Assignment 2

18:

int LList:: countNode() {

node\* temp = myfirst;

int count = 0;

while (temp != nullptr) {

temp = temp->next;

count++;

}

return count;

};

19: #ifndef LLIST\_H

#define LLIST\_H

#include<iostream>

using namespace std;

class LList {

private:

class node {

public:

int value;

node\* next;

node() :value(0), next(0) {}

node(int item) : value(item), next(0) {}

};

node\* myfirst;

int mysize;

node\* pos2ptr(int);

public:

LList() :myfirst(0), mysize(0) {}

LList(LList&);

~LList();

bool isEmpty();

void addAtEnd(int);

void addAtBeginning(int);

void addAtIndex(int, int);

void eraseAtIndex(int);

void eraseFirst();

void eraseEnd();//

bool find(int);

int size();

LList operator=(LList&); //overload

friend ostream& operator<<(ostream&, LList&);

bool isAcsend();

void reverseList();

int countNode();

};

CPP:

#include "LList.h"

#include<iostream>

using namespace std;

LList::LList(LList& L) {

if (L.myfirst == 0) myfirst = 0;

else {

myfirst = new node(L.myfirst->value);

node\* tempL = myfirst;

node\* tempR = L.myfirst;

while (tempR->next != 0) {

tempL->next = new node(tempR->next->value);

tempR = tempR->next;

tempL = tempL->next;

}

}

mysize = L.mysize;

}

LList::~LList() {

node\* temp;

while (myfirst != 0) {//1

temp = myfirst;

//2

myfirst = myfirst->next;

delete temp;

}

}

bool LList::isEmpty() { return (myfirst == 0); }

void LList::addAtEnd(int item) {

if (isEmpty()) { myfirst = new node(item); }

else {

node\* temp = myfirst;

while (temp->next != 0) temp = temp->next;

temp->next = new node(item);

}

mysize++;

}

void LList::addAtBeginning(int item) { //add item at the beginning of list

node\* temp = new node(item);

if (myfirst == 0) myfirst = temp;

else { temp->next = myfirst; myfirst = temp; }

mysize++;

}

void LList::addAtIndex(int pos, int item) {

if (!(pos < mysize && pos >= 0))cout << "invalid position" << endl;

else {

if (pos == 0) addAtBeginning(item);

else {

node\* ptr = pos2ptr(pos);

node\* temp = new node(item);

temp->next = ptr->next;

ptr->next = temp;

}

}

mysize++;

}

void LList::eraseFirst()

{

if (!isEmpty()) {

node\* temp = myfirst;

myfirst = myfirst->next;

delete temp;

mysize--;

}

}

void LList::eraseEnd() {

if (myfirst != 0) {

if (myfirst->next == 0) { delete myfirst; myfirst = 0; }

else {

node\* temp = myfirst;

while (temp->next->next != 0) { temp = temp->next; }

delete temp->next;

temp->next = 0;

}

mysize--;

}

else cout << "Can not erase, the list is empty" << endl;

}

bool LList::find(int item) {

node\* temp = myfirst;

bool found = false;

while (temp != 0) {

if (temp->value == item) { found = true; break; }

temp = temp->next;

}

return found;

}

int LList::size() { return mysize; }

LList LList::operator=( LList& node\_)

{

if (this == &node\_) {

return \*this;

}

else {

this->~LList();

LList::LList(node\_);

return \*this;

}

}

// helper function pos2ptr returns a pointer to node predecessor to the index node

LList::node\* LList::pos2ptr(int pos) {

node\* temp = myfirst;

int counter = 1;

while (counter < pos) { temp = temp->next; counter++; }

return temp;

}

ostream& operator<<(ostream& out, LList& L) {

LList::node\* temp = L.myfirst;

while (temp != 0) {

out << " " << temp->value;

temp = temp->next;

}

cout << endl;

return out;

}

void LList::eraseAtIndex(int index)

{

if (!(index < mysize && index >= 0)) {

std::cout << "invalid index." << std::endl;

}

else {

if (index == 0) { eraseFirst(); }

else if (index == mysize - 1) {

eraseEnd();

}

else {

node\* ptr = pos2ptr(index);

node\* pre = pos2ptr(index - 1);

pre->next = ptr->next;

delete ptr;

mysize--;

}

}

}

void LList::reverseList()

{

if (myfirst == nullptr || myfirst->next == nullptr) {

return;

}

node\* pre = nullptr;

node\* curr = myfirst;

node\* next\_ = nullptr;

while (curr != nullptr) {

next\_ = curr->next;

curr->next = pre;

pre = curr;

curr = next\_;

}

myfirst = pre;

}

bool LList::isAcsend()

{

if (myfirst == nullptr) {

return true;

}

for (node\* t = myfirst; t->next != nullptr; t = t->next) {

if (t->value >= t->next->value) {

cout << "It's not Acsend" << std::endl;

return false;

}

}

cout << "It's Ascend" << std::endl;

return true;

}

int LList:: countNode() {

node\* temp = myfirst;

int count = 0;

while (temp != nullptr) {

temp = temp->next;

count++;

}

return count;

};

20: void LList::reverseList()

{

if (myfirst == nullptr || myfirst->next == nullptr) {

return;

}

node\* pre = nullptr;

node\* curr = myfirst;

node\* next\_ = nullptr;

while (curr != nullptr) {

next\_ = curr->next;

curr->next = pre;

pre = curr;

curr = next\_;

}

myfirst = pre;

}

21: bool LList::isAcsend()

{

if (myfirst == nullptr) {

return true;

}

for (node\* t = myfirst; t->next != nullptr; t = t->next) {

if (t->value >= t->next->value) {

cout << "It's not Acsend" << std::endl;

return false;

}

}

cout << "It's Ascend" << std::endl;

return true;

}

THE DISPLAY A screenshot of a computer

Description automatically generated with medium confidence