Lista circular duplamente encadeada com nó sentinela

I) Introdução

A estrutura de dados descrita neste relatório representa uma sequência duplamente encadeada de elementos. Cada elemento consiste em uma estrutura contendo uma variável do tipo inteiro, representado no programa como do tipo Item e duas variáveis que apontam para outros elementos da lista, uma aponta para o elemento anterior da sequência e a outra variável aponta para o elemento posterior, sendo portanto, uma lista duplamente encadeada.

II) Descrição da lista

A struct que representa os nós da lista é implementado no cabeçalho Node.h, a variável *data* do tipo Item definido como um inteiro pela palavra chave *typedef* representa o conteúdo que o nó armazena, a variável data pode armazenar outros tipos

de dados como strings, char, double, bastando somente modificar o *typedef* sem a necessidade do programador realizar mudanças em todo o programa. Além disso, as variáveis *previous* e *next* são ponteiros que apontam respectivamente para o nó anterior e posterior da lista, oferecendo a lista implementada a característica de ser duplamente encadeada.

Uma representação gráfica da struct Node é demonstrada abaixo:

Na implementação da classe List no arquivo List.h foi definido como atributos privados as variáveis *m_size* que representa o número de nós na lista e a variável apontadora *head* que aponta para o primeiro nó da lista, o nó sentinela, que não possui variável para armazenar valor, contém apenas as variáveis apontadoras *next* que aponta para o primeiro nó que contém dado armazenado na lista e a variável *previous* que aponta para o último nó que armazena dado na lista, dando a lista implementada a característica de ser circular.

Como atributos públicos da classe List estão as funções. Abaixo estão descritas todas as funções implementadas na classe List e suas funcionalidades.

List ( );

Construtor default da classe List, cria uma lista vazia com o atributo *head* da classe apontando para o nó sentinela, este nó é alocado dinamicamente com suas variáveis *previous* e *next* apontando para *nullptr*.

void clear ( );

Função que esvazia a lista deletado todos os nós da lista alocados dinamicamente deixando apenas o nó sentinela da lista e atribuindo à variável *head* da classe o valor 0, indicando que a lista não contém elementos.

List (const List& lista);

Esta função recebe como parâmetro uma referência para o objeto *lista* do tipo List e cria uma nova lista que contém os mesmos elementos da lista passada como parâmetro.

bool empty ( ) const;

Esta função retorna um valor booleano *true* se a lista estiver vazia, ou seja, não conter nenhum elemento ou o valor booleano *false* caso contrário.

size_t size ( ) const;

Esta função retorna o valor armazenado no atributo *m_size* da lista contendo o número de elementos da lista.

Item& front ( );

Esta função retorna uma referência para o primeiro elemento da lista.

Item& back ( );

Esta função retorna uma referência para o último elemento da lista.

void push_back (const Item& data);

Esta função recebe como parâmetro uma referência do objeto *data* do tipo Item, aloca dinamicamente um nó com o mesmo dado passado como parâmetro e insere o nó no final da lista.

void pop_front ( );

Esta função deleta o primeiro elemento da lista alocado dinamicamente e usa o nó sentinela para apontar para o próximo nó da lista.

void pop_back ( );

Esta função deleta o último elemento da lista alocado dinamicamente e usa o nó sentinela para apontar para o nó anterior ao nó deletado da lista.

Item removeAt (int index);

Esta função remove da lista o elemento alocado dinamicamente de índice igual ao valor passado como parâmetro armazenado na variável *index* e retorna este valor.

void removeAll (const Item& data);

Esta função remove da lista todos os elementos iguais ao objeto *data* passado como referência do tipo *Item* passado como parâmetro.

void swap (List& lst);

Troca o conteúdo da lista com o conteúdo da lista objeto *lst* passado como referência do tipo *List* passado como parâmetro.

List *copy ( );

Retorna um ponteiro para uma nova lista criada dinamicamente igual a lista objeto.

void append (Item vec[ ], int n);

Insere na lista todos os nós criados dinamicamente contendo os elementos do vetor de tipo *Item* passado por referência de tamanho *n*.

bool equals (const List& lst);

Retorna o valor booleano *true* se a lista for igual à lista objeto *lst* passada como referência do tipo *List* passada como parâmetro ou retorna *false* caso o contrário.

void merge (List& lst);

Insere na lista objeto os elementos de forma intercalada da lista objeto *lst* passada como referência do tipo *List* passada como parâmetro. Depois disso, a lista *lst* será uma lista vazia, contendo apenas o nó sentinela.

friend std::ostream& operator<< (std::ostream& out, const List& lst);

Operador de extração sobrecarregado de forma global como função amiga.

Item& operator[ ] (int index);

Operador sobrecarregado de indexação, retorna uma referência do tipo *Item* para o elemento na lista de índice *index*.

List& operator= (const List& lst);

Operador sobrecarregado de atribuição. Nesta função os elementos da lista objeto *lst* passada como referência do tipo *List* passada como parâmetro são atribuídos à lista objeto da função.

~List ( );

Destrutor. Esta função remove toda a memória alocada dinamicamente utilizada na criação da lista.

III) Referências bibliográficas

https://www.cplusplus.com/reference/ios/ios_base/setf/