IMD0029 - Estrutura de Dados Básicas 1 – 2018.1
Prof. Eiji Adachi M. Barbosa
Atividade Avaliativa Prática em Laboratório – Unidade 2

ANTES DE COMEÇAR, leia atentamente as seguintes instruções:

- Esta é uma atividade de caráter individual e sem consultas a pessoas ou material (impresso ou eletrônico).
- A atividade vale 5,0 pontos na 1ª unidade. O valor de cada questão é informado no seu enunciado.
- Celulares e outros dispositivos eletrônicos devem permanecer desligados durante toda a prova.
- Desvios éticos ou de honestidade levarão à anulação da atividade do candidato (nota igual a zero).
- Junto a este enunciado, você também recebeu uma estrutura de diretórios contendo um diretório para cada questão. Em cada um destes diretórios, já existe uma assinatura de função e uma função main com um pequeno teste executável. A solução da sua questão deverá seguir a assinatura da função já estabelecida. Ou seja, não mude esta assinatura. Se necessário, crie funções auxiliares com outras assinaturas, mas **não mude a assinatura da função original!**

Questão 1 (1,0 ponto): Considere a classe `LinkedList` no diretório /q1, a qual implementa uma lista simplesmente encadeada que possui apenas um ponteiro para o primeiro elemento (`first`) da lista. Para esta lista, implemente o método `LinkedList::invert()`, o qual inverte a ordem dos nós da lista simplesmente encadeada. Este método deverá ser obrigatoriamente recursivo.
Obs.: Antes de implementar o método `invert`, é necessário implementar o método `insertBegin`.

Questão 2 (1,5 ponto): Muitas aplicações acessam um mesmo dado em curtos períodos de tempo. Para este tipo de aplicação, uma estratégia para otimizar o acesso aos dados comumente implementada em listas encadeadas é a estratégia de contagem de acessos (*Count Frequency*). Nesta estratégia, mantém-se um atributo extra em cada nó para contar o número de vezes que aquele nó foi acessado. Desta forma, a cada vez que um nó é buscado com sucesso, o seu atributo que conta o número de acessos é incrementado. Além disso, após cada busca realizada com sucesso na lista, os nós devem ser rearranjados de modo que fiquem ordenados em ordem decrescente em relação ao número de acesso dos nós. Nesta questão, a classe `LinkedList` do diretório /q2 define a estrutura básica para uma lista duplamente encadeada com sentinelas cabeça (`Head`) e cauda (`Tail`). Implemente o método `LinkedList::searchCF(string key)`, o qual realiza a busca seguindo a estratégia de contagem de acessos.
Obs.: Antes de implementar o método `searchCF`, é necessário implementar o método `insertEnd`.

Questão 3 (1,0 ponto): Considere a classe `Queue` do diretório /q3, a qual define a estrutura básica para uma Fila. Nesta questão, use os conceitos de lista simplesmente encadeada para implementar a classe `Queue` de modo que os métodos `Queue<T>::peek()`, `Queue<T>::enqueue(T element)` e `Queue<T>::dequeue()` sigam as regras de uma Fila e tenham complexidade assintótica O(1). Em outras palavras, implemente na classe `Queue` uma lista simplesmente encadeada que se comporta como uma Fila através dos métodos `peek`, `enqueue` e `dequeue`.

Questão 4 (1,5 ponto): Uma aplicação bastante comum em analisadores de texto, como compiladores e interpretadores, é verificar se uma dada sequência de caracteres especiais está bem formada. Uma sequência de parênteses, colchetes e chaves é bem formada quando o abre-fecha de parênteses, colchetes e chaves é feito na ordem e quantidade corretas. Por exemplo, a sequência de parênteses e colchetes a seguir é bem formada `[ ( ) ( ( ) ) ]` enquanto a sequência `[ ( ] )` não é bem formada. A primeira sequência é bem formada pois o abre-fecha de parênteses, colchetes e chaves está correto, enquanto na segunda sequência o colchete foi fechado antes de se fechar o parênteses. Nesta questão, implemente a função `isWellFormed(string str)`, a qual recebe uma string representando uma sequência de parênteses, colchetes e chaves e verifica se esta sequência é bem formada ou não. Use uma Pilha (`Stack`) da STL para resolver esta questão.

Obs.: Pode assumir nesta questão que os caracteres da string de entrada serão apenas parênteses, colchetes ou chaves. Assuma também que parênteses, colchetes e chaves podem aparecer em qualquer ordem, isto é, não existe uma ordem pré-estabelecida de prioridade entre estes elementos em sequências bem formadas.

ENTREGÁVEL

O entregável desta atividade deverá seguir a mesma estrutura de diretórios do código fonte que você recebeu com este enunciado, obviamente, contendo os arquivos fonte utilizados para construir sua solução nos diretórios de cada questão. Além disso, o diretório pai deverá ter o seu nome e matrícula, seguindo o padrão `<PRIMEIRO_NOME>_<SOBRENOME>-<MATRICULA>` (Dica: basta renomear o diretório /src com o padrão definido anteriormente). Por exemplo:

```
> JOAO_SILVA-200012345
    > q1
    > q2
    > q3
    > q4
```

Toda esta estrutura de diretórios, incluindo os arquivos fonte com sua solução, deverá ser compactada num arquivo .zip que também deverá seguir o padrão `<PRIMEIRO_NOME>_<SOBRENOME>-<MATRICULA>.` Este arquivo compactado deverá ser entregue via SIGAA até as **20:25**. **Este é um prazo fixo que não será estendido**, exceto em casos muito excepcionais (ex.: SIGAA fora do ar). Ou seja, entregas após este horário não serão aceitas. A atividade do SIGAA permite apenas um envio, portanto certifique-se de que está enviando a versão correta antes de anexar ao SIGAA.

CRITÉRIOS DE CORREÇÃO

Para a correção desta atividade, serão levados em consideração, dentre outros, os seguintes pontos:

- Obediência às regras definidas para as assinaturas de função e para o entregável (arquivo .zip), conforme especificado no enunciado desta atividade
- Existência de erros ou warnings de compilação do código fonte[1]
- Programas executam sem apresentar falhas e produzem os resultados esperados
- Soluções atendem critérios de complexidade, caso estabelecido no enunciado
- Apresentação e organização do código fonte entregue (identação, nome das variáveis, modularização do código em funções, etc)

Obs.: Para cada questão, já há uma função main com um pequeno teste executável. Este é um teste simples que **não garante** a corretude da sua implementação. Ou seja, se sua implementação passou no teste executável disponibilizado junto a este enunciado, isto não é garantia de que ela está totalmente correta. Para fins de correção, eu utilizarei outra bateria de testes mais completa, além de analisar manualmente o código produzido.

[1] Compile usando as flags `–Wall –pedantic –std=c++11`