Chapter 5 Projects #2, 4, 12 (b & e), 15 and 17.

//

// main.cpp

// Coen70HW3.1 Chapter 5 Problem 2

//

// Created by Yousef Zoumot on 2/1/16.

// Copyright (c) 2016 Yousef Zoumot. All rights reserved.

//

#include <iostream>

#include <assert.h>

#include <cstdlib>

using namespace std;

class Node{

public:

int \_data;

Node\* next;

Node();

};

class LinkedList{

int used;

Node\* head;

public:

LinkedList();

void add(int x);

void removeRepitition();

void printValues();

};

Node::Node(){

\_data=0;

next=NULL;

}

LinkedList::LinkedList(){

head=new Node();

used=0;

}

void LinkedList::removeRepitition(){

Node\* tmp=head;

Node\* tmp2=head->next;

Node\* tmp3=head;

Node\* tmp5;

while(tmp!=NULL){

tmp2=tmp->next;

tmp3=tmp;

while(tmp2!=NULL){

if(tmp->\_data==tmp2->\_data){

tmp3->next=tmp2->next;

tmp5=tmp2;

tmp2=tmp2->next;

delete tmp5;

used--;

tmp3=tmp3->next;

}

else{

tmp2=tmp2->next;

tmp3=tmp3->next;

}

}

tmp=tmp->next;

}

}

void LinkedList::add(int x){

Node\* tmp;

tmp=new Node();

tmp->next=head->next;

tmp->\_data=x;

head->next=tmp;

used++;

}

void LinkedList::printValues(){

int i=0;

Node\* tmp=head->next;

while(tmp!=NULL){

cout<<"\n"<<tmp->\_data;

tmp=tmp->next;

i++;

}

cout<<"\n";

}

int main(int argc, const char \* argv[]) {

LinkedList l1;

l1.add(5);

l1.add(4);

l1.add(3);

l1.add(2);

l1.add(1);

l1.add(2);

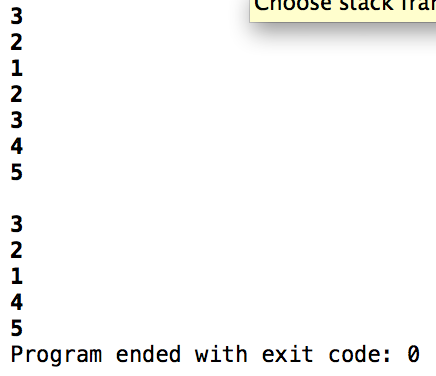
l1.add(3);

l1.printValues();

l1.removeRepitition();

l1.printValues();

return 0;

}

//

// main.cpp

// Coen70HW3.2 Chapter 5 Problem 4

//

// Created by Yousef Zoumot on 2/1/16.

// Copyright (c) 2016 Yousef Zoumot. All rights reserved.

//

#include <iostream>

#include <assert.h>

#include <cstdlib>

using namespace std;

class Node{

public:

int \_data;

Node\* \_next;

Node\* \_prev;

Node();

Node(int data);

};

class LinkedList{

int used;

Node\* head;

public:

LinkedList();

void add(int x);

void removeAll();

void removeRepitition();

void printValues();

void reverseOrder(Node\*& source);

Node\*& getHead();

};

Node::Node(){

\_data=0;

\_next=NULL;

\_prev=NULL;

}

Node::Node(int data){

this->\_data = data;

this->\_next = NULL;

this->\_prev = NULL;

}

void LinkedList::removeAll(){

Node\* tmp=head;

while(tmp->\_next!=NULL)

tmp=tmp->\_next;

Node\* tmp2;

while(used!=0 && tmp!=NULL){

tmp2=tmp->\_prev;

delete tmp;

used--;

tmp=tmp2;

}

used=0;

}

LinkedList::LinkedList(){

head=new Node();

used=0;

}

Node\*& LinkedList:: getHead(){

return head;

}

void LinkedList::reverseOrder(Node\*& source){

Node\* tmp=head;

LinkedList l1;

while(tmp!=NULL){

l1.add(tmp->\_data);

tmp=tmp->\_next;

}

while(used!=0)

removeAll();

head=l1.head;

used=l1.used-1;

}

void LinkedList::removeRepitition(){

Node\* tmp=head;

Node\* tmp2=head->\_next;

Node\* tmp3=head;

Node\* tmp5;

while(tmp!=NULL){

tmp2=tmp->\_next;

tmp3=tmp;

while(tmp2!=NULL){

if(tmp->\_data==tmp2->\_data){

tmp3->\_next=tmp2->\_next;

tmp5=tmp2;

tmp2=tmp2->\_next;

delete tmp5;

used--;

tmp3=tmp3->\_next;

}

else{

tmp2=tmp2->\_next;

tmp3=tmp3->\_next;

}

}

tmp=tmp->\_next;

}

}

void LinkedList::add(int x){

Node\* tmp=new Node(x);

head->\_prev=tmp;

tmp->\_next=head->\_next;

tmp->\_prev=NULL;

head->\_next=tmp;

used++;

}

void LinkedList::printValues(){

int i=0;

Node\* tmp=head->\_next;

while(i!=used){

cout<<"\n"<<tmp->\_data;

tmp=tmp->\_next;

i++;

}

cout<<"\n";

}

int main(int argc, const char \* argv[]) {

LinkedList l1;

l1.add(5);

l1.add(4);

l1.add(3);

l1.add(2);

l1.add(1);

l1.add(2);

l1.add(3);

l1.add(7);

l1.add(8);

l1.add(9);

l1.add(10);

l1.printValues();

l1.removeRepitition();

l1.printValues();

// Node tmp=l1.getHead();

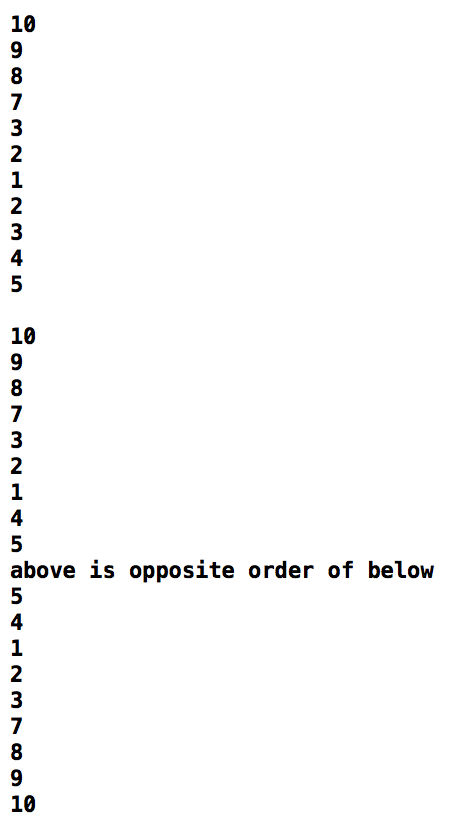
l1.reverseOrder(l1.getHead());

cout<<"above is opposite order of below";

l1.printValues();

return 0;

}



//

// main.cpp

// Coen70HW3.3 \*Chpater 5 Problem 12b

//

// Created by Yousef Zoumot on 2/2/16.

// Copyright (c) 2016 Yousef Zoumot. All rights reserved.

//

#include <iostream>

#include <stdlib.h>

#include <assert.h>

using namespace std;

class Set{

protected:

struct Node{

Node\* \_prev;

Node\* \_next;

int \_data;

Node(int data, Node\* prev = NULL, Node\* next = NULL){

this->\_data = data;

this->\_prev = prev;

this->\_next = next;

}

int& data(){return \_data;};

Node\*& next(){return \_next;};

Node\*& prev(){return \_prev;};

};

Node\* cursor;

int n;

public:

Set();

Set(const Set& source);

~Set();

void start();

void end();

void advance();

void reverse();

int size();

void insert(int data);

void attach(int data);

int current();

void remove();

void display();

void removeRepitition();

bool contains(const int& target);

Set& operator=(const Set& other);

friend ostream& operator<<(ostream &out, const Set &other);

};

bool Set:: contains(const int& target){//prints 1 if true 0 if false

Node\* tmp=cursor;

while(tmp->\_prev!=NULL){

tmp=tmp->\_prev;

}

while(tmp->\_next!=NULL){

if(tmp->\_data==target)

return true;

tmp=tmp->\_next;

}

return false;

}

void Set::removeRepitition(){

Node\* tmp=cursor;

while(tmp->\_prev!=NULL)

tmp=tmp->\_prev;

Node\* tmp2=tmp->\_next;

Node\* tmp3=tmp;

Node\* tmp5;

while(tmp!=NULL){

tmp2=tmp->\_next;

tmp3=tmp;

while(tmp2!=NULL){

if(tmp->\_data==tmp2->\_data){

tmp3->\_next=tmp2->\_next;

tmp5=tmp2;

tmp2=tmp2->\_next;

delete tmp5;

n--;

tmp3=tmp3->\_next;

}

else{

tmp2=tmp2->\_next;

tmp3=tmp3->\_next;

}

}

tmp=tmp->\_next;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Set::Set(){

cursor = NULL;

n = 0;

}

Set::~Set(){}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Set::Set(const Set& source){

operator=(source);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void Set::start(){

while(cursor->prev() != NULL){

cursor = cursor->prev();

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void Set::end(){

while(cursor->next() != NULL){

cursor = cursor->next();

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void Set::advance(){

if(cursor->next() != NULL){

cursor = cursor->next();

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void Set::reverse(){

if(cursor->prev() != NULL){

cursor = cursor->prev();

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int Set::size(){

return n;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int Set::current(){

assert(n != 0);

return cursor->data();

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

std::ostream& operator<<(std::ostream &out, const Set& other){

Set::Node\* tmp = other.cursor;

while(tmp != NULL){

out << tmp->data() << std::endl;

tmp = tmp->next();

}

return out;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Set& Set::operator=(const Set& other){

if(this != &other){

while(size() != 0){

remove();

}

Node\* tmp = other.cursor;

while(tmp->prev()!=NULL)

tmp=tmp->prev();

while(tmp != NULL){

insert(tmp->data());

tmp = tmp->next();

}

}

return \*this;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void Set::insert(int data){

if(n == 0){

cursor= new Node(data);

n++;

}

else{

Node\* tmp = new Node(data);

tmp->next() = cursor;

tmp->prev() = NULL;

cursor->prev() = tmp;

cursor = tmp;

n++;

}

removeRepitition();

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void Set::attach(int data){

if(n == 0){

cursor= new Node(data);

n++;

}

else{

Node\* tmp = new Node(data);

cursor->next() = tmp;

tmp->prev() = cursor;

tmp->next() = NULL;

cursor = tmp;

n++;

}

removeRepitition();

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void Set::remove(){

Node\* tmp = cursor;

while(tmp != NULL){

//if(tmp->data() == target){

if(cursor->next() == NULL){

cursor = cursor->prev();

//cursor->next();

delete tmp;

n--;

} else{

cursor->prev()->next() = cursor->next();

cursor->next()->prev() = cursor->prev();

delete tmp;

n--;

}

//}

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void Set::display(){

start();

Node\* tmp = cursor;

while(tmp != NULL){

std::cout << tmp->data() << "\n";

tmp = tmp->next();

}

std::cout << "The size of the Set is: " << n << endl;

}

int main(int argc, const char \* argv[]) {

Set s1;

s1.insert(5);

s1.insert(4);

s1.insert(3);

s1.insert(2);

s1.insert(1);

s1.insert(2);

s1.insert(3);

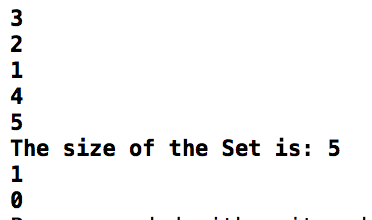
s1.display();

cout<<s1.contains(1)<<"\n";//prints 1 if true 0 if false

cout<<s1.contains(10)<<"\n";//prints 1 if true 0 if false

return 0;

}



//

// main.cpp

// Coen70HW3.4

//

// Created by Yousef Zoumot on 2/2/16.

// Copyright (c) 2016 Yousef Zoumot. All rights reserved.

//

#include <iostream>

#include <cassert>

#include <cstdlib> //provide size\_t

using namespace std;

class Keyed\_Bag

{

public:

//CONSTRUCTOR

Keyed\_Bag();

//MODIFICATION

bool erase\_one(const int& target);

void insert(const int& entry, int key);

void operator +=(const Keyed\_Bag& addend);

Keyed\_Bag operator -(const Keyed\_Bag& b);

void operator -=(const Keyed\_Bag& remove);

//CONSTANT MEMBER FUNCTIONS

int size() const { return used;}

int count(const int& target) const;

void printValues();

private:

struct Node{

Node\* \_prev;

Node\* \_next;

int \_data;

int \_key;

Node(int data, int key, Node\* prev = NULL, Node\* next = NULL){

this->\_data = data;

this->\_key=key;

this->\_prev = prev;

this->\_next = next;

}

int& data(){return \_data;};

Node\*& next(){return \_next;};

Node\*& prev(){return \_prev;};

};

Node\* head;

int used;

};

Keyed\_Bag::Keyed\_Bag(){

head=NULL;

used=0;

}

//NONMEMBER FUNCTIONS for the Keyed\_Bag class

Keyed\_Bag operator +(const Keyed\_Bag& b1, const Keyed\_Bag& b2);

bool Keyed\_Bag::erase\_one(const int& key1){

Node\* tmp=head;

while(tmp->\_next!=NULL && tmp->\_key != key1)

tmp=tmp->\_next;

if(tmp->\_next==NULL)

return false;

--used;

if(tmp->\_prev!=NULL)

tmp->\_prev->\_next=tmp->\_next;

if(tmp->\_prev==NULL)

head=tmp->\_next;

delete tmp;

return true;

}

void Keyed\_Bag::insert(const int& entry, int key){

Node\* tmp=new Node(entry, key);

Node\* dummy=head;

Node\* mummy=head;

if(head==NULL){

head=tmp;

return;

}

while(mummy!=NULL){

if(mummy->\_key==tmp->\_key)

return;

mummy=mummy->\_next;

}

while(dummy->\_next!=NULL){

dummy=dummy->\_next;

}

dummy->\_next=tmp;

tmp->\_prev=dummy;

tmp->\_next=NULL;

++used;

return;

}

void Keyed\_Bag :: printValues(){//a function that prints all the values in order to clean up the main function

Node\* tmp=head;

cout<<"\n";

while(tmp->\_next!=NULL){

cout<<"data: "<<tmp->\_data<<" with key: "<<tmp->\_key<<"\n";

tmp=tmp->\_next;

}

}

int main(int argc, const char \* argv[]) {

Keyed\_Bag b, b2;

b.insert(1,1);

b.insert(2,2);

b.insert(3,3);

b.insert(4,4);

b.insert(3,5);

b.insert(7,4);

b.insert(8,5);

b.insert(9,6);

b2.insert(3,6);

b2.insert(7,7);

b2.insert(2,2);

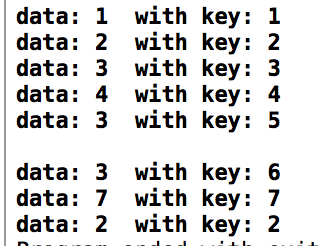
b2.insert(3,3);

b2.insert(3,7);

b.printValues();

b2.printValues();

return 0;

}

//

// main.cpp

// Coen70HW3.5

//

// Created by Yousef Zoumot on 2/3/16.

// Copyright (c) 2016 Yousef Zoumot. All rights reserved.

//

#include <iostream>

#include <cstdlib>

#include <cassert>

using namespace std;

class Sequence{

public:

Sequence();

//Sequence(const Sequence& source);

//~Sequence();

//Modification Member Functions

void start() {ptr\_cursor = ptr\_head;};

void end() {ptr\_cursor = ptr\_tail;};

void advance() {ptr\_cursor = ptr\_cursor->next();};

void retreat() {ptr\_cursor = ptr\_cursor->prev();};

void insert(const int& data);

//void attach(const int& data);

void remove\_current();

void addToFront(const int& data);

void addToEnd(const int& data);

//CONSTANT MEMBER FUNCTIONS

void printValues();

int size() const {return used;};

bool is\_item() const;

int current() const {return ptr\_cursor->data();};

private:

struct Node{

Node\* \_prev;

Node\* \_next;

int \_data;

Node(int data, Node\* prev = NULL, Node\* next = NULL){

this->\_data = data;

this->\_prev = prev;

this->\_next = next;

}

int& data(){return \_data;};

Node\*& next(){return \_next;};

Node\*& prev(){return \_prev;};

};

Node\* ptr\_head;

Node\* ptr\_tail;

Node\* ptr\_cursor;

int used; //How much of the array is used

};

////\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*////

Sequence::Sequence(){

ptr\_head=NULL;

ptr\_tail=NULL;

ptr\_cursor=NULL;

used=0;

}

////\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*////

void Sequence::addToFront(const int& data){//Does not move cursor

Node\* tmp=new Node(data);

if(used==0 && ptr\_head==NULL){

ptr\_head=tmp;

ptr\_tail=tmp;

ptr\_cursor=tmp;

used++;

return;

}

ptr\_head->prev()=tmp;

tmp->next()=ptr\_head;

ptr\_head=tmp;

tmp->prev()=NULL;

used++;

}

////\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*////

void Sequence::addToEnd(const int& data){//Does not move cursor

Node\* tmp=new Node(data);

if(used==0 && ptr\_tail==NULL){

ptr\_head=tmp;

ptr\_tail=tmp;

ptr\_cursor=tmp;

used++;

return;

}

ptr\_tail->next()=tmp;

tmp->prev()=ptr\_tail;

ptr\_tail=tmp;

tmp->next()=NULL;

used++;

}

////\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*////

void Sequence::insert(const int& data){//Does move cursor

Node\* tmp=new Node(data);

if(used==0 && ptr\_cursor==NULL){

ptr\_head=tmp;

ptr\_tail=tmp;

ptr\_cursor=tmp;

used++;

return;

}

if(ptr\_cursor->prev()==NULL){

ptr\_cursor->prev()=tmp;

tmp->next()=ptr\_cursor;

ptr\_cursor=tmp;

tmp->prev()=NULL;

ptr\_head=tmp;

used++;

}

else{

tmp->prev()=ptr\_cursor->prev();

ptr\_cursor->prev()=tmp;

tmp->next()=ptr\_cursor;

ptr\_cursor=tmp;

used++;

}

}

////\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*////

void Sequence::remove\_current(){//Moves cursor to next unless cursor is at the end

Node\* tmp;

if(used==0 && ptr\_cursor==NULL){

return;

}

if(ptr\_cursor->next()==NULL){

tmp=ptr\_cursor;

ptr\_cursor=ptr\_cursor->prev();

ptr\_cursor->next()=NULL;

ptr\_tail=ptr\_cursor;

delete tmp;

used--;

return;

}

if(ptr\_cursor->prev()==NULL){

tmp=ptr\_cursor;

ptr\_cursor=ptr\_cursor->next();

ptr\_cursor->prev()=NULL;

ptr\_head=ptr\_cursor;

delete tmp;

used--;

return;

}

else{

tmp=ptr\_cursor;

ptr\_cursor=ptr\_cursor->next();

ptr\_cursor->prev()=tmp->prev();

delete tmp;

used--;

return;

}

}

////\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*////

bool Sequence:: is\_item() const{

if(ptr\_cursor==NULL)

return false;

else{

return true;

}

}

////\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*////

void Sequence:: printValues(){

Node\* tmp=ptr\_head;

cout<<"The values are:"<<"\n";

while(tmp!=NULL){

cout<<tmp->data()<<"\n";

tmp=tmp->next();

}

cout<<"\n";

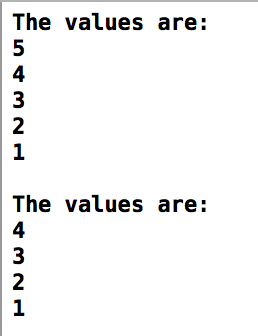
}

int main(int argc, const char \* argv[]) {

Sequence s1;

s1.insert(1);

s1.insert(2);

 s1.insert(3);

s1.insert(4);

s1.insert(5);

Sequence s2(s1);

s2.start();

s2.remove\_current();

s1.printValues();

s2.printValues();

return 0;

}

//

// main.cpp

// Coen70HW3.6 \*Chapter 5 #17

//

// Created by Yousef Zoumot on 2/3/16.

// Copyright (c) 2016 Yousef Zoumot. All rights reserved.

//

#include <algorithm>

#include <iostream>

#include <cassert>

using namespace std;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

class Node{

private:

int \_data;

int \_key;

Node\* \_next;

Node\* \_prev;

public:

Node(const int& = int(), Node\* = NULL);

int& data(){return \_data;}

Node\*& next(){return \_next;}

};

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

Node\* location(Node\* front\_ptr, size\_t position);

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

class bag{

public:

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

bag(){front = NULL; back= NULL; used = 0;}

bag(const bag& source);

~bag(){deleteList(front);}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

bool erase\_one(const int& target);

bool contains(const int& target);

void insert(const int& data);

int size() const { return used; }

int count( const int& target);

void printValues();

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

bag& operator =(const bag& source);

void operator -=(const bag& removeIt);

void operator +=(const bag& addend);

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

void list\_insert(Node\*& previous\_ptr, const int& data);

Node\* list\_search(Node\* front\_ptr, const int& target);

void insertAtFront(Node\*& front\_ptr, Node\*& back\_ptr, const int& data);

void list\_copy(Node\* source\_ptr, Node\*& front\_ptr, Node\*& back\_ptr);

void remove(Node\*& front\_ptr);

void removeNode(Node\*& previous\_ptr);

void deleteList(Node\*& front\_ptr);

int grab() const;

private:

Node\* front;

Node\* back;

int used;

void incSize();

};

bag operator +(const bag& b1, const bag& b2);

bag operator -(const bag& source1, const bag& source2);

int main(){

bag x;

bag y;

bag z;

x.insert(1);

x.insert(2);

x.insert(3);

x.insert(4);

x.insert(5);

y.insert(5);

y.insert(6);

y.insert(7);

x.printValues();

y.printValues();

x += y;

z = y;

x.printValues();

z.printValues();

x -= y;

x.printValues();

x.erase\_one(3);

x.erase\_one(4);

x.printValues();

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

Node:: Node(const int& data, Node\* next){

\_data = data;

\_next = next;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

bag& bag:: operator=(const bag& source){

if(this == &source)

return \*this;

deleteList(front);

used = 0;

if(source.used == 0){

used = 0;

front = NULL;

back = NULL;

return \*this;

}

Node\* temp = source.front;

insert(temp->data());

temp = temp->next();

while(temp != source.front){

insert(temp->data());

temp = temp->next();

}

used = source.used;

return \*this;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

bag:: bag(const bag& source){

Node\* back\_ptr;

list\_copy(source.front, front, back\_ptr);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

void bag::remove(Node\*& front\_ptr){

Node\* temp = front;

front\_ptr = front\_ptr->next();

delete temp;

used--;

return;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

void bag::list\_insert(Node\*& previous\_ptr, const int& data){

Node\* insert\_ptr = new Node;

insert\_ptr->data() = data;

insert\_ptr->next() = previous\_ptr->next();

previous\_ptr->next() = insert\_ptr;

previous\_ptr = insert\_ptr;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

void bag::insertAtFront(Node\*& front\_ptr, Node\*& back\_ptr, const int& data){

front\_ptr = new Node(data, front);

back\_ptr = front;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

void bag:: deleteList(Node\*& front\_ptr){

while(used != 0)

remove(front\_ptr);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

void bag:: removeNode(Node\*& previous\_ptr){

Node \*temp;

temp = previous\_ptr->next();

previous\_ptr->next() = temp->next();

delete temp;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

Node\* bag::list\_search(Node\* front\_ptr, const int& target){

Node\* cursor;

for(cursor = front\_ptr; cursor ->next() != NULL; cursor = cursor->next())

if(target == cursor->next()->data())

return cursor;

return NULL;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

Node\* location(Node\* front\_ptr, size\_t position){

assert(position>0);

Node\* cursor;

cursor = front\_ptr;

for(size\_t i = 1; (i < position) && (cursor != NULL); ++i)

cursor = cursor->next();

return cursor;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

void bag::list\_copy(Node\* source\_ptr, Node\*& front\_ptr, Node\*& back\_ptr){

front\_ptr = NULL;

back\_ptr = NULL;

if(source\_ptr == NULL)

return;

Node\* temp = source\_ptr;

insertAtFront(front\_ptr, back\_ptr, source\_ptr->data());

temp = temp -> next();

while(temp){

list\_insert(back\_ptr, temp->data());

temp = temp->next();

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

bool bag::erase\_one(const int& target){

Node\* cursor = front;

Node\* prev = back;

if (cursor == NULL)

return false;

if(cursor->data() == target){

prev->next() = cursor->next();

free(cursor);

front = prev->next();

used--;

return true;

}

else{

prev = cursor;

cursor = cursor->next();

while(cursor != front){

if(cursor->data() == target){

if(cursor==back){

back=prev;

}

if(cursor==front){

front=cursor->next();

}

prev->next() = cursor->next();

delete cursor;

used--;

return true;

}

else{

prev = cursor;

cursor = cursor->next();

}

}

return false;

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

void bag::insert(const int& data){

if(used == 0){

insertAtFront(front, back, data);

front -> next() = front;

}

else{

list\_insert(back, data);

}

used++;

return;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

void bag:: operator -=(const bag& removeIt){

Node\* cursor = removeIt.front;

erase\_one(cursor->data());

cursor = cursor->next();

while(cursor != removeIt.front){

erase\_one(cursor->data());

cursor = cursor->next();

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

void bag::operator +=(const bag& addend){

Node\* temp = addend.front;

insert(temp -> data());

temp = temp->next();

while(temp != addend.front){

insert(temp->data());

temp = temp->next();

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

int bag::count(const int& target) {

int answer;

Node\* cursor;

answer = 0;

cursor = list\_search(front, target);

while(cursor != NULL){

answer++;

cursor = cursor->next();

cursor = list\_search(cursor, target);

}

return answer;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

int bag::grab() const{

int i;

Node\* cursor;

assert(size() > 0);

i = (rand() % size()) + 1;

cursor = location(front, i);

return cursor->data();

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

bag operator -(const bag& source1, const bag& source2){

bag answer;

answer = source1;

answer -= source2;

return answer;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

bag operator +(const bag& b1, const bag& b2){

bag answer;

answer += b1;

answer += b2;

return answer;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

void bag::printValues(){

Node\* cursor = front;

while(cursor){

cout << cursor->data() << ", ";

cursor = cursor->next();

if(cursor == front){

break;

}

}

cout << endl;

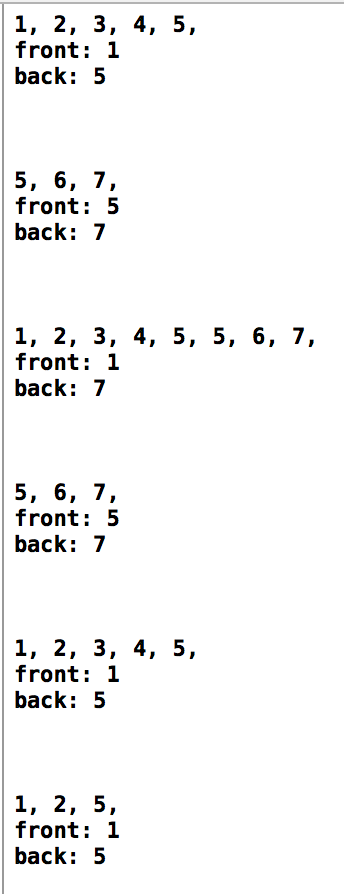
cout << "front: " << front->data() << endl;

cout << "back: " << back->data() << endl;

cursor = front;

cout << endl << endl << endl;

}

 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//\