main.cpp

#include <iostream>

#include <string>

#include <iostream>

#include "IntList.h"

using namespace std;

int main() {

/// Create an instance of IntList.

IntList myList;

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n";

cout << "Testing the insertNode function, nodes inserted in this order: 12, 4, 3, 5, 77, 13\n";

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n";

/// Build a list.Will test the program with : sorted list 3,4,5,12,13,77

myList.insertNode(12);

myList.insertNode(4);

myList.insertNode(3); // beginning

myList.insertNode(5); //middle

myList.insertNode(77); //end

myList.insertNode(13);

/// Display the nodes.

cout << "Here are the values in myList:\n";

myList.print();

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n";

cout << "Testing the removeByPos function\n";

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n";

/// Remove node at position 0( beginning position ).

cout << "Removing node at position 0 which is the first node, 3 should be removed...\n";

myList.removeByPos(0);

myList.print();

/// Remove node at position 4 ( end position).List now 4,5,12,13,77

cout << "Removing node at position 4 which is the last node, 77 should be removed...\n";

myList.removeByPos(4);

myList.print();

/// Remove node at position 2 (middle position). List now 4,5,12,13

cout << "Removing node at position 2 which is a middle node, 12 should be removed...\n";

myList.removeByPos(2);

myList.print();

/// Try a position that is too big.List should remain the same

cout << "Removing node 99, a position that is too big.List should remain the same...\n";

myList.removeByPos(99);

myList.print();

cout << "\n\nAdding back the deleted nodes so we can rest the deleteNode function... \n\n";

myList.insertNode(12);

myList.insertNode(3);

myList.insertNode(77);

myList.print();

/// delete node.

cout << "\n\n Testing the deleteNode function\n\n";

cout << "Removing number 5, the first node, 5 should be removed...\n";

myList.deleteNode(5);

myList.print();

cout << "Removing number 77 which is at the end, 77 should be removed...\n";

myList.deleteNode(77);

myList.print();

cout << "Removing number 12 which is in the middle of the list, 12 should be removed...\n";

myList.deleteNode(12);

myList.print();

cout << "Removing number 100 which is not in the list, list should remain the same...\n";

myList.deleteNode(100);

myList.print();

cout << "\n\nAdding back the deleted nodes \n";

myList.insertNode(12);

myList.insertNode(5);

myList.insertNode(77);

myList.print();

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n";

cout << "Testing the reverse function\n";

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n";

myList.reverse();

myList.print();

return 0;

}

IntList.h

#ifndef IntList\_h

#define IntList\_h

struct ListNode {

int value;

struct ListNode\* next;

};

// Specification file for the IntList class

class IntList {

private:

ListNode\* head; // List head pointer

// Destroy function

void destroy();

public:

// Constructor

IntList() {

head = nullptr;

}

// Destructor

~IntList();

// List operations

void insertNode(int);

void deleteNode(int);

void print();

void reverse();

void removeByPos(int);

};

#endif

IntList.cpp

// Ryan Jacoby

#include<iostream>

#include "IntList.h"

/\*

\* Destroys list and safely unallocates it. Called by destructor and others.

\*/

void IntList::destroy() {

ListNode \* current, \* next = nullptr;

current = head;

while(current != nullptr) {

next = current->next; // Save next node so we can keep going once current node is

delete current; // deleted.

current = next;

}

head = nullptr; // Make sure head isn't pointing to unallocated memory.

}

/\*

\* Deallocates list.

\*/

IntList::~IntList() {

destroy();

}

/\*

\* Inserts node into sorted list, keeps list sorted

\*

\* @param Number to insert

\*/

void IntList::insertNode(int n) {

ListNode \* newNode, \* current, \* previousNode = nullptr;

newNode = new ListNode;

newNode->value = n;

if(!head) {

head = newNode; // If list is empty, make this the only node.

newNode->next = nullptr;

} else {

current = head;

previousNode = nullptr;

while(current != nullptr && current->value < n) { // Find insertion location.

previousNode = current;

current = current->next;

}

if(previousNode == nullptr) { // Link current node to new node and new node to old next.

head = newNode;

newNode->next = current;

} else {

previousNode->next = newNode;

newNode->next = current;

}

}

}

/\*

\* Deletes node with value passed. If it does not exist, it will return.

\*

\* @param Value of node to delete.

\*/

void IntList::deleteNode(int n) {

ListNode \* current, \* previous = nullptr;

if(!head) return;

if(head->value == n) { // Reassign head if necessary.

current = head->next;

delete head;

head = current;

} else {

current = head;

while(current != nullptr && current->value != n) { // Find node to delete.

previous = current;

current = current->next;

}

if(current) { // If we found it, link around it and unallocate node.

previous->next = current->next;

delete current;

}

}

}

/\*

\* Loop through list and print out values with a space in between.

\*/

void IntList::print() {

ListNode \* current = this->head;

while(current) {

std::cout << current->value << ' ';

current = current->next;

}

std::cout << '\n';

}

/\*

\* Reverse list by creating a new one, building it backwards, deleting the old one and fixing head.

\*/

void IntList::reverse() {

ListNode \* newHead = nullptr, \* newNode, \* current, \* tempPtr;

current = head;

while(current) { // Make new list with all values of old list.

newNode = new ListNode;

newNode->value = current->value;

newNode->next = nullptr;

if(newHead != nullptr) { // Save node so we can reverse link.

tempPtr = newHead;

newHead = newNode;

newNode->next = tempPtr;

} else {

newHead = newNode;

}

current = current->next;

}

destroy(); // Destroy old list(in head).

head = newHead; // Put new list in head.

}

/\*

\* Increment into list n amount if possible and remove node there.

\*

\* @param Integer index to remove node at

\*/

void IntList::removeByPos(int n) {

ListNode \* current = head;

if(current == nullptr) return;

while(n > 0 && current) {

current = current->next;

n--; // Find node to remove if it is within bounds.

}

if(current == head) { // Check if we went out of bounds.

head = head->next;

delete current;

return;

}

if(current == nullptr) return;

ListNode \* trailer = head; // Link around and unallocate.

while(trailer->next != current) trailer = trailer->next;

trailer->next = current->next;

delete current;

}

ListDriver.cpp

// Ryan Jacoby

#include <iostream>

#include <string>

#include <iostream>

#include "List.h"

using namespace std;

int main() {

/// Create an instance of List.

List<char> myList;

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n";

cout << "Testing the insertNode function, nodes inserted in this order: A, B, C, H, z, q\n";

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n";

/// Build a list.Will test the program with : sorted list A,B,C,H,q,z

myList.insertNode('A');

myList.insertNode('B');

myList.insertNode('C'); // beginning

myList.insertNode('H'); //middle

myList.insertNode('z'); //end

myList.insertNode('q');

/// Display the nodes.

cout << "Here are the values in myList:\n";

myList.print();

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n";

cout << "Testing the removeByPos function\n";

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n";

/// Remove node at position 0( beginning position ).

cout << "Removing node at position 0 which is the first node, A should be removed...\n";

myList.removeByPos(0);

myList.print();

/// Remove node at position 4 ( end position).List now 4,5,12,13,77

cout << "Removing node at position 4 which is the last node, z should be removed...\n";

myList.removeByPos(4);

myList.print();

/// Remove node at position 2 (middle position). List now 4,5,12,13

cout << "Removing node at position 2 which is a middle node, H should be removed...\n";

myList.removeByPos(2);

myList.print();

/// Try a position that is too big.List should remain the same

cout << "Removing node 99, a position that is too big.List should remain the same...\n";

myList.removeByPos(99);

myList.print();

cout << "\n\nAdding back the deleted nodes so we can rest the deleteNode function... \n\n";

myList.insertNode('H');

myList.insertNode('A');

myList.insertNode('z');

myList.print();

/// delete node.

cout << "\n\n Testing the deleteNode function\n\n";

cout << "Removing number C, the first node, C should be removed...\n";

myList.deleteNode('C');

myList.print();

cout << "Removing number z which is at the end, z should be removed...\n";

myList.deleteNode('z');

myList.print();

cout << "Removing number H which is in the middle of the list, H should be removed...\n";

myList.deleteNode('H');

myList.print();

cout << "Removing number Q which is not in the list, list should remain the same...\n";

myList.deleteNode('Q');

myList.print();

cout << "\n\nAdding back the deleted nodes \n";

myList.insertNode('H');

myList.insertNode('C');

myList.insertNode('z');

myList.print();

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n";

cout << "Testing the reverse function\n";

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n";

myList.reverse();

myList.print();

return 0;

}

UML Diagram

-----------------------

| List |

-----------------------

| -head : ListNode \* |

| -Destroy() |

-----------------------

| +List() |

| +~List() |

| +insertNode(T) |

| +deleteNode(T) |

| +print() |

| +reverse() |

| +removeByPos(int) |

-----------------------