

INF 213 - Roteiro da Aula Prática 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 implementação do iterador de pos-incremento/decremento). Sua tarefa consiste em completar o código 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 contém erros de memória.

Etapa 2

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

Etapa 3

Utilizando iteradores para a nossa classe de listas duplamente encadeadas, crie uma função não-membro chamada "reverse" (termine a implementação no início do arquivo Testes.cpp). Tal função deverá receber como argumento uma lista (disponível no arquivo MyList2NewIterator.h) e inverter a ordem dos elementos nela. Essa operação deverá ser realizada utilizando apenas iteradores e as funções begin() e end() da nossa lista (**não utilize outras funções da classe MyList2 -- exceto as funções begin() e end()**).

A função reverse deverá ser implementada no arquivo Testes.cpp.

Dicas: um desafio é conseguir um iterador para o último elemento da lista.

Etapa 4

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

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

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

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

Obs:

- seu método deve funcionar mesmo se a lista for modificada após a criação 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 prática).
- faça sempre diagramas para entender melhor o comportamento das estruturas de dados!
- a possibilidade de decrementar o `end()` pode facilitar várias operações com listas (exemplo: achar o último elemento de uma lista rapidamente).

Submissão da aula prática:

A solução deve ser submetida até as 18 horas da próxima Segunda-Feira utilizando o sistema submittity (submittity.dpi.ufv.br). Envie todos seus arquivos (o `.h` da classe e o de testes). Atualmente a submissão só pode ser realizada dentro da rede da UFV.