//HashT.h start here

#pragma once

#include "LList.h"

class HashT

{

private:

static const int Cap = 10;

LList HTable[Cap];

public:

HashT();

HashT(HashT&);

~HashT();

int hash(int);

void add(int);

bool lookup(int);

void erase(int);

};

//HashT.h end here

//HastT.cpp start here

#include "HastT.h"

HashT::HashT()

{

}

HashT::HashT(HashT& h)

{

if (this != nullptr)

{

this->~HashT();

}

for (int i = 0; i < Cap; i++)

{

HTable[i] = h.HTable[i];

}

}

HashT::~HashT()

{

for (int i = 0; i < Cap; i++)

{

HTable[i].~LList();

}

}

int HashT::hash(int item)

{

return item%Cap;

}

void HashT::add(int item)

{

if (item >= 10)

{

HTable[hash(item)].addAtEnd(item);

}

}

bool HashT::lookup(int item)

{

if (HTable[hash(item)].find(item)) {

return true;

}

cout << "Item not found!" << endl;

return false;

}

void HashT::erase(int item)

{

if (lookup(item))

{

HTable[hash(item)].DelAtItem(item);

}

else

{

cout << "Item cannot be found" << endl;

}

}

//HashT.cpp end here

//LList.h start here

#pragma once

#include <iostream>

using namespace std;

class LList {

private:

class node

{

public:

int value; //the part that store value

node\* next; //the pointer to the next node

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

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

};

int mySize;

node\* myFirst;

node\* pos2ptr(int);

public:

LList() :myFirst(0), mySize(0) {}; //Default construtor

LList(LList&);//copy constructor

~LList();//destructor

bool isEmpty(); //????

void eraseAtFirst();//erase a node at first

bool find(int);//find an integer (or an item)

void addAtEnd(int);//add a node at the end

friend ostream& operator<< (ostream&, LList&); //overload the << operator

void eraseAtIndex(int);

void addAtIndex(int, int); //Index first, value after

void addAtBegin(int);//add at beginning, input a value

LList operator=(LList&);

int countNode(); //count the nodes

node\* search(int item); //search for an item

void reverse();

bool isASC(); //Check if in order of increasing or not

void DelAtItem(int); //Input an item, delete when item found

};

//LList.h end here

//LList.cpp start here

#include "LList.h"

LList::LList(LList& list)//copy constructor and add the copied node to the end

{

if (list.myFirst == 0)

{

myFirst = 0;

}

else

{

node\* temp = list.myFirst;

myFirst = new node(list.myFirst->value);

node\* copyitem = myFirst;

while (temp -> next != 0)

{

copyitem->next = new node(temp->next-> value);

temp = temp->next;

copyitem = copyitem->next;

}

}

mySize = list.mySize;

};

LList::node\* LList::pos2ptr(int index)

{

if (0 > index || index > mySize)

{

std::cout << "Invalid index!";

return NULL;

}

else

{

node\* temp = myFirst;

//temp = temp->next;

for (int i = 0; i < index; i++)

{

temp = temp->next;

}

return temp;

}

}

LList::~LList()//destructor

{

node\* temp; //Step 1

while (myFirst != 0)

{

temp = myFirst; //Step 2

myFirst = myFirst->next;

delete temp;

}

}

bool LList::isEmpty() //????

{

return (myFirst == nullptr);

}

void LList::eraseAtFirst()//erase a node at first

{

node\* temp;

temp = myFirst;

myFirst = myFirst->next;

mySize--;

}

bool LList::find(int item)//find a node

{

node\* temp = myFirst;

bool found = false;

while (temp != 0)

{

if (temp->value == item)

{

found = true;

break;

}

temp = temp->next;

}

return found;

}

void LList::addAtEnd(int item)//add a node at the end

{

if (isEmpty())

{

myFirst = new node(item);

}

else {

node\* temp = myFirst;

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

temp->next = new node(item);

}

mySize++;

}

ostream& operator << (ostream& out, LList& list) //overload the << operator

{

LList::node\* temp = list.myFirst;

while (temp != nullptr)

{

std::cout << temp->value << " ";

temp = temp->next;

}

return out;

}

void LList::eraseAtIndex(int index)

{

if (index > 0 && index <= mySize)

{

node\* deletenode = pos2ptr(index);

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

temp->next = deletenode->next;

deletenode = temp;

mySize--;

}

else if (index == 0)

{

eraseAtFirst();

}

else

{

std::cout << "Invalid index!" << std::endl;

}

}

void LList::addAtIndex(int index, int value)

{

if (index >= 0 && index <= mySize)

{

if (index == 0)

{

addAtBegin(value);

}

else

{

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

node\* newnode = new node(value);

newnode->next = temp->next;

temp->next = newnode;

mySize++;

}

}

else

{

std::cout << "Invalid index! ";

}

};

void LList::addAtBegin(int value)

{

node\* newnode = new node(value);

if (myFirst == 0)

{

myFirst = newnode;

}

else

{

newnode->next = myFirst;

myFirst = newnode;

}

mySize++;

}

LList LList::operator=(LList& list)

{

if (list.myFirst == 0)

{

myFirst = 0;

}

else

{

node\* temp = list.myFirst;

myFirst = new node(list.myFirst->value);

node\* copyitem = myFirst;

while (temp->next != 0)

{

copyitem->next = new node(temp->next->value);

temp = temp->next;

copyitem = copyitem->next;

}

}

mySize = list.mySize;

return \*this;

}

int LList::countNode()

{

int count=0;

node\* temp = myFirst;

while (temp != 0)

{

count++;

temp = temp->next;

}

return count;

}

LList::node\* LList::search(int item)

{

node\* temp = myFirst;

while (temp != 0)

{

if (temp->value == item)

{

return temp;

break;

}

else

{

temp = temp->next;

}

}

cout << "Item cannot be found! " << endl;

return nullptr;

}

void LList::reverse()

{

node\* temp = myFirst;

node\* prev = 0;

node\* next = 0;

while (temp != 0)

{

next = temp->next;

temp->next = prev;

prev = temp;

temp = next;

}

myFirst = prev;

}

bool LList::isASC()

{

node\* temp = myFirst;

for (int i=0; i< mySize -1; i++)

{

if (temp->value <= temp->next->value)

{

temp = temp->next;

}

else

{

return false;

}

}

return true;

}

void LList::DelAtItem(int item)

{

int i = 0;

node\* temp = myFirst;

while (temp->value != item)

{

temp = temp->next;

i++;

}

eraseAtIndex(i);

}

//LList.cpp end here

//Main start here

#include <iostream>

#include "HastT.h"

int main()

{

HashT M;

for (int i = 0; i < 45; i++)

{

M.add(i);

}

cout << "Before delete 13:";

cout << M.lookup(13) << endl;

M.erase(13);

cout << "After delete 13: " << M.lookup(13) << endl;

for (int i = 10; i < 30; i++)

{

cout << "Item " << i << " is: " << M.lookup(i) << endl;

}

cout << endl;

HashT H(M);

M.~HashT();

cout << "Destructor testing: " << endl;

for (int i = 10; i < 30; i++)

{

cout << "Item " << i << " is: " << M.lookup(i) << endl;

}

cout << endl;

cout << "Copy constructor testing: " << endl;

for (int i = 10; i < 30; i++)

{

cout << "Item " << i << " is: " << H.lookup(i) << endl;

}

}

//Main end here

Text

Description automatically generatedA picture containing text

Description automatically generated