**CSC 2201 – Computer Science II**

**Lab #05**

**Take-Home**

**60 Points**

1. Save a copy of this document with your name and the assignment number somewhere in the file name. For example, the file name *“Jane\_Doe\_CSC2200\_Lab1.docx”*
2. Copy-and-paste your answers (e.g., C++ source code) into the document.
3. Copy-and-paste the program output window.
4. Submit the following files separately (do not compress the files) to the Blackboard in one submission:

1) This document as a word document (i.e., with the extension ***.doc*** or ***.docx***).

2) All C++ source code solution file(s) (only the ***.cpp*** and ***.h*** files) to the Canvas item associated with this assignment/lab solution. ***\*\*If you modified it, submit it***

\*Submit entire Visual Studio solution, if possible, otherwise only the required files.

Questions:

- implement the LinkedList ADT [the declaration is given in ListLinked.h]

- implement the following operations:

- remove, replace, clear; (30 pts)

- isFull, isEmpty; (10 pts)

- gotoBeginning, gotoEnd, gotoNext, gotoPrior, getCursor; (20 pts)

- finish Test Plan 5-1, and change data type from a character to an integer (change the value of LAB5\_TEST1 from 0 to 1 in config.h) to finish Test Plan 5-2;

\* **Copying-and-pasting your *C++ program code* to a Word document**

1) From within the Visual Studio program, press **CTRL-A** and press **CTRL-C**.

2) From within the Word document, press **CTRL-V**.

#include "ListLinked.h"

#include <iostream>

//ListNode constructor for nodes in ListLinked

template <typename DataType>

List<DataType>::ListNode::ListNode(const DataType& nodeData, ListNode\* nextPtr)

{

this->dataItem = nodeData;

this->next = nextPtr;

}

//Default constructor for an empty list

template <typename DataType>

List<DataType>::List(int ignored)

{

head = nullptr;

cursor = nullptr;

}

//Copy elements from current List to List 'other'

template <typename DataType>

List<DataType>::List(const List& other)

{

head = nullptr;

cursor = nullptr;

if (!other.isEmpty()) {

ListNode\* otherCurrent = other.head;

while (otherCurrent != nullptr) {

insert(otherCurrent->dataItem);

otherCurrent = otherCurrent->next;

}

}

}

//= operator overloading

template <typename DataType>

List<DataType>& List<DataType>::operator=(const List& other)

{

if (this != &other)

{

//Clear current list

clear();

//Copy elements from 'other' List, if List is not empty

if (!other.isEmpty()) {

ListNode\* otherCurrent = other.head;

while (otherCurrent != nullptr) {

insert(otherCurrent->dataItem);

otherCurrent = otherCurrent->next;

}

}

}

return \*this;

}

//ListLinked destructor

template <typename DataType>

List<DataType>::~List()

{

clear();

}

//insert implementation for ListLinked

template <typename DataType>

void List<DataType>::insert(const DataType& newDataItem) throw (logic\_error)

{

if (isEmpty()) {

head = new ListNode(newDataItem, nullptr);

cursor = head;

}

else {

cursor->next = new ListNode(newDataItem, cursor->next);

cursor = cursor->next;

}

}

//remove implementation for ListLinked

template <typename DataType>

void List<DataType>::remove() throw (logic\_error)

{

if (!isEmpty()) {

if (cursor == head) {

ListNode\* temp = head;

head = head->next;

delete temp;

cursor = head;

}

else {

ListNode\* prevNode = head;

while (prevNode != nullptr && prevNode->next != cursor) {

prevNode = prevNode->next;

}

if (prevNode != nullptr) {

prevNode->next = cursor->next;

delete cursor;

cursor = prevNode->next;

}

}

}

}

//replace implementation for ListLinked

template <typename DataType>

void List<DataType>::replace(const DataType& newDataItem) throw (logic\_error)

{

if (!isEmpty()) {

cursor->dataItem = newDataItem;

}

}

//Clears ListLinked of elements

template <typename DataType>

void List<DataType>::clear()

{

while (!isEmpty()) {

remove();

}

}

//

template <typename DataType>

bool List<DataType>::isEmpty() const

{

return head == nullptr;

}

template <typename DataType>

bool List<DataType>::isFull() const

{

//Always assumes more memory is available

return false;

}

//Moves cursor to beginning of ListLinked

template <typename DataType>

void List<DataType>::gotoBeginning() throw (logic\_error)

{

cursor = head;

}

//Moves cursor to end of ListLinked

template <typename DataType>

void List<DataType>::gotoEnd() throw (logic\_error)

{

while (cursor != nullptr && cursor->next != nullptr) {

cursor = cursor->next;

}

}

//Moves cursor to next element of ListLinked

template <typename DataType>

bool List<DataType>::gotoNext() throw (logic\_error)

{

if (cursor != nullptr && cursor->next != nullptr) {

cursor = cursor->next;

return true;

}

return false;

}

//Moves cursor to previous element of ListLinked

template <typename DataType>

bool List<DataType>::gotoPrior() throw (logic\_error)

{

if (cursor != nullptr && cursor != head) {

ListNode\* prevNode = head;

while (prevNode != nullptr && prevNode->next != cursor) {

prevNode = prevNode->next;

}

if (prevNode != nullptr) {

cursor = prevNode;

return true;

}

}

return false;

}

//Returns dataItem at cursor location

template <typename DataType>

DataType List<DataType>::getCursor() const throw (logic\_error)

{

if (!isEmpty()) {

return cursor->dataItem;

}

throw logic\_error("List is empty");

}

//Moves current cursor element to beginning of ListLinked

template <typename DataType>

void List<DataType>::moveToBeginning() throw (logic\_error)

{

{

if (!isEmpty() && cursor != head) {

// Initialize pointers

ListNode\* prevNode = nullptr;

ListNode\* current = head;

//Traverse the list to find the last element

while (current->next != nullptr) {

prevNode = current;

current = current->next;

}

//Move all elements to the right by 1

prevNode->next = nullptr; //Disconnect the last element

current->next = head; //Connect the last element to the current first element

head = current; //Update the head to the last element

// Update the cursor to point to the newly positioned element

cursor = head;

}

}

}

//Inserts element before the cursor

template <typename DataType>

void List<DataType>::insertBefore(const DataType& newDataItem) throw (logic\_error)

{

if (!isEmpty() && cursor != head) {

ListNode\* prevNode = head;

while (prevNode != nullptr && prevNode->next != cursor) {

prevNode = prevNode->next;

}

if (prevNode != nullptr) {

prevNode->next = new ListNode(newDataItem, cursor);

}

}

}

template <typename DataType>

void List<DataType>::showStructure() const

{

if (isEmpty()) {

cout << "Empty list" << endl;

}

else {

for (ListNode\* temp = head; temp != nullptr; temp = temp->next) {

if (temp == cursor) {

cout << "[";

}

// Assumes that dataItem can be printed via << because

// it is either primitive or operator<< is overloaded.

cout << temp->dataItem;

if (temp == cursor) {

cout << "]";

}

cout << " ";

}

cout << endl;

}

}

\*\* **Copying-and-pasting a C++ “*output window”* to a Word document**

1) From the Visual Studio output window, press **ALT-PrintScreen**.

2) From within the Word document, press **CTRL-V**.

A screen shot of a computer

Description automatically generated