessário comentar a função `main()` de uma parte de forma que a outra compile e rode.

Parte 1: Trabalhando com pilhas

Passo 1: Adicione o arquivo **polonesa.cpp** que acompanha esta prática ao projeto.

Esse arquivo implementa uma calculadora polonesa simples baseada em pilha, conforme explicado em sala de aula. Esse arquivo inclui **pilha.h**, criado a seguir.

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

```
template <class T>
class Pilha {
private:
    // propriedades 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
    }
};
```

Dê implementações adequadas às funções. Em caso de problemas usando templates, faça usando `int` e depois adapte para funcionar com templates.

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

Modifique o programa, adicionando novas expressões e verificando o resultado.

Passo 4: (Desafio) Modifique o programa para usar a classe `std::stack` da **STL** (`<stack>`).

Parte 2: Trabalhando com Filas

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

Esse arquivo 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 atua como um buffer, permitindo ao usuário e impressora trabalharem de forma paralela. Esse arquivo usa **fila.h**, criado no próximo passo.

(É preciso configurar o projeto para usar o padrão C++11 para compilar.)

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 itens, capacidade, tamanho, posição inicial
    }

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

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

    T dequeue() {
        // remove um item do início 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 para tornar um buffer circular.

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

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 a fila fique quase sempre vazia.

Passo 4: (Desafio) Modifique o programa para usar a classe `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 operações de inserção, remoção e exibição em uma lista de caracteres simples. Utiliza o arquivo **lista.h** criado a seguir.

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) {
        // inicialização 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 exhibe() {
        // exhibe os itens da saída padrão separados por espaços
    }
};
```

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 nos arrays.

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) Modifique o programa para usar a classe `std::list` da **STL** (`<list>`).