INF 213 - Roteiro da Aula Pratica 8

→ LEMBREM-SE DE USAR PAPEL E CANETA COMO RASCUNHO ANTES DE IMPLEMENTAR <<--

Arquivos fonte e diagramas utilizados nesta aula:
https://drive.google.com/file/d/1mVsGBG0FnsllaKsAmizCc5BthOemEJKe/view?usp=sharing

Etapa 1

O arquivo MyList2NewIterator.h contém a implementação parcial da lista por contiguidade vista em sala de aula. Porém, parte do código foi removido (na implementacao do iterador de pos-incremento/decremento). Sua tarefa consiste em completar o codigo e testar a classe utilizando o programa Testes.cpp .

Utilize um Debugger de memória (http://valgrind.org/docs/manual/quick-start.html) para verificar se sua implementação não contem erros de memória.

Etapa 2

Implemente a funcao size() utilizando **iteradores** para contar o número de elementos em cada chamada. Faca o mesmo com a função empty(). (infelizmente você terá que tirar a palavra-chave const dessas funcões, já que não temos iteradores de acesso constante nessa implementacão)

Etapa 3

Utilizando iteradores para a nossa classe de listas duplamente encadeadas, crie uma funcao não-membro chamada “reverse” (termine a implementacao no início do arquivo Testes.cpp). Tal funcao devera receber como argumento uma lista (disponivel no arquivo MyList2NewIterator.h) e inverter a ordem dos elementos nela. Essa operacao devera ser realizada utilizando apenas iteradores e as funcoes begin() e end() da nossa lista (**não utilize outras funcoes da classe MyList2 -- exceto as funcoes begin() e end()**).

A funcao reverse devera ser implementada no arquivo Testes.cpp.

Dicas: um desafio e’ conseguir um iterador para o ultimo elemento da lista.

Etapa 4

Na nossa implementacao de iteradores para listas duplamente encadeadas (arquivo MyList2NewIterator.h), e’ possivel incrementar e decrementar os iteradores. Porem, como o iterador representando o final da lista (end()) possui internamente um ponteiro apontando para NULL, não e’ possivel decrementar esse iterador.

Desenhe um diagrama de lista duplamente encadeada em uma folha de papel para entender melhor o motivo de não ser possivel decrementar tal iterador.

Em C++ (na STL), por outro lado, o iterador para listas suporta a operacao de decremento mesmo se ele estiver apontando para o final da lista.

Modifique nossa implementacao de iterador (disponivel no arquivo MyList2NewIterator.h) para que o comportamento seja similar ao dos iteradores da STL.

Obs:
- seu metodo deve funcionar mesmo se a lista for modificada após a criacao do iterador (conforme testado em Testes.cpp)! Essa etapa exige um pouco de criatividade (se precisar de ajuda, não exite em postar no Moodle ou perguntar ao professor durante a aula pratica).
- faca sempre diagramas para entender melhor o comportamento das estruturas de dados!
- a possibilidade de decrementar o end() pode facilitar varias operacoes com listas (exemplo: achar o último elemento de uma lista rapidamente).

Submissao da aula pratica:
A solucao deve ser submetida ate as 18 horas da proxima Segunda-Feira utilizando o sistema submitty (submitty.dpi.ufv.br). Envie todos seus arquivos (o .h da classe e o de testes). Atualmente a submissao so pode ser realizada dentro da rede da UFV.