// Yousef Zoumot

//

// Coen70HW1.1 \*Chapter 2 Problems 2 & 3

//

// Created by Yousef Zoumot on 1/10/16.

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

//max, min, mean, last, sum and length

#include <iostream>

using namespace std;

class statistician{

private:

double max, min, mean, last;

double sum =0;

double length=0;

public:

double getMax(){return max;};//returns max

double getMin(){return min;};//returns min

double getMean(){return mean;};//returns mean

double getLast(){return last;};//returns last

double getSum(){return sum;};//returns sum

double getLength(){return length;};//returns length

//instead of making variables public, I created functions to change the values

void changeMax(double n){max=n;};//changes max

void changeMin(double n){min=n;};//changes min

void changeMean(double n){mean=n;};//changes mean

void changeLast(double n){last=n;};//changes last

void changeSum(double n){sum=n;};//changes sum

void changeLength(double n){length=n;};//changes length

void next\_number(double n);

void newSequence();

void printValues();

};

statistician operator +(statistician s1, statistician s2){

statistician temp;

temp.changeSum( (s1.getSum() + s2.getSum()) );

temp.changeLength( (s1.getLength() + s2.getLength()) );

temp.changeMean( (temp.getSum()/temp.getLength()) );

temp.changeLast(s2.getLast());

if(s1.getMax() > s2.getMax())

temp.changeMax(s1.getMax());

else

temp.changeMax(s2.getMax());

if(s1.getMin() < s2.getMin())

temp.changeMin(s1.getMin());

else

temp.changeMin(s2.getMin());

return temp;

}

void statistician :: next\_number(double n){//this function takes in a double value and checks to see if length is zero. If so, it sets max and min to the parameter value. If not it compares the value to the max and min and replaces max and min if applicable. It then adds the value to the sum, increments length by 1, resets the mean value to the new mean, and places the value as the new last

if(length==0){

max=n;

min= n;

}

else{

if(n>max)

max=n;

if(n<min)

min=n;

}

sum= sum + n;

length++;

mean= sum/length;

last=n;

}

void statistician :: newSequence(){//only sets sum and length to zero b/c when the function next\_number is called, it checks if length is equal to zero, and if so then it resets the max and min variables as well as mean and last

sum=0;

length=0;

}

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

cout<<"\nThe max value is: " << getMax();

cout<<"\nThe min value is: " << getMin();

cout<<"\nThe mean value is: " << getMean();

cout<<"\nThe last value is : " << getLast();

cout<<"\nThe sum of all the values is: " << getSum();

cout<<"\nThe length value is: " << getLength()<<"\n";

}

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

statistician s1, s2, s3;

s1.next\_number(4);

s1.next\_number(5);

s1.next\_number(6);

s1.printValues();

//s1.newSequence();

s2.next\_number(1);

s2.next\_number(2);

s2.next\_number(3);

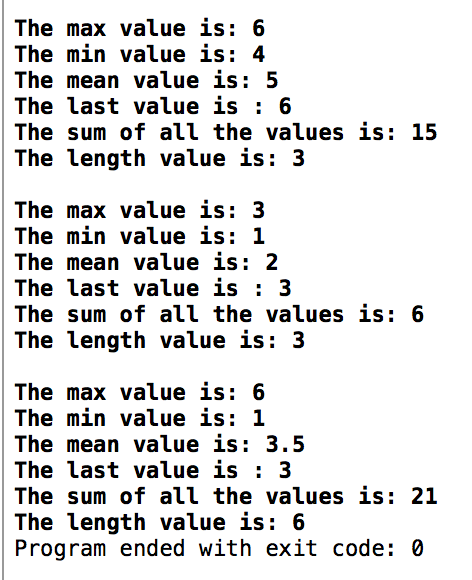
s2.printValues();

s3= (s1+ s2);

s3.printValues();

return 0;

}



// Yousef Zoumot

// main.cpp

// Coen70HW1.2 Chapter 2 Problem 5

//

// Created by Yousef Zoumot on 1/13/16.

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

//

#include <iostream>

#include <math.h>//

#include <iomanip>//

using namespace std;

#define PI 3.14159265

class position{

private:

double x, y, z;

public:

void setX(double i){x=i;};

void setY(double j){y=j;};

void setZ(double k){z=k;};

void shiftX(double i){x+=i;};

void shiftY(double j){y+=j;};

void shiftZ(double k){z+=k;};

void rotateAroundX(double theta);

void rotateAroundY(double theta);

void rotateAroundZ(double theta);

void printValues();

};

void position:: rotateAroundX(double theta){

double tempY=y;

double tempZ=z;

y= (tempY\*cos((theta\*PI)/180)) - (tempZ\*sin((theta\*PI)/180));

z= (tempY\*sin((theta\*PI)/180)) + (tempZ\*cos((theta\*PI)/180));

}

void position:: rotateAroundY(double theta){

double tempX=x;

double tempZ=z;

x=(tempX\*cos((theta\*PI)/180)) + (tempZ\*sin((theta\*PI)/180));

z=(-tempX\*sin((theta\*PI)/180)) + (tempZ\*cos((theta\*PI)/180));

}

void position:: rotateAroundZ(double theta){

double tempX=x;

double tempY=y;

x=(tempX\*cos((theta\*PI)/180)) - (tempY\*sin((theta\*PI)/180));

y=(tempX\*sin((theta\*PI)/180)) + (tempY\*cos((theta\*PI)/180));

}

void position:: printValues(){

cout<<"\nThe x value is: "<<std::fixed<<x;

cout<<"\nThe y value is: "<<std::fixed<<y;

cout<<"\nThe z value is: "<<std::fixed<<z<<"\n";

}

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

// insert code here...

position p;

p.setX(1);

p.setY(0.0);

p.setZ(0.0);

p.rotateAroundZ(90);

p.printValues();

p.shiftY(-1);

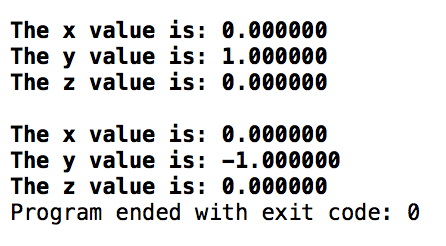
p.shiftZ(1);

p.rotateAroundX(90);

p.printValues();

return 0;

}



// Yousef Zoumot

// main.cpp

// Coen70HW1.3 \*Chapter 3 Problem 2

//

// Created by Yousef Zoumot on 1/12/16.

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

//

#include <iostream>

#include <cassert>

#include <cstdlib> //provide size\_t

using namespace std;

class bag

{

public:

//TYPEDEFS and MEMBER CONSTANTS

typedef int value\_type;

typedef std::size\_t size\_type;

static const size\_type CAPACITY=30;

//CONSTRUCTOR

bag() {used = 0;}

//MODIFICATION

size\_type erase (const value\_type& target);

bool erase\_one(const value\_type& target);

void insert(const value\_type&entry);

void operator +=(const bag& addend);

bag operator -(const bag& b);

void operator -=(const bag& remove);

//CONSTANT MEMBER FUNCTIONS

size\_type size() const { return used;}

size\_type count(const value\_type& target) const;

void printValues();

private:

value\_type data[CAPACITY]; //the array to store items

size\_type used; //How much of the array is used

};

//NONMEMBER FUNCTIONS for the bag class

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

const bag:: size\_type bag::CAPACITY;

bag::size\_type bag::erase(const value\_type& target){

size\_type index = 0;

size\_type many\_removed = 0;

while(index < used){

if (data[index] == target){

--used;

data[index] = data [used];

++many\_removed;

}

else

++index;

}

return many\_removed;

}

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

size\_type index;

index = 0;

while((index < used) && (data[index] != target))

++index;

if(index == used)

return false;

--used;

data[index] = data[used];

return true;

}

void bag::insert(const value\_type& entry){

assert(size() < CAPACITY);

data[used] = entry;

++used;

}

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

assert(size() + addend.size() <= CAPACITY);

copy(addend.data, addend.data + addend.used, data + used);

used += addend.used;

}

bag bag:: operator -(const bag& b){

bag temp = \*this;

for(bag::value\_type i=0; i< b.size(); i++)

temp.erase\_one(b.data[i]);

return temp;

}

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

for(bag::value\_type i=0; i< remove.size(); i++)

erase\_one(remove.data[i]);

}

bag::size\_type bag::count(const value\_type& target) const {

size\_type answer;

size\_type i;

answer = 0;

for(i = 0; i < used; ++i)

if (target == data[i])

++answer;

return answer;

}

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

bag answer;

assert(b1.size() + b2.size() <= bag::CAPACITY);

answer += b1;

answer += b2;

return answer;

}

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

size\_type index=0;

cout<<"\n";

while(size() > index){

cout<<data[index]<<"\n";

index++;

}

}

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

// insert code here...

bag b, b2;

b.insert(1);

b.insert(2);

b.insert(3);

b.insert(4);

b.insert(3);

b2.insert(3);

b2.insert(7);

b2.insert(2);

b2.insert(3);

b.printValues();

b2.printValues();

bag c;

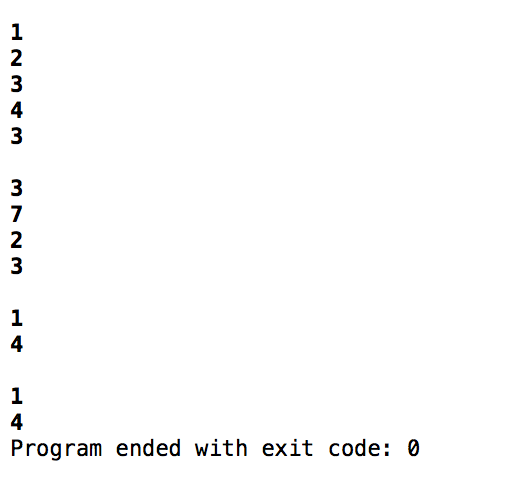
c=b-b2;

c.printValues();

b-=b2;

b.printValues();

return 0;

}

// Yousef Zoumot

// main.cpp

// Coen70HW1.4 \*Chapter 3 Problem 3

//

// Created by Yousef Zoumot on 1/13/16.

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

//

#include <iostream>

#include <assert.h>

#include <cstdlib>//Provides size\_t

using namespace std;

class sequence{

public:

//TYPEDEFS and MEMBER CONSTANTS

typedef double value\_type;

typedef std::size\_t size\_type;

static const size\_type CAPACITY=30;

//CONSTRUCTOR

sequence();

//MODIFICATION MEMBER FUNCTIONS

void start();

void advance();

void insert(const value\_type& entry);

void attach(const value\_type& entry);

void remove\_current();

void addToFront(const value\_type& entry);

void removeFront();

void addToEnd(const value\_type& entry);

void lastToCurrent();

sequence operator +(const sequence& s2);

void operator +=(const sequence& s2);

void printValues();

//CONSTANT MEMBER FUNCTIONS

size\_type size() const;

bool is\_item() const;

value\_type current() const;

private:

value\_type data[CAPACITY];

size\_type used;

size\_type current\_index;

};

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

// insert code here...

sequence s1, s2;

s1.addToEnd(1);

s1.addToEnd(2);

s1.addToEnd(3);

s1.addToEnd(4);

s1.addToEnd(5);

s2.addToEnd(6);

s2.addToEnd(7);

s2.addToEnd(8);

s2.addToEnd(9);

s1.printValues();

s2.printValues();

sequence s3;

s3= s1+s2;

s3.printValues();

sequence s4;

s4+=s1;

s4+=s2;

s4.printValues();

return 0;

}

// MODIFICATION MEMBER FUNCTIONS

sequence::sequence ()

{

current\_index = 0;

used = 0;

}

void sequence::start( )

{

current\_index = 0;

}

void sequence::advance( )

{

current\_index++;

}

void sequence::insert(const value\_type& entry)

{

if(current\_index==used){

data[current\_index]=entry;

used++;

return;

}

size\_type i;

for (i = used; i > current\_index; i--)

data[i]= data[i-1];

data[current\_index] = entry;

used++;

}

void sequence::attach(const value\_type& entry)

{

if(!is\_item()){

data[current\_index]=entry;

used++;

return;

}

size\_type i;

for (i = used; i > current\_index+1; i--)

data[i] = data[i+1];

data[current\_index+1] = entry;

current\_index++;

used++;

}

void sequence::remove\_current( )

{

size\_type i;

for (i= current\_index; i < used-1; i++)

data[i] = data[i+1];

used--;

}

void sequence:: addToFront(const value\_type& entry){

if(current\_index==used){

data[current\_index]=entry;

used++;

return;

}

size\_type i;

for (i = used; i > 0; i--)

data[i]= data[i-1];

data[0] = entry;

start();

used++;

}

void sequence:: removeFront(){

start();

remove\_current();

}

void sequence:: addToEnd(const value\_type& entry){

current\_index=used;

data[current\_index]=entry;

used++;

}

void sequence:: lastToCurrent(){

data[current\_index]=data[used-1];

used--;

}

sequence sequence:: operator +(const sequence& s2){

sequence temp;

size\_type i=0;

size\_type f=0;

while(temp.size() < size()){

temp.data[i]=data[i];

i++;

temp.used++;

}

while (temp.size() < (size()+s2.size())) {

temp.data[i]=s2.data[f];

f++;

i++;

temp.used++;

}

return temp;

}

void sequence:: operator +=(const sequence& s2){

\*this=\*this+s2;

/\* //This code is not needed since I already overloaded the + operator, I can just call the plus operator in this overloaded operator

size\_type i=0;

while(this->size() < (this->size()+ s2.size())){

cout<<i;

data[used]=s2.data[i];

i++;

used++;

}\*/

}

void sequence:: printValues(){

cout<<"The values in the sequence are as follows: "<<"\n";

size\_type i;

for(i=0; i<size(); i++)

cout<<data[i]<<" \n";

}

// CONSTANT MEMBER FUNCTIONS

sequence::size\_type sequence::size( ) const

{

return used;

}

bool sequence::is\_item( ) const

{

return current\_index != used;

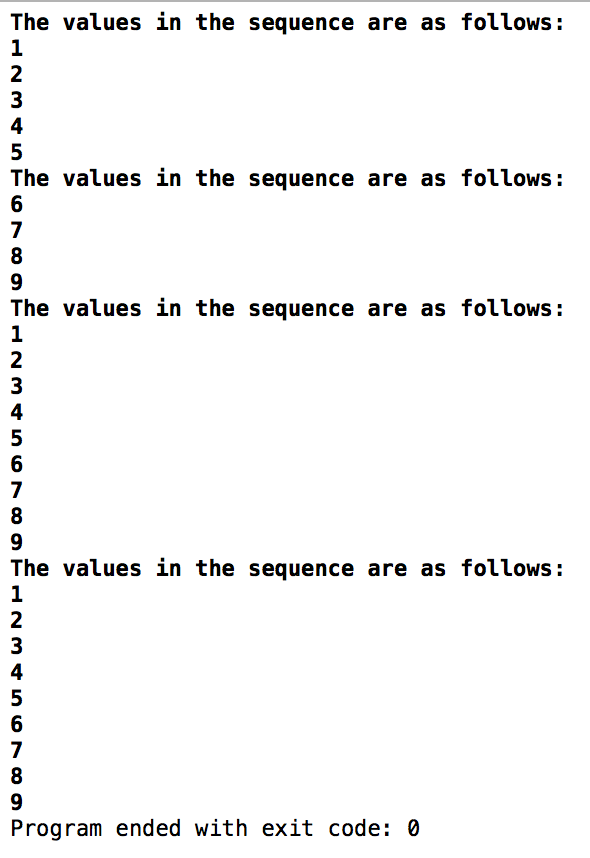
}

sequence::value\_type sequence::current( ) const

{

return data[current\_index];

}



// Yousef Zoumot

// main.cpp

// Coen70HW2.1 \*Chapter 3 Problem 4

//

// Created by Yousef Zoumot on 1/18/16.

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

//

#include <iostream>

#include <assert.h>

#include <cstdlib>//Provides size\_t

using namespace std;

class sequence{

public:

//TYPEDEFS and MEMBER CONSTANTS

typedef double value\_type;

typedef std::size\_t size\_type;

static const size\_type CAPACITY=30;

//CONSTRUCTOR

sequence();

//MODIFICATION MEMBER FUNCTIONS

void start();

void advance();

void insert(const value\_type& entry);

void attach(const value\_type& entry);

void remove\_current();

void addToFront(const value\_type& entry);

void removeFront();

void addToEnd(const value\_type& entry);

void lastToCurrent();

sequence operator +(const sequence& s2);

void operator +=(const sequence& s2);

value\_type operator[](size\_type index);

void printValues();

//CONSTANT MEMBER FUNCTIONS

size\_type size() const;

bool is\_item() const;

value\_type current() const;

private:

value\_type data[CAPACITY];

size\_type used;

size\_type current\_index;

};

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

// insert code here...

sequence s1, s2;

s1.addToEnd(1);

s1.addToEnd(2);

s1.addToEnd(3);

s1.addToEnd(4);

s1.addToEnd(5);

s2.addToEnd(6);

s2.addToEnd(7);

s2.addToEnd(8);

s2.addToEnd(9);

s1.printValues();

s2.printValues();

sequence s3;

s3= s1+s2;

s3.printValues();

sequence s4;

s4+=s1;

s4+=s2;

s4.printValues();

cout<<s4[0];

return 0;

}

// MODIFICATION MEMBER FUNCTIONS

sequence::sequence ()

{

current\_index = 0;

used = 0;

}

void sequence::start( )

{

current\_index = 0;

}

void sequence::advance( )

{

current\_index++;

}

void sequence::insert(const value\_type& entry)

{

if(current\_index==used){

data[current\_index]=entry;

used++;

return;

}

size\_type i;

for (i = used; i > current\_index; i--)

data[i]= data[i-1];

data[current\_index] = entry;

used++;

}

void sequence::attach(const value\_type& entry)

{

if(!is\_item()){

data[current\_index]=entry;

used++;

return;

}

size\_type i;

for (i = used; i > current\_index+1; i--)

data[i] = data[i+1];

data[current\_index+1] = entry;

current\_index++;

used++;

}

void sequence::remove\_current( )

{

size\_type i;

for (i= current\_index; i < used-1; i++)

data[i] = data[i+1];

used--;

}

void sequence:: addToFront(const value\_type& entry){

if(current\_index==used){

data[current\_index]=entry;

used++;

return;

}

size\_type i;

for (i = used; i > 0; i--)

data[i]= data[i-1];

data[0] = entry;

start();

used++;

}

void sequence:: removeFront(){

start();

remove\_current();

}

void sequence:: addToEnd(const value\_type& entry){

current\_index=used;

data[current\_index]=entry;

used++;

}

void sequence:: lastToCurrent(){

data[current\_index]=data[used-1];

used--;

}

double sequence:: operator[](size\_type index){

value\_type invalid=100000;

if(index<size())

return data[index];

else{

cout<<"This is not a valid index";

return invalid;

};

}

sequence sequence:: operator +(const sequence& s2){

sequence temp;

size\_type i=0;

size\_type f=0;

while(temp.size() < size()){

temp.data[i]=data[i];

i++;

temp.used++;

}

while (temp.size() < (size()+s2.size())) {

temp.data[i]=s2.data[f];

f++;

i++;

temp.used++;

}

return temp;

}

void sequence:: operator +=(const sequence& s2){

\*this=\*this+s2;

}

void sequence:: printValues(){

cout<<"The values in the sequence are as follows: "<<"\n";

size\_type i;

for(i=0; i<size(); i++)

cout<<data[i]<<" \n";

}

// CONSTANT MEMBER FUNCTIONS

sequence::size\_type sequence::size( ) const

{

return used;

}

bool sequence::is\_item( ) const

{

return current\_index != used;

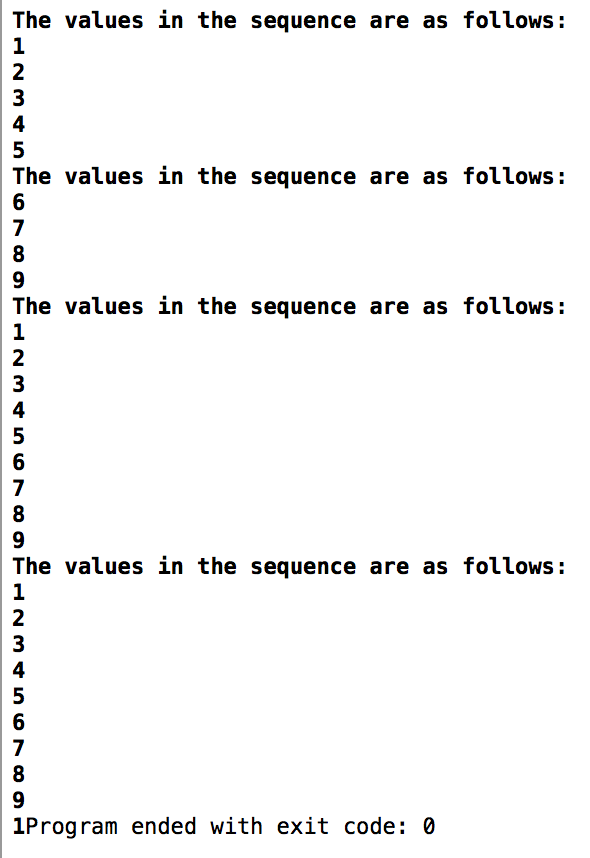
}

sequence::value\_type sequence::current( ) const

{

return data[current\_index];

}



// Yousef Zoumot

// main.cpp

// Coen70HW2.2 \*Chapter 3 Problem 5

//

// Created by Yousef Zoumot on 1/12/16.

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

//

#include <iostream>

#include <cassert>

#include <cstdlib> //provide size\_t

using namespace std;

class set

{

public:

//TYPEDEFS and MEMBER CONSTANTS

typedef int value\_type;

typedef std::size\_t size\_type;

static const size\_type CAPACITY=30;

//CONSTRUCTOR

set() {used = 0;}

//MODIFICATION

size\_type erase (const value\_type& target);

bool erase\_one(const value\_type& target);

void insert(const value\_type&entry);

void operator +=(const set& addend);

set operator -(const set& b);

void operator -=(const set& remove);

//CONSTANT MEMBER FUNCTIONS

size\_type size() const { return used;}

size\_type count(const value\_type& target) const;

void printValues();

bool contains(const value\_type& target);

set operator +(const set& b2);

private:

value\_type data[CAPACITY]; //the array to store items

size\_type used; // much of the array is used

};

//NONMEMBER FUNCTIONS for the set class

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

const set:: size\_type set::CAPACITY;

bool set:: contains(const value\_type& target){

for(size\_type i=0; i<size(); i++){

if(data[i]== target)

return true;

}

return false;

}

set::size\_type set::erase(const value\_type& target){

size\_type index = 0;

size\_type many\_removed = 0;

while(index < used){

if (data[index] == target){

--used;

data[index] = data [used];

++many\_removed;

}

else

++index;

}

return many\_removed;

}

bool set::erase\_one(const value\_type& target){

size\_type index;

index = 0;

while((index < used) && (data[index] != target))

++index;

if(index == used)

return false;

--used;

data[index] = data[used];

return true;

}

void set::insert(const value\_type& entry){

assert(size() < CAPACITY);

if(contains(entry)){

return;

}

data[used] = entry;

++used;

return;

}

set set:: operator -(const set& b){

set temp = \*this;

for(set::value\_type i=0; i< b.size(); i++)

temp.erase\_one(b.data[i]);

return temp;

}

void set:: operator -=(const set& remove){

for(set::value\_type i=0; i< remove.size(); i++)

erase\_one(remove.data[i]);

}

set::size\_type set::count(const value\_type& target) const {

size\_type answer;

size\_type i;

answer = 0;

for(i = 0; i < used; ++i)

if (target == data[i])

++answer;

return answer;

}

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

assert(size() + addend.size() <= CAPACITY);

for(int i=0; i<addend.size(); i++){

if(!contains(addend.data[i])){

data[used]=addend.data[i];

used++;

}

}

}

set set:: operator +(const set& b2){

set answer;

assert(size() + b2.size() <= CAPACITY);

answer=\*this;

answer+=b2;

return answer;

}

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

size\_type index=0;

cout<<"\n";

while(size() > index){

cout<<data[index]<<"\n";

index++;

}

}

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

// insert code here...

set b, b2;

b.insert(1);

b.insert(2);

b.insert(3);

b.insert(4);

b.insert(3);

b2.insert(4);

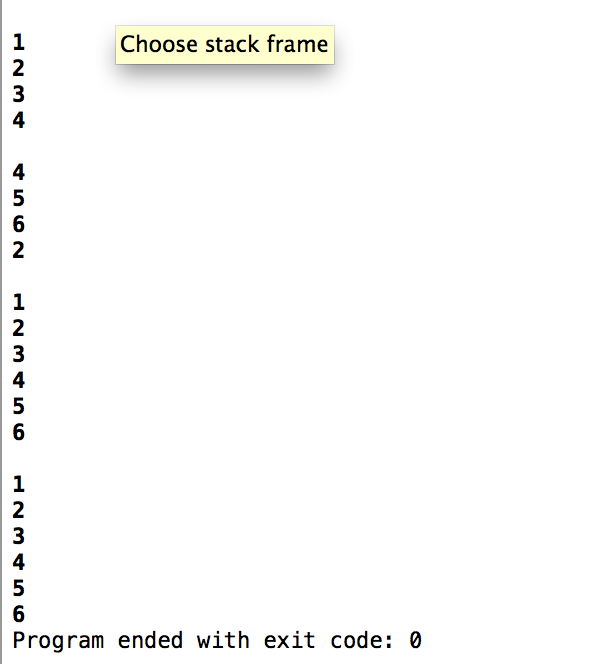
b2.insert(5);

b2.insert(6);

b2.insert(2);

b.printValues();

b2.printValues();



set c;

c=b+b2;

c.printValues();

b+=b2;

b.printValues();

return 0;

}

// Yousef Zoumot

// main.cpp

// Coen70HW2.3 \*Chapter 3 Problem 8

//

// Created by Yousef Zoumot on 1/12/16.

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

//

#include <iostream>

#include <cassert>

#include <cstdlib> //provide size\_t

using namespace std;

class bag

{

public:

//TYPEDEFS and MEMBER CONSTANTS

typedef int value\_type;

typedef std::size\_t size\_type;

static const size\_type CAPACITY=30;

//CONSTRUCTOR

bag() {used = 0;}

//MODIFICATION

size\_type erase (const value\_type& target);

bool erase\_one(const value\_type& target);

void insert(const value\_type& entry, int key);

void operator +=(const bag& addend);

bag operator -(const bag& b);

void operator -=(const bag& remove);

//CONSTANT MEMBER FUNCTIONS

size\_type size() const { return used;}

size\_type count(const value\_type& target) const;

void printValues();

private:

value\_type data[CAPACITY];//the array to store items

int keys[CAPACITY];

size\_type used; //How much of the array is used

};

//NONMEMBER FUNCTIONS for the bag class

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

const bag:: size\_type bag::CAPACITY;

bag::size\_type bag::erase(const value\_type& target){

size\_type index = 0;

size\_type many\_removed = 0;

while(index < used){

if (data[index] == target){

--used;

data[index] = data [used];

++many\_removed;

}

else

++index;

}

return many\_removed;

}

bool bag::erase\_one(const value\_type& key1){

size\_type index;

index = 0;

while((index < used) && (keys[index] != key1))

++index;

if(index == used)

return false;

--used;

data[index] = data[used];

keys[index]=keys[used];

return true;

}

void bag::insert(const value\_type& entry, int key){

assert(size() < CAPACITY);

for(int i=0; i<size(); i++){

if(keys[i]==key)

return;

}

data[used] = entry;

keys[used]=key;

++used;

return;

}

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

assert(size() + addend.size() <= CAPACITY);

bool tmp=false;

for(int i=0; i<addend.size(); i++){

for(int j=0; j<size(); j++){

if(addend.keys[i]==keys[j])

tmp=true;

}

if(tmp==false){

data[used]=addend.data[i];

keys[used]=addend.keys[i];

used++;

}

tmp=false;

}

//copy(addend.data, addend.data + addend.used, data + used);

//copy(addend.keys, addend.keys + addend.used, keys + used);

//used += addend.used;

}

bag bag:: operator -(const bag& b){

bag temp = \*this;

for(bag::value\_type i=0; i< b.size(); i++)

temp.erase\_one(b.keys[i]);

return temp;

}

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

for(bag::value\_type i=0; i< remove.size(); i++)

erase\_one(remove.data[i]);

}

bag::size\_type bag::count(const value\_type& target) const {

size\_type answer;

size\_type i;

answer = 0;

for(i = 0; i < used; ++i)

if (target == data[i])

++answer;

return answer;

}

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

bag answer;

assert(b1.size() + b2.size() <= bag::CAPACITY);

answer += b1;

answer += b2;

return answer;

}

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

size\_type index=0;

cout<<"\n";

while(size() > index){

cout<<data[index]<<" with index: "<<keys[index]<<"\n";

index++;

}

}

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

// insert code here...

bag b, b2;

b.insert(1,1);

b.insert(2,2);

b.insert(3,3);

b.insert(4,4);

b.insert(3,5);

b2.insert(3,6);

b2.insert(7,7);

b2.insert(2,2);

b2.insert(3,3);

b.printValues();

b2.printValues();

bag c;

c=b-b2;

c.printValues();

b-=b2;

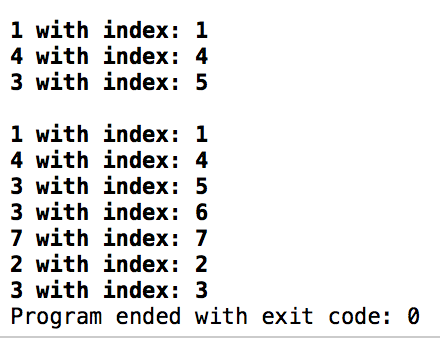
b.printValues();

b+=b2;

b.printValues();

return 0;

}



// Yousef Zoumot

// main.cpp

// Coen70HW2.4 \*Chapter 4 Problem 1 parts A, D, F, G b/c professor said we only need to choose 4 to do

//

// Created by Yousef Zoumot on 1/18/16.

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

//

#include <iostream>

#include <assert.h>

#include <cstdlib>//Provides size\_t

using namespace std;

class String{

char myString[200];

size\_t length;

public:

//CONSTRUCTOR FOR THE STRING CLASS

String(const char str[] = "");

String(const char c);//part A

//CONSTANT MEMBER FUNCTIONS FOR THE STRING CLASS

int searchF(const char c);//Part F returns index of first occurrence or returns -1

int searchA(const char c);//Part G returns number of occurrences or returns 0

//MODIFICATION MEMBER FUNCTIONS FOR THE STRING CLASS

void replaceChar(const char c, int index);//part D

//Tester Function

void printValues();

};

String:: String(const char str[]){

int i=0;

if(str[i]=='\0'){

myString[i]='\0';

length++;

return;

}

while(str[i]!= '\0'){

myString[i]=str[i];

i++;

length++;

}

return;

}

String::String(const char c){

if(c=='\0'){

myString[0]=c;

length++;

return;

}

myString[0]=c;

length++;

myString[1]='\0';

length++;

return;

}

void String:: replaceChar(const char c, int index){

myString[index]=c;

return;

}

int String:: searchF(const char c){

int i=0;

while(myString[i]!='\0'){

if(myString[i]==c)

return i;

i++;

}

cout<<"Not Found ";

return -1;

}

int String:: searchA(const char c){

int i=0;

int answer=0;

while(myString[i]!='\0'){

if(myString[i]==c){

answer++;

}

i++;

}

return answer;

}

void String:: printValues(){

int i=0;

while(myString[i]!='\0'){

cout<<myString[i];

i++;

}

cout<<"\n";

}

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

String s1("Hello my name is Johnny");

s1.printValues();

char c='A';

String s2(c);

s2.printValues();

s2.replaceChar('B', 0);

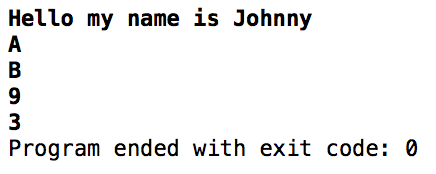
s2.printValues();

c='n';

cout<<s1.searchF(c)<<"\n";

cout<<s1.searchA(c)<<"\n";

}



// Yousef Zoumot

// main.cpp

// Coen70HW2.5 \*Chapter 4 Problem 2A

//

// Created by Yousef Zoumot on 1/18/16.

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

//

#include <iostream>

#include <assert.h>

#include<vector>

#include <cstdlib>//Provides size\_t

using namespace std;

class sequence{

public:

//TYPEDEFS and MEMBER CONSTANTS

typedef double value\_type;

typedef std::size\_t size\_type;

//CONSTRUCTOR

sequence(const size\_t cap);

//MODIFICATION MEMBER FUNCTIONS

void start();

void advance();

void insert(const value\_type& entry);

void attach(const value\_type& entry);

void remove\_current();

void addToFront(const value\_type& entry);

void removeFront();

void addToEnd(const value\_type& entry);

void lastToCurrent();

sequence operator +(const sequence& s2);

void operator +=(const sequence& s2);

value\_type operator[](size\_type index);

void printValues();

//CONSTANT MEMBER FUNCTIONS

size\_type size() const;

bool is\_item() const;

value\_type current() const;

private:

value\_type\* data;

size\_t capacity;

size\_type used;

size\_type current\_index;

void increaseS();

};

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

// insert code here...

sequence s1(100), s2(100);

s1.addToEnd(1);

s1.addToEnd(2);

s1.addToEnd(3);

s1.addToEnd(4);

s1.addToEnd(5);

s2.addToEnd(6);

s2.addToEnd(7);

s2.addToEnd(8);

s2.addToEnd(9);

s1.printValues();

s2.printValues();

sequence s3(100);

s3= s1+s2;

s3.printValues();

sequence s4(100);

s4+=s1;

s4+=s2;

s4.printValues();

cout<<s4[0];

return 0;

}

// MODIFICATION MEMBER FUNCTIONS

sequence::sequence (const size\_t cap)

{

current\_index = 0;

used = 0;

capacity=cap;

data= new value\_type[capacity];

}

void sequence::start( )

{

current\_index = 0;

}

void sequence::advance( )

{

current\_index++;

}

void sequence::insert(const value\_type& entry)

{

if(current\_index==used){

data[current\_index]=entry;

used++;

return;

}

size\_type i;

for (i = used; i > current\_index; i--)

data[i]= data[i-1];

data[current\_index] = entry;

used++;

}

void sequence::attach(const value\_type& entry)

{

if(!is\_item()){

data[current\_index]=entry;

used++;

return;

}

size\_type i;

for (i = used; i > current\_index+1; i--)

data[i] = data[i+1];

data[current\_index+1] = entry;

current\_index++;

used++;

}

void sequence::remove\_current( )

{

size\_type i;

for (i= current\_index; i < used-1; i++)

data[i] = data[i+1];

used--;

}

void sequence:: addToFront(const value\_type& entry){

if(current\_index==used){

data[current\_index]=entry;

used++;

return;

}

size\_type i;

for (i = used; i > 0; i--)

data[i]= data[i-1];

data[0] = entry;

start();

used++;

}

void sequence:: removeFront(){

start();

remove\_current();

}

void sequence:: addToEnd(const value\_type& entry){

current\_index=used;

data[current\_index]=entry;

used++;

}

void sequence:: lastToCurrent(){

data[current\_index]=data[used-1];

used--;

}

double sequence:: operator[](size\_type index){

value\_type invalid=100000;

if(index<size())

return data[index];

else{

cout<<"This is not a valid index";

return invalid;

};

}

sequence sequence:: operator +(const sequence& s2){

sequence temp(100);

size\_type i=0;

size\_type f=0;

while(temp.size() < size()){

temp.data[i]=data[i];

i++;

temp.used++;

}

while (temp.size() < (size()+s2.size())) {

temp.data[i]=s2.data[f];

f++;

i++;

temp.used++;

}

return temp;

}

void sequence:: operator +=(const sequence& s2){

\*this=\*this+s2;

}

void sequence:: printValues(){

cout<<"The values in the sequence are as follows: "<<"\n";

size\_type i;

for(i=0; i<size(); i++)

cout<<data[i]<<" \n";

}

void sequence:: increaseS(){

value\_type\* tmp= new value\_type[2\*capacity];

for(int i=0; i<size(); i++){

tmp[i]=data[i];

}

delete[] data;

data=tmp;

capacity\*=2;

}

// CONSTANT MEMBER FUNCTIONS

sequence::size\_type sequence::size( ) const

{

return used;

}

bool sequence::is\_item( ) const

{

return current\_index != used;

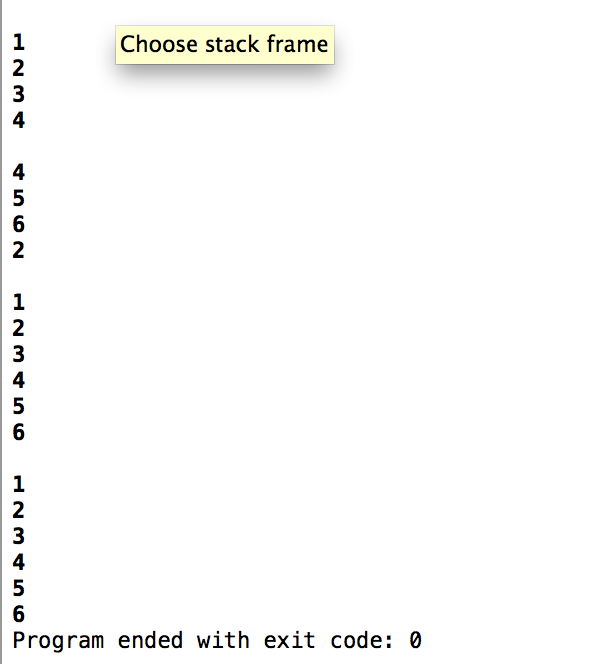
}

sequence::value\_type sequence::current( ) const

{

return data[current\_index];

}



// Yousef Zoumot

// main.cpp

// Coen70HW2.6 \*Chapter 4 Problem 2b

//

// Created by Yousef Zoumot on 1/23/16.

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

//

#include <algorithm>

#include <iostream>

#include <cassert>

using namespace std;

class set{

int\* data;

int capacity;

void incSize();

int used;

public:

set(int x = 20);

set(const set& source);

~set();

int erase(const int& target);

bool erase\_one(const int& target);

void insert(const int& target);

set operator -(const set& b2);

set& operator =(const set& source);

void operator -=(const set& removeIt);

void operator +=(const set& addend);

set operator +(const set& b2);

bool contains(const int& target) const;

int size() const { return used; }

int count( const int& target) const;

void printValues();

};

int main(){

set a;

set b;

set c;

set d;

a.insert(1);

a.insert(2);

a.insert(2);

a.insert(3);

a.printValues();

b.insert(3);

b.insert(2);

b.insert(5);

b.printValues();

c = a - b;

c.printValues();

c = a + b;

c.printValues();

d.insert(3);

c += d;

c.printValues();

c -= d;

c.printValues();

c.erase\_one(1);

c.printValues();

}

int set::erase(const int& target){

int index = 0;

int many\_removed = 0;

while(index < used){

if (data[index] == target){

--used;

data[index] = data [used];

++many\_removed;

}

else

++index;

}

return many\_removed;

}

set:: set (int x){

assert(x>0);

used = 0;

capacity = x;

data = new int[x];

}

set:: ~set(){

if (data)

delete[] data;

}

set:: set(const set& source){

data = NULL;

\*this = source;

}

void set:: incSize(){

int\* temp = new int[2\*capacity];

for(int i = 0; i < capacity; i++){

temp[i] = data[i];

}

delete[] data;

data = temp;

capacity \*= 2;

}

void set::printValues(){

int i;

for(i = 0; i < used; i++){

cout << data[i] << " ";

}

cout << endl;

}

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

int index;

index = 0;

while((index < used) && (data[index] != target))

++index;

if(index == used)

return false;

--used;

data[index] = data[used];

return true;

}

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

int i;

if(size() + addend.size() >= capacity)

incSize();

for(i = 0; i < addend.used; i++){

if(!contains(addend.data[i])){

data[used] = addend.data[i];

used++;

}

}

}

set set:: operator -(const set& b2){

set answer = \*this;

for(int i = 0; i < b2.used; i++)

answer.erase\_one(b2.data[i]);

return answer;

}

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

int answer;

int i;

answer = 0;

for(i = 0; i < used; ++i)

if (target == data[i])

++answer;

return answer;

}

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

int i;

for(i = 0; i < removeIt.used; i++)

erase\_one(removeIt.data[i]);

}

void set::insert(const int& entry){

if(contains(entry))

return;

if(size() >= capacity)

incSize();

data[used] = entry;

++used;

return;

}

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

if(this == &source)

return \*this;

if (data)

delete[] data;

if(source.used == 0){

used = 0;

capacity = 20;

data = new int[capacity];

return \*this;

}

data = new int[source.capacity];

for(int i = 0; i < source.capacity; i++){

data[i] = source.data[i];

}

used = source.used;

capacity = source.capacity;

return \*this;

}

bool set:: contains(const int& target) const{

int i;

for(i = 0; i < used; ++i)

if (target == data[i])

return true;

return false;

}

set set::operator +(const set& b2){

set answer = \*this;

if(answer.size() + b2.size() >= capacity)

incSize();

for(int i = 0; i < b2.used; i++){

if(!answer.contains(b2.data[i])){

answer.data[used] = b2.data[i];

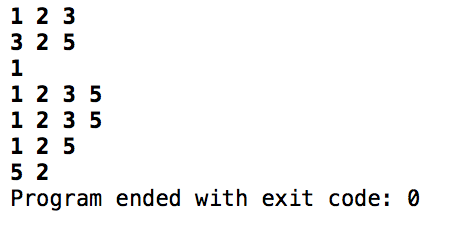
answer.used++;

}

}

return answer;

}



// Yousef Zoumot

// main.cpp

// Coen70HW2.7 \*Chapter 4 Problem 2e

//

// Created by Yousef Zoumot on 1/12/16.

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

//

#include <algorithm>

#include <iostream>

#include <cassert>

using namespace std;

class bag{

public:

bag(int x = 20);

bag(const bag& source);

~bag();

int erase(const int& target);

bool erase\_one(const int& target);

bool contains(const int& target);

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

int size() const { return used; }

int count( const int& target) const;

void printValues();

bag operator +(const bag& b2);

bag& operator =(const bag& source);

bag operator -(const bag& b2);

void operator -=(const bag& removeIt);

void operator +=(const bag& addend);

private:

int\*\* data;

int capacity;

int used;

void incSize();

};

int main(){

bag a;

bag b;

bag c;

bag d;

a.insert(1, 1);

a.insert(2, 2);

a.insert(3, 3);

a.insert(4, 4);

b.insert(5, 5);

b.insert(6, 6);

b.insert(7, 7);

a.printValues();

b.printValues();

c = a + b;

c.printValues();

c = a - b;

c.printValues();

d.insert(1,6);

d.insert(2, 2);

d.insert(3, 5);

d.printValues();

c -= d;

c.printValues();

d.erase\_one(6);

d.erase\_one(5);

d.printValues();

}

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

int index = 0;

int many\_removed = 0;

while(index < used){

if (data[index][0] == target){

--used;

data[index][0] = data[used][0];

data[index][1] = data[used][1];

++many\_removed;

}

else

++index;

}

return many\_removed;

}

bag:: bag (int x){

assert(x>0);

used = 0;

capacity = x;

data = new int\*[x];

for(int i = 0; i < x; i++){

data[i] = new int[2];

}

}

bag:: bag(const bag& source){

data = NULL;

\*this = source;

}

bag:: ~bag(){

for(int i = 0; i < capacity; i++){

delete[] data[i];

}

delete[] data;

}

void bag::printValues(){

int i;

for(i = 0; i < used; i++){

cout <<data[i][0]<<" " ;

}

cout << endl;

for(i = 0; i < used; i++){

cout << data[i][1] << " ";

}

cout << endl << endl << endl;

}

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

int index;

index = 0;

while((index < used) && (data[index][1] != target))

++index;

if(index == used)

return false;

--used;

data[index][1] = data[used][1];

data[index][0] = data[used][0];

return true;

}

void bag:: incSize(){

int\*\* temp = new int\* [2\*capacity];

for(int i = 0; i < 2\*capacity; i++){

temp[i][0] = data[i][0];

temp[i][1] = data[i][1];

}

for(int i = 0; i < capacity; i++){

temp[i][0] = data[i][0];

temp[i][1] = data[i][1];

}

for(int i = 0; i < capacity; i++){

delete[] data[i];

}

delete[] data;

data = temp;

capacity \*= 2;

}

void bag::insert(const int& entry, int key1){

if(size() == capacity)

incSize();

data[used][0] = entry;

for(int i = 0; i < used; i++){

if(data[i][1] == key1){

cout << "That key is already used. Enter another";

cin >> key1;

i = 0;

}

}

data[used][1] = key1;

++used;

return;

}

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

\*this = \*this + addend;

}

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

/\*bag answer;

answer.data = NULL;

answer = \*this;\*/

bag answer;

if(answer.data){

for(int i = 0; i < answer.capacity; i++){

delete[] answer.data[i];

}

delete[] answer.data;

}

answer.data = new int\*[capacity];

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

answer.data[i] = new int[2];

for(int i = 0; i < source.capacity; i++){

answer.data[i][0] = data[i][0];

answer.data[i][1] = data[i][1];

}

answer.capacity = capacity;

answer.used = used;

for(int i = 0; i < source.used; i++){

answer.erase\_one(source.data[i][1]);

}

return answer;

}

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

\*this = \*this - removeIt;

}

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

int answer;

int i;

answer = 0;

for(i = 0; i < used; ++i)

if (target == data[i][0])

++answer;

return answer;

}

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

if(this == &source)

return \*this;

if(data){

for(int i = 0; i < capacity; i++){

delete[] data[i];

}

delete[] data;

}

if(source.used == 0){

used = 0;

capacity = 20;

data = new int\*[20];

for(int i = 0; i < 20; i++){

data[i] = new int[2];

}

return \*this;

}

data = new int\*[source.capacity];

for(int i = 0; i < source.capacity; i++)

data[i] = new int[2];

for(int i = 0; i < source.used; i++){

data[i][0] = source.data[i][0];

data[i][1] = source.data[i][1];

}

capacity = source.capacity;

used =source.used;

return \*this;

}

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

bag answer;

answer = \*this;

if(source.used + used >= capacity)

answer.incSize();

for(int i = 0; i < source.used; i++){

if(!answer.contains(source.data[i][1])){

answer.insert(source.data[i][0],source.data[i][1]);

}

}

return answer;

}

bool bag:: contains(const int &target){

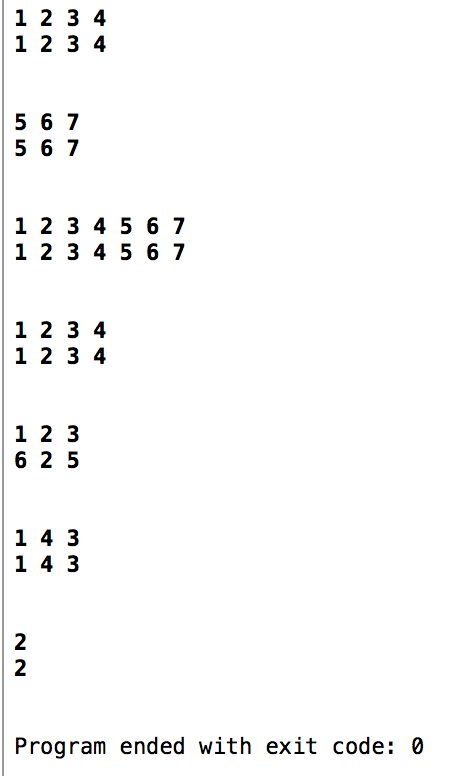
for(int i = 0; i < used; i++){

if(data[i][1] == target)

return true;

}

return false;

}

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);

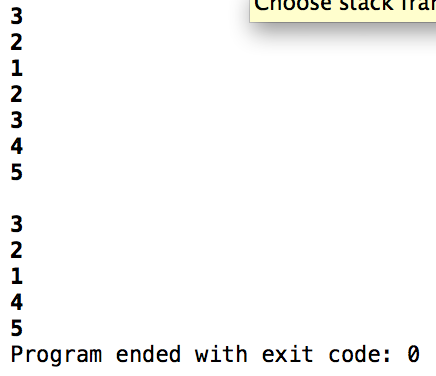
l1.add(3);

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();

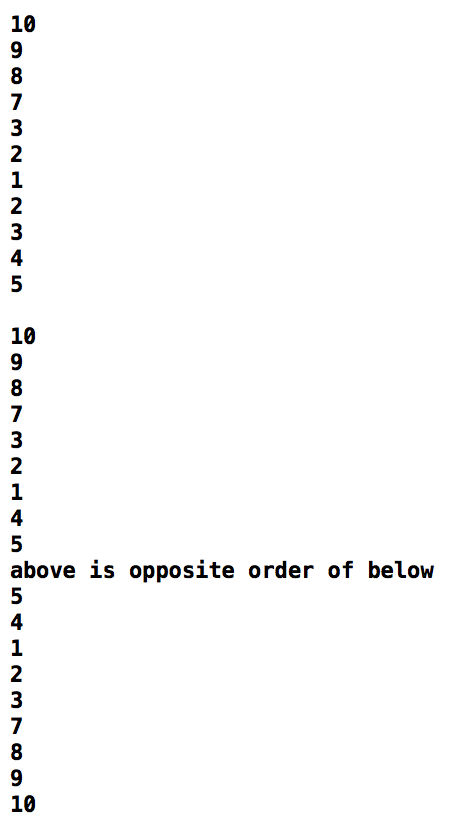
l1.reverseOrder(l1.getHead());

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);

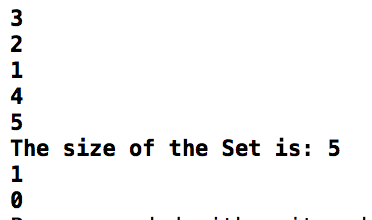
s1.display();

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);

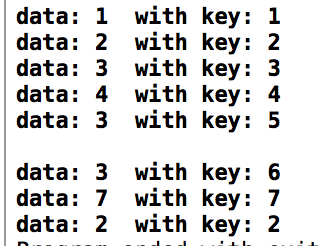
b2.insert(3,3);

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";

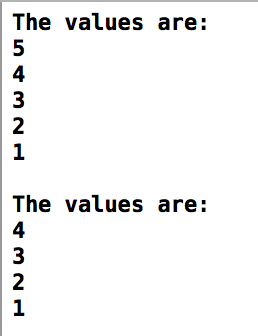
}

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;

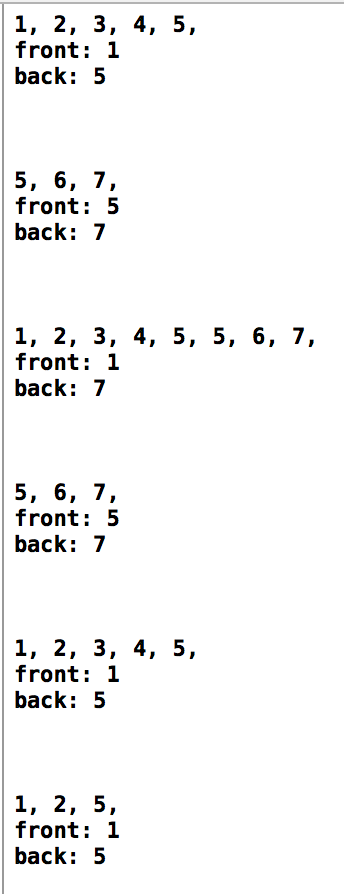
cout << "front: " << front->data() << endl;

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

cursor = front;

cout << endl << endl << endl;

}

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

// Yousef Zoumot

// main.cpp

// Coen70HW4.1 \*Chapter 6 Problem 2a

//

// Created by Yousef Zoumot on 2/14/16.

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

//

#include <iostream>

#include <cassert>

#include <vector>

using namespace std;

template < class T > class set{

public:

set(T x = 20);

set(const set& source);

~set();

T erase(const T& target);

bool erase\_one(const T& target);

void insert(const T& target);

set operator -(const set& b2);

set& operator =(const set& source);

void operator -=(const set& removeIt);

void operator +=(const set& addend);

set operator +(const set& b2);

bool contains(const T& target) const;

T size() const { return used; }

T count( const T& target) const;

void prT();

private:

T\* data;

T capacity;

void incSize();

T used;

};

int main(){

set<int> a;

set<int> b;

set<int> c;

set<int> d;

a.insert(2);

a.insert(2);

a.insert(4);

a.insert(5);

a.prT();

b.insert(4);

b.insert(2);

b.insert(6);

b.prT();

c = a - b;

c.prT();

c = a + b;

c.prT();

d.insert(7);

c += d;

c.prT();

c -= d;

c.prT();

c.erase\_one(3);

c.prT();

}

template<class T>

T set<T>::erase(const T& target){

T index = 0;

T many\_removed = 0;

while(index < used){

if (data[index] == target){

--used;

data[index] = data [used];

++many\_removed;

}

else

++index;

}

return many\_removed;

}

template<class T>

set<T>:: set(T x){

assert(x>0);

used = 0;

capacity = x;

data = new T[x];

}

template<class T>

set<T>:: set(const set& source){

data = NULL;

\*this = source;

}

template<class T>

set<T>:: ~set(){

if (data)

delete[] data;

}

template<class T>

void set<T>:: incSize(){

T\* temp = new T[2\*capacity];

for(T i = 0; i < capacity; i++){

temp[i] = data[i];

}

delete[] data;

data = temp;

capacity \*= 2;

}

template<class T>

void set<T>::prT(){

T i;

for(i = 0; i < used; i++){

cout << data[i] << ", ";

}

cout << endl;

}

template<class T>

bool set<T>::erase\_one(const T& target){

T index;

index = 0;

while((index < used) && (data[index] != target))

++index;

if(index == used)

return false;

--used;

data[index] = data[used];

return true;

}

template<class T>

void set<T>::insert(const T& entry){

if(contains(entry))

return;

if(size() >= capacity)

incSize();

data[used] = entry;

++used;

return;

}

template<class T>

void set<T>::operator +=(const set& addend){

T i;

if(size() + addend.size() >= capacity)

incSize();

for(i = 0; i < addend.used; i++){

if(!contains(addend.data[i])){

data[used] = addend.data[i];

used++;

}

}

}

template<class T>

set<T> set<T>:: operator -(const set& b2){

set answer = \*this;

for(T i = 0; i < b2.used; i++)

answer.erase\_one(b2.data[i]);

return answer;

}

template<class T>

void set<T>:: operator -=(const set& removeIt){

T i;

for(i = 0; i < removeIt.used; i++)

erase\_one(removeIt.data[i]);

}

template<class T>

T set<T>::count(const T& target) const {

T answer;

T i;

answer = 0;

for(i = 0; i < used; ++i)

if (target == data[i])

++answer;

return answer;

}

template<class T>

set<T>& set<T>:: operator =(const set& source){

if(this == &source)

return \*this;

if (data)

delete[] data;

if(source.used == 0){

used = 0;

capacity = 20;

data = new T[capacity];

return \*this;

}

data = new T[source.capacity];

for(T i = 0; i < source.capacity; i++){

data[i] = source.data[i];

}

used = source.used;

capacity = source.capacity;

return \*this;

}

template<class T>

set<T> set<T>::operator +(const set& b2){

set answer = \*this;

if(answer.size() + b2.size() >= capacity)

incSize();

for(T i = 0; i < b2.used; i++){

if(!answer.contains(b2.data[i])){

answer.data[used] = b2.data[i];

answer.used++;

}

}

return answer;

}

template<class T>

bool set<T>:: contains(const T& target) const{

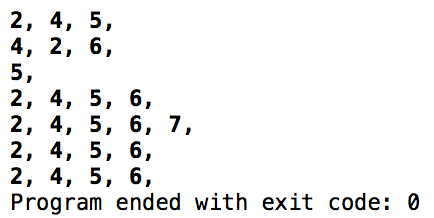
T i;

for(i = 0; i < used; ++i)

if (target == data[i])

return true;

return false;

}

// Yousef Zoumot

// main.cpp

// Coen70HW4.2 Chapter 6 Problem 2b

//

// Created by Yousef Zoumot on 2/14/16.

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

//

#include <iostream>

#include <assert.h>

#include <cstdlib>//Provides size\_t

using namespace std;

template <class T>

class sequence{

public:

//TYPEDEFS and MEMBER CONSTANTS

typedef std::size\_t size\_type;

static const size\_type CAPACITY=30;

//CONSTRUCTOR

sequence();

//MODIFICATION MEMBER FUNCTIONS

void start();

void advance();

void insert(const T& entry);

void attach(const T& entry);

void remove\_current();

void addToFront(const T& entry);

void removeFront();

void addToEnd(const T& entry);

void lastToCurrent();

sequence operator +(const sequence& s2);

void operator +=(const sequence& s2);

T operator[](size\_type index);

void printValues();

//CONSTANT MEMBER FUNCTIONS

size\_type size() const;

bool is\_item() const;

T current() const;

private:

T data[CAPACITY];

size\_type used;

size\_type current\_index;

};

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

// insert code here...

sequence<int> s1;

sequence<int> s2;

s1.addToEnd(1);

s1.addToEnd(2);

s1.addToEnd(3);

s1.addToEnd(4);

s1.addToEnd(5);

s2.addToEnd(6);

s2.addToEnd(7);

s2.addToEnd(8);

s2.addToEnd(9);

s1.printValues();

s2.printValues();

sequence<int> s3;

s3= s1+s2;

s3.printValues();

sequence<int> s4;

s4+=s1;

s4+=s2;

s4.printValues();

cout<<s4[0];

return 0;

}

// MODIFICATION MEMBER FUNCTIONS

template <class T>

sequence<T>::sequence ()

{

current\_index = 0;

used = 0;

}

template <class T>

void sequence<T>::start( )

{

current\_index = 0;

}

template <class T>

void sequence<T>::advance( )

{

current\_index++;

}

template <class T>

void sequence<T>::insert(const T& entry)

{

if(current\_index==used){

data[current\_index]=entry;

used++;

return;

}

size\_type i;

for (i = used; i > current\_index; i--)

data[i]= data[i-1];

data[current\_index] = entry;

used++;

}

template <class T>

void sequence<T>::attach(const T& entry)

{

if(!is\_item()){

data[current\_index]=entry;

used++;

return;

}

size\_type i;

for (i = used; i > current\_index+1; i--)

data[i] = data[i+1];

data[current\_index+1] = entry;

current\_index++;

used++;

}

template <class T>

void sequence<T>::remove\_current( )

{

size\_type i;

for (i= current\_index; i < used-1; i++)

data[i] = data[i+1];

used--;

}

template <class T>

void sequence<T>:: addToFront(const T& entry){

if(current\_index==used){

data[current\_index]=entry;

used++;

return;

}

size\_type i;

for (i = used; i > 0; i--)

data[i]= data[i-1];

data[0] = entry;

start();

used++;

}

template <class T>

void sequence<T>:: removeFront(){

start();

remove\_current();

}

template <class T>

void sequence<T>:: addToEnd(const T& entry){

current\_index=used;

data[current\_index]=entry;

used++;

}

template <class T>

void sequence<T>:: lastToCurrent(){

data[current\_index]=data[used-1];

used--;

}

template <class T>

T sequence<T>:: operator[](size\_type index){

T invalid=100000;

if(index<size())

return data[index];

else{

cout<<"This is not a valid index";

return invalid;

};

}

template <class T>

sequence<T> sequence<T>:: operator +(const sequence& s2){

sequence temp;

size\_type i=0;

size\_type f=0;

while(temp.size() < size()){

temp.data[i]=data[i];

i++;

temp.used++;

}

while (temp.size() < (size()+s2.size())) {

temp.data[i]=s2.data[f];

f++;

i++;

temp.used++;

}

return temp;

}

template <class T>

void sequence<T>:: operator +=(const sequence& s2){

\*this=\*this+s2;

}

template <class T>

void sequence<T>:: printValues(){

cout<<"The values in the sequence are as follows: "<<"\n";

size\_type i;

for(i=0; i<size(); i++)

cout<<data[i]<<" \n";

}

// CONSTANT MEMBER FUNCTIONS

template <class T>

size\_t sequence<T>::size( ) const

{

return used;

}

template <class T>

bool sequence<T>::is\_item( ) const

{

return current\_index != used;

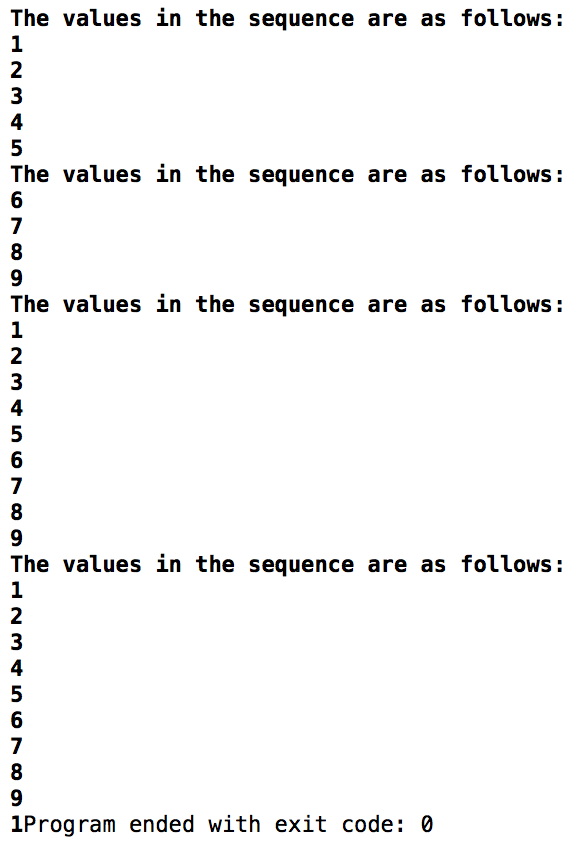
}

template <class T>

T sequence<T>::current( ) const

{

return data[current\_index];

}

// Yousef Zoumot

// main.cpp

// Coen70HW4.3 \*Chapter 6 Problem 2e

//

// Created by Yousef Zoumot on 2/14/16.

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

//

#include <iostream>

#include <cassert>

#include <cstdlib> //provide size\_t

using namespace std;

template<class T>

class Keyed\_Bag

{

public:

//CONSTRUCTOR

Keyed\_Bag();

//MODIFICATION

bool erase\_one(const T& target);

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

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

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

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

//CONSTANT MEMBER FUNCTIONS

T size() const { return used;}

T count(const T& target) const;

void prTValues();

private:

struct Node{

Node\* \_prev;

Node\* \_next;

T \_data;

T \_key;

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

this->\_data = data;

this->\_key=key;

this->\_prev = prev;

this->\_next = next;

}

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

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

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

};

Node\* head;

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

};

template<class T>

Keyed\_Bag<T>::Keyed\_Bag(){

head=NULL;

used=0;

}

//NONMEMBER FUNCTIONS for the Keyed\_Bag class

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

template<class T>

bool Keyed\_Bag<T>::erase\_one(const T& 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;

}

template<class T>

void Keyed\_Bag<T>::insert(const T& entry, T 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;

}

template<class T>

void Keyed\_Bag<T>:: prTValues(){//a function that prTs 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[]) {

// insert code here...

Keyed\_Bag<int> 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);

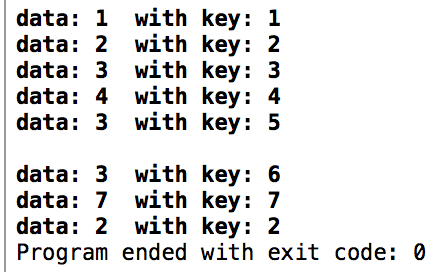
b2.insert(2,2);

b2.insert(3,3);

b2.insert(3,7);

b.prTValues();

b2.prTValues();

 return 0;

}

//

// main.cpp

// Coen70HW4.4

//

// Created by Yousef Zoumot on 2/14/16.

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

//

#include <iostream>

#include <cassert>

#include <cstdlib> //provide size\_t

#include <utility>

using namespace std;

template<class T, class K>

class Keyed\_Bag

{

public:

//CONSTRUCTOR

Keyed\_Bag();

//MODIFICATION

bool erase\_one(const T& target);

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

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

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

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

//CONSTANT MEMBER FUNCTIONS

T size() const { return used;}

T count(const T& target) const;

void prTValues();

private:

struct Node{

Node\* \_prev;

Node\* \_next;

T \_data;

K \_key;

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

this->\_data = data;

this->\_key=key;

this->\_prev = prev;

this->\_next = next;

}

T& first(){return \_data;};

K& second(){return \_key;};

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

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

};

Node\* head;

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

};

template<class T, class K>

Keyed\_Bag<T,K>::Keyed\_Bag(){

head=NULL;

used=0;

}

//NONMEMBER FUNCTIONS for the Keyed\_Bag class

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

template<class T, class K>

bool Keyed\_Bag<T,K>::erase\_one(const T& 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;

}

template<class T, class K>

void Keyed\_Bag<T,K>::insert(const T& entry, T 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;

}

template<class T, class K>

void Keyed\_Bag<T,K>:: prTValues(){//a function that prTs 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[]) {

// insert code here...

Keyed\_Bag<int, double> 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);

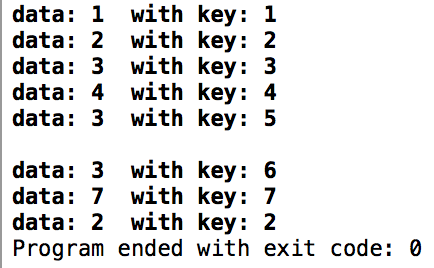
b2.insert(3,6);

b2.insert(7,7);

b2.insert(2,2);

b2.insert(3,3);

b2.insert(3,7);

 b.prTValues();

b2.prTValues();

return 0;

}

// Yousef Zoumot

// main.cpp

// Coen70HW4.5 Chapter 6 Problem 8

//

// Created by Yousef Zoumot on 2/14/16.

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

//

#include <iostream>

#include <cassert>

#include <cstdlib> //provide size\_t

#include <utility>

using namespace std;

class Gift{

char \_gift[40];

public:

void typeGift();

void printGift();

};

class Person{

private:

char name[40];

public:

Person(){used\_g = 0;};

Gift gifts[100];

void addGift(Gift& g);

void typeName();

void printName();

int used\_g;

};

class Gift\_List{

Person people[100];

int used\_p;

public:

Gift\_List(){used\_p=0;};

void addPerson(Person& p);

void removeLast();

void printList();

};

void Gift\_List:: printList(){

cout<<"The list is as follows: "<<"\n";

for(int i=0; i<used\_p; i++){

people[i].printName();

cout<< " has a gift list that consists of: "<<"\n";

for(int k=0; k<people[i].used\_g; k++ ){

people[i].gifts[k].printGift();

cout<<"\n";

}

}

}

void Gift\_List:: removeLast(){

used\_p--;

}

void Gift\_List:: addPerson(Person& p){

people[used\_p]=p;

used\_p++;

}

void Person:: addGift(Gift& g){

gifts[used\_g]=g;

used\_g++;

}

void Gift:: typeGift(){

cout<<"Please type a gift less that 40 characters long: "<< "\n";

cin>>\_gift;

}

void Gift:: printGift(){

cout<<\_gift;

}

void Person:: printName(){

cout<<name;

}

void Person:: typeName(){

cout<<"Please type a name less that 40 characters long: "<< "\n";

cin>>name;

}

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

// insert code here...

Person p1, p2, p3;

p1.typeName();

p2.typeName();

p3.typeName();

Gift g1, g2, g3, g4, g5, g6;

g1.typeGift();

g2.typeGift();

g3.typeGift();

g4.typeGift();

g5.typeGift();

g6.typeGift();

p1.addGift(g1);

p1.addGift(g4);

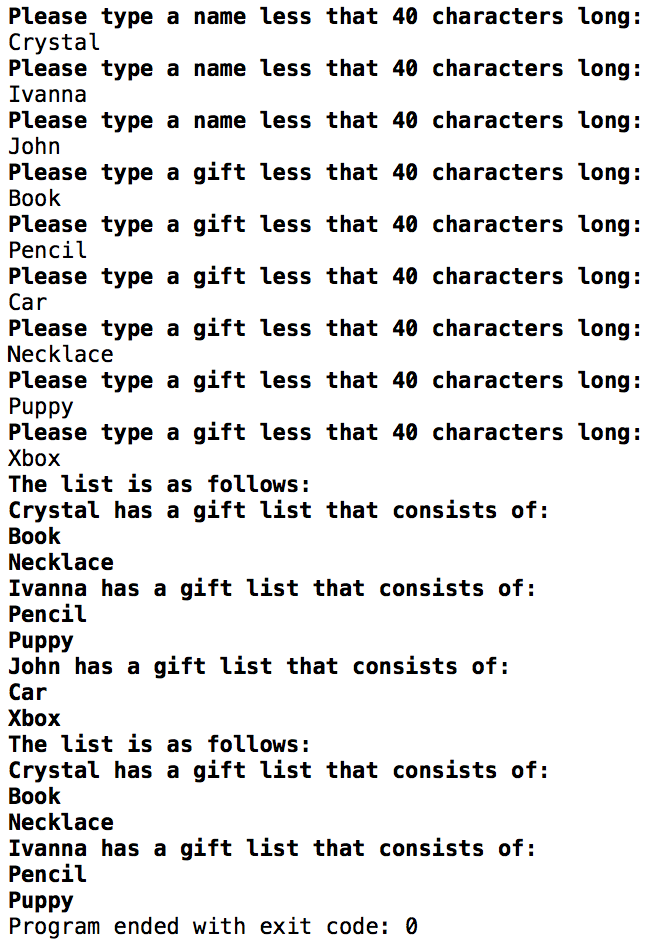
p2.addGift(g2);

p2.addGift(g5);

p3.addGift(g3);

p3.addGift(g6);

Gift\_List gl;

 gl.addPerson(p1);

gl.addPerson(p2);

gl.addPerson(p3);

gl.printList();

gl.removeLast();

gl.printList();

return 0;

}

// Yousef Zoumot

// main.cpp

// Coen70HW5.1 Chapter 11 Problem 1

//

// Created by Yousef Zoumot on 2/21/16.

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

//

#include <iostream>

#include <vector>

#include <utility>

using namespace std;

class Heap{

vector<pair<int, int>> data;

int count;

int order;

public:

Heap();

void push(int);

int pop();

int top(){return data[0].first;};

int size(){return count;};

bool isEmpty(){return count==0;};

int lc(int k){return (2\*k)+1;};

int rc(int k){return (2\*k)+2;};

int p(int k){return (k-1)/2;};

void printValues();

};

Heap:: Heap(){

vector<pair<int, int>> data(100);

count=0;

order=0;

}

void Heap:: push(int input){

if(count==0){

data.push\_back(pair<int, int> (input, order));

count++;

order++;

return;

}

int k=count;

pair<int, int> tmp;

data.push\_back(pair<int, int> (input, order));

tmp = data[k];

while(k>0 && data[ p(k) ].first < input){

data[k]=data[ p(k) ];

k= p(k);

}

data[k]= tmp;

count++;

order++;

}

int Heap:: pop(){

int i, x, child, max, xo;

max=data[0].first;

x=data[count-1].first;

xo=data[count-1].second;

i=0;

while(lc(i) < count-1){

child= lc(i);

if(rc(i) < count && data[lc(i)] < data[rc(i)])

child=rc(i);

if(data[lc(i)].first == data[rc(i)].first){

if(data[lc(i)].second < data[rc(i)].second)

child=lc(i);

else

child=rc(i);

}

if(x < data[child].first ){

data[i]=data[child];

i=child;

}else

break;

}

data[i].first=x;

data[i].second=xo;

count--;

return max;

}

void Heap:: printValues(){

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

cout<<data[i].first<<" "<<data[i].second<< "\n";

cout<<"\n";

}

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

// insert code here...

Heap h1;

h1.push(1);

h1.push(2);

h1.push(3);

h1.push(4);

h1.push(4);

h1.push(5);

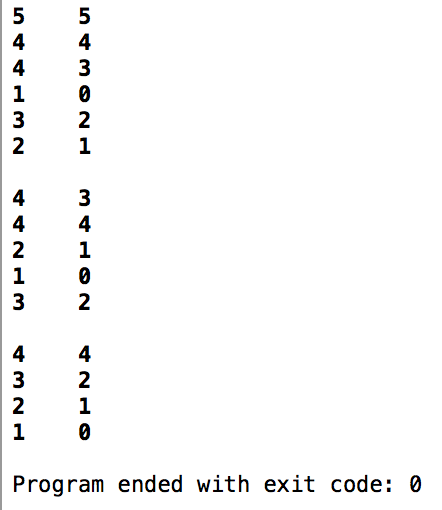
//cout<<h1.isEmpty();

h1.printValues();

h1.pop();

h1.printValues();

h1.pop();

 h1.printValues();

return 0;

}

// Yousef Zoumot

// main.cpp

// Coen70HW5.2 Chapter 11 Problem 5

//

// Created by Yousef Zoumot on 2/21/16.

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

//

#include <iostream>

#include <vector>

#include <utility>

using namespace std;

class Heap{

vector<pair<int, int>> data;

int count;

int order;

public:

Heap();

void push(int);

int pop();

int top(){return data[0].first;};

int size(){return count;};

bool isEmpty(){return count==0;};

int lc(int k){return (2\*k)+1;};

int rc(int k){return (2\*k)+2;};

int p(int k){return (k-1)/2;};

void printValues();

};

Heap:: Heap(){

vector<pair<int, int>> data(100);

count=0;

order=0;

}

void Heap:: push(int input){

if(count==0){

data.push\_back(pair<int, int> (input, order));

count++;

order++;

return;

}

int k=count;

pair<int, int> tmp;

data.push\_back(pair<int, int> (input, order));

tmp = data[k];

while(k>0 && data[ p(k) ].first > input){

data[k]=data[ p(k) ];

k= p(k);

}

data[k]= tmp;

count++;

order++;

}

int Heap:: pop(){

int i, x, child, max, xo;

max=data[0].first;

x=data[count-1].first;

xo=data[count-1].second;

i=0;

while(lc(i) < count-1){

child= lc(i);

if(rc(i) < count && data[lc(i)] > data[rc(i)])

child=rc(i);

if(data[lc(i)].first == data[rc(i)].first){

if(data[lc(i)].second < data[rc(i)].second)

child=lc(i);

else

child=rc(i);

}

if(x > data[child].first ){

data[i]=data[child];

i=child;

}else

break;

}

data[i].first=x;

data[i].second=xo;

count--;

return max;

}

void Heap:: printValues(){

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

cout<<data[i].first<<" "<<data[i].second<< "\n";

cout<<"\n";

}

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

// insert code here...

Heap h1;

h1.push(5);

h1.push(4);

h1.push(3);

h1.push(2);

h1.push(1);

//cout<<h1.isEmpty();

h1.printValues();

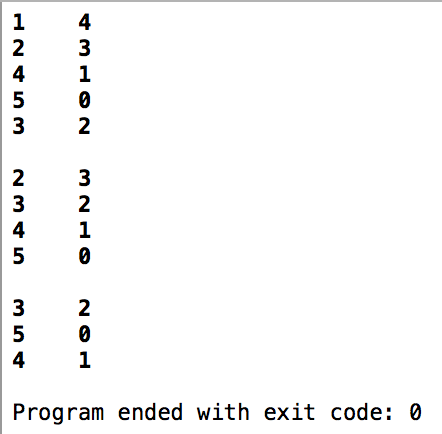
h1.pop();

h1.printValues();

h1.pop();

h1.printValues();

return 0;

}

// Yousef Zoumot

// main.cpp

// Coen70HW6.1 \*Chapter 10 Problem #2

//

// Created by Yousef Zoumot on 3/6/16.

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

//

#include <iostream>

#include <math.h>

using namespace std;

template <class T>

struct Node {

T value;

Node \*left;

Node \*right;

Node(T val) {

this->value = val;

}

Node(T val, Node<T> left, Node<T> right) {

this->value = val;

this->left = left;

this->right = right;

}

};

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

template <class T>

class BinaryTree {

private:

Node<T> \*root;

void addRecursive(Node<T> \*root, T val);

void printRecursive(Node<T> \*root);

int nodesCountRecursive(Node<T> \*root);

int heightRecursive(Node<T> \*root);

bool deleteValueRecursive(Node<T>\* parent, Node<T>\* current, T value);

public:

void add(T val);

void print();

int nodesCount();

int height();

bool deleteValue(T value);

};

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

template <class T>

void BinaryTree<T>:: addRecursive(Node<T> \*root, T val) {

if (root->value > val) {

if (!root->left) {

root->left = new Node<T>(val);

} else {

addRecursive(root->left, val);

}

} else {

if (!root->right) {

root->right = new Node<T>(val);

} else {

addRecursive(root->right, val);

}

}

}

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

template <class T>

void BinaryTree<T>:: printRecursive(Node<T> \*root) {

if (!root) return;

printRecursive(root->left);

cout<<root->value<<' ';

printRecursive(root->right);

}

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

template <class T>

int BinaryTree<T>:: nodesCountRecursive(Node<T> \*root) {

if (!root) return 0;

else return 1 + nodesCountRecursive(root->left) + nodesCountRecursive(root->right);

}

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

template <class T>

int BinaryTree<T>:: heightRecursive(Node<T> \*root) {

if (!root) return 0;

else return 1 + max(heightRecursive(root->left), heightRecursive(root->right));

}

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

template <class T>

bool BinaryTree<T>:: deleteValueRecursive(Node<T>\* parent, Node<T>\* current, T value) {

if (!current) return false;

if (current->value == value) {

if (current->left == NULL || current->right == NULL) {

Node<T>\* temp = current->left;

if (current->right) temp = current->right;

if (parent) {

if (parent->left == current) {

parent->left = temp;

} else {

parent->right = temp;

}

} else {

this->root = temp;

}

} else {

Node<T>\* substitute = current->right;

while (substitute->left) {

substitute = substitute->left;

}

T temp = current->value;

current->value = substitute->value;

substitute->value = temp;

return deleteValueRecursive(current, current->right, temp);

}

delete current;

return true;

}

return deleteValueRecursive(current, current->left, value) ||

deleteValueRecursive(current, current->right, value);

}

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

template <class T>

void BinaryTree<T>:: add(T val) {

if (root) {

this->addRecursive(root, val);

} else {

root = new Node<T>(val);

}

}

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

template <class T>

void BinaryTree<T>:: print() {

printRecursive(this->root);

cout<<"\n";

}

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

template <class T>

int BinaryTree<T>:: nodesCount() {

return nodesCountRecursive(root);

}

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

template <class T>

int BinaryTree<T>:: height() {

return heightRecursive(this->root);

}

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

template <class T>

bool BinaryTree<T>:: deleteValue(T value) {

return this->deleteValueRecursive(NULL, this->root, value);

}

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

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

// insert code here...

BinaryTree<int> \*bst1=new BinaryTree<int>();

bst1->add(5);

bst1->add(4);

bst1->add(7);

bst1->add(2);

bst1->add(9);

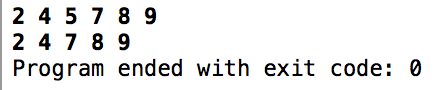
bst1->add(8);

bst1->print();

bst1->deleteValue(5);

bst1->print();

return 0;

}

// Yousef Zoumot

// main.cpp

// Coen70HW6.2 \*Chapter 10 Problem #3

//

// Created by Yousef Zoumot on 3/6/16.

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

//

#include<iostream>

#include<cstdio>

#include<sstream>

#include<algorithm>

#define pow2(n) (1 << (n))

using namespace std;

struct avl\_Node

{

int data;

struct avl\_Node \*left;

struct avl\_Node \*right;

};

class avlTree

{

public:

avlTree()

{

root = NULL;

}

int height(){return heightRecursive(root);};

void insert(int value){root=insertRecursive(root , value);};

void remove(int value){root=removeRecursive(root, value);};

void display();

void inOrder();

void preOrder();

void postOrder();

private:

avl\_Node \*root;

int heightRecursive(avl\_Node \*);

int heightDifferenceRecursive(avl\_Node \*);

avl\_Node \*rightright\_rotationRecursive(avl\_Node \*);

avl\_Node \*leftleft\_rotationRecursive(avl\_Node \*);

avl\_Node \*leftright\_rotationRecursive(avl\_Node \*);

avl\_Node \*rightleft\_rotationRecursive(avl\_Node \*);

avl\_Node\* balanceRecursive(avl\_Node \*);

avl\_Node\* insertRecursive(avl\_Node \*, int );

avl\_Node\* removeRecursive(avl\_Node \*, int );

void displayRecursive(avl\_Node \*, int);

void inOrderRecursive(avl\_Node \*);

void preOrderRecursive(avl\_Node \*);

void postOrderRecursive(avl\_Node \*);

avl\_Node\* minValueNode(avl\_Node\* node);

};

void avlTree:: display(){

if(root ==NULL)

cout<<"This AVL Tree is empty"<< "\n";

else{

displayRecursive(root, 1);

}

cout<<"\n";

cout<<"\n";

cout<<"\n";

cout<<"\n";

cout<<"\n";

cout<<"\n";

}

void avlTree:: preOrder(){

preOrderRecursive(root);

}

void avlTree:: inOrder(){

inOrderRecursive(root);

}

void avlTree:: postOrder(){

postOrderRecursive(root);

}

avl\_Node\* avlTree:: minValueNode(avl\_Node\* node)

{

avl\_Node\* current = node;

/\* loop down to find the leftmost leaf \*/

while (current->left != NULL)

current = current->left;

return current;

}

//\* Height of AVL Tree

int avlTree::heightRecursive(avl\_Node \*temp)

{

int h = 0;

if (temp != NULL)

{

int l\_height = heightRecursive (temp->left);

int r\_height = heightRecursive (temp->right);

int max\_height = max (l\_height, r\_height);

h = max\_height + 1;

}

return h;

}

// \* Height Difference

int avlTree::heightDifferenceRecursive(avl\_Node \*temp)

{

int l\_height = heightRecursive (temp->left);

int r\_height = heightRecursive (temp->right);

int b\_factor= l\_height - r\_height;

return b\_factor;

}

// Right- Right rotationRecursive

avl\_Node \*avlTree::rightright\_rotationRecursive(avl\_Node \*parent)

{

avl\_Node \*temp;

temp = parent->right;

parent->right = temp->left;

temp->left = parent;

return temp;

}

// Left- Left rotationRecursive

avl\_Node \*avlTree::leftleft\_rotationRecursive(avl\_Node \*parent)

{

avl\_Node \*temp;

temp = parent->left;

parent->left = temp->right;

temp->right = parent;

return temp;

}

// Left - Right rotationRecursive

avl\_Node \*avlTree::leftright\_rotationRecursive(avl\_Node \*parent)

{

avl\_Node \*temp;

temp = parent->left;

parent->left = rightright\_rotationRecursive (temp);

return leftleft\_rotationRecursive (parent);

}

// Right- Left rotationRecursive

avl\_Node \*avlTree::rightleft\_rotationRecursive(avl\_Node \*parent)

{

avl\_Node \*temp;

temp = parent->right;

parent->right = leftleft\_rotationRecursive (temp);

return rightright\_rotationRecursive (parent);

}

// Balancing AVL Tree

avl\_Node \*avlTree::balanceRecursive(avl\_Node \*temp)

{

int bal\_factor = heightDifferenceRecursive (temp);

if (bal\_factor > 1)

{

if (heightDifferenceRecursive (temp->left) > 0)

temp = leftleft\_rotationRecursive (temp);

else

temp = leftright\_rotationRecursive (temp);

}

else if (bal\_factor < -1)

{

if (heightDifferenceRecursive (temp->right) > 0)

temp = rightleft\_rotationRecursive (temp);

else

temp = rightright\_rotationRecursive (temp);

}

return temp;

}

//insertRecursive Element into the tree

avl\_Node \*avlTree::insertRecursive(avl\_Node \*root, int value)

{

if (root == NULL)

{

root = new avl\_Node;

root->data = value;

root->left = NULL;

root->right = NULL;

return root;

}

else if (value < root->data)

{

root->left = insertRecursive(root->left, value);

root = balanceRecursive (root);

}

else if (value >= root->data)

{

root->right = insertRecursive(root->right, value);

root = balanceRecursive (root);

}

return root;

}

//removes element from tree

avl\_Node \*avlTree:: removeRecursive(avl\_Node \*root, int key)

{

// PERFORM STANDARD BST DELETE

if (root == NULL)

return root;

// If the key to be deleted is smaller than the root's key,

// then it lies in left subtree

if ( key < root->data )

root->left = removeRecursive(root->left, key);

// If the key to be deleted is greater than the root's key,

// then it lies in right subtree

else if( key > root->data )

root->right = removeRecursive(root->right, key);

// if key is same as root's key, then This is the node

// to be deleted

else

{

// node with only one child or no child

if( (root->left == NULL) || (root->right == NULL) )

{

avl\_Node\* temp = root->left ? root->left : root->right;

// No child case

if(temp == NULL)

{

temp = root;

root = NULL;

}

else // One child case

\*root = \*temp; // Copy the contents of the non-empty child

delete temp;

}

else

{

// node with two children: Get the inorder successor (smallest

// in the right subtree)

avl\_Node\* temp = minValueNode(root->right);

// Copy the inorder successor's data to this node

root->data = temp->data;

// Delete the inorder successor

root->right = removeRecursive(root->right, temp->data);

}

}

// If the tree had only one node

if (root == NULL)

return root;

// GET THE BALANCE FACTOR OF THIS NODE (to check whether

// this node became unbalanced)

int balance = heightDifferenceRecursive(root);

// If unbalanced, there are 4 cases

// Left Left Case

if (balance > 1 && heightDifferenceRecursive(root->left) >= 0)

return leftleft\_rotationRecursive(root);

// Left Right Case

if (balance > 1 && heightDifferenceRecursive(root->left) < 0)

{

return leftright\_rotationRecursive(root);

}

// Right Right Case

if (balance < -1 && heightDifferenceRecursive(root->right) <= 0)

return rightright\_rotationRecursive(root);

// Right Left Case

if (balance < -1 && heightDifferenceRecursive(root->right) > 0)

{

return rightleft\_rotationRecursive(root);

}

return root;

}

//displayRecursive AVL Tree

void avlTree::displayRecursive(avl\_Node \*ptr, int level)

{

int i;

if (ptr!=NULL)

{

displayRecursive(ptr->right, level + 1);

printf("\n");

if (ptr == root)

cout<<"Root -> ";

for (i = 0; i < level && ptr != root; i++)

cout<<" ";

cout<<ptr->data;

displayRecursive(ptr->left, level + 1);

}

}

//inOrderRecursive Traversal of AVL Tree

void avlTree::inOrderRecursive(avl\_Node \*tree)

{

if (tree == NULL)

return;

inOrderRecursive (tree->left);

cout<<tree->data<<" ";

inOrderRecursive (tree->right);

}

// preOrderRecursive Traversal of AVL Tree

void avlTree::preOrderRecursive(avl\_Node \*tree)

{

if (tree == NULL)

return;

cout<<tree->data<<" ";

preOrderRecursive (tree->left);

preOrderRecursive (tree->right);

}

// postOrderRecursive Traversal of AVL Tree

void avlTree::postOrderRecursive(avl\_Node \*tree)

{

if (tree == NULL)

return;

postOrderRecursive ( tree ->left );

postOrderRecursive ( tree ->right );

cout<<tree->data<<" ";

}

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

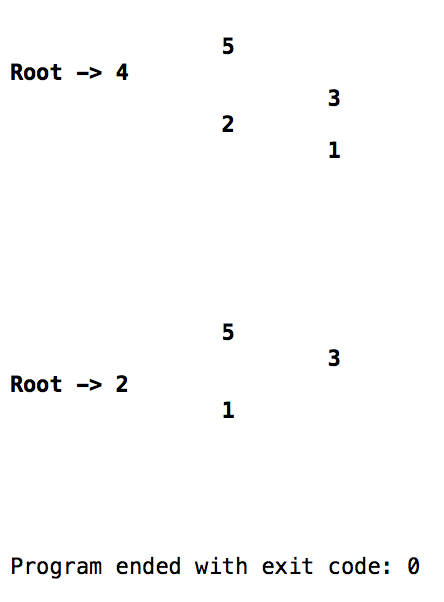
avlTree avlt1 = avlTree();

avlt1.insert(5);

avlt1.insert(4);

avlt1.insert(3);

avlt1.insert(2);

 avlt1.insert(1);

avlt1.display();

avlt1.remove(4);

avlt1.display();

}

// Yousef Zoumot

// main.cpp

// Coen70HW6.3 \* Chapter 10 Problem 4

//

// Created by Yousef Zoumot on 3/6/16.

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

//

#include <iostream>

#include <math.h>

#include <vector>

#include <utility>

#define p(x) (((x)-1)/2) //returns parent location

#define l(x) ((x)\*2+1) //returns left child location

#define r(x) ((x)\*2+2) //returns right child location

using namespace std;

template <class T>

struct Node {

T value;

Node \*left;

Node \*right;

Node(T val) {

this->value = val;

}

Node(T val, Node<T> left, Node<T> right) {

this->value = val;

this->left = left;

this->right = right;

}

};

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

template <class T>

class BinaryTree {

private:

Node<T> \*root;

int count;

void addRecursive(Node<T> \*root, T val);

void printRecursive(Node<T> \*root);

int nodesCountRecursive(Node<T> \*root);

int heightRecursive(Node<T> \*root);

bool deleteValueRecursive(Node<T>\* parent, Node<T>\* current, T value);

public:

vector<bool > is\_present;

void updatePresent1();

void updatePresent2(Node<T> \*tree, int);

void is\_present\_user(int i);

int size(){return count;};

void add(T val);

void print();

int nodesCount();

int height();

bool deleteValue(T value);

};

template<class T>

void BinaryTree<T>:: updatePresent1(){

is\_present.resize(pow(2, height())-1);

updatePresent2(root, 0);

}

template<class T>

void BinaryTree<T>:: updatePresent2(Node<T> \*tree, int index){

if (tree == NULL){

is\_present[index]=false;

return;

}

else{

is\_present[index]=true;

}

index=l(index);

updatePresent2(tree->left, index);

index=r(index);

updatePresent2(tree->right, index);

}

template <class T>

void BinaryTree<T>:: is\_present\_user(int i){

updatePresent1();

if(is\_present[i]){

cout<<"True";

}

else{

cout<<"False";

}

cout<<"\n";

}

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

template <class T>

void BinaryTree<T>:: addRecursive(Node<T> \*root, T val) {

if (root->value > val) {

if (!root->left) {

root->left = new Node<T>(val);

} else {

addRecursive(root->left, val);

}

} else {

if (!root->right) {

root->right = new Node<T>(val);

} else {

addRecursive(root->right, val);

}

}

}

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

template <class T>

void BinaryTree<T>:: printRecursive(Node<T> \*root) {

if (!root) return;

printRecursive(root->left);

cout<<root->value<<' ';

printRecursive(root->right);

}

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

template <class T>

int BinaryTree<T>:: nodesCountRecursive(Node<T> \*root) {

if (!root) return 0;

else return 1 + nodesCountRecursive(root->left) + nodesCountRecursive(root->right);

}

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

template <class T>

int BinaryTree<T>:: heightRecursive(Node<T> \*root) {

if (!root) return 0;

else return 1 + max(heightRecursive(root->left), heightRecursive(root->right));

}

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

template <class T>

bool BinaryTree<T>:: deleteValueRecursive(Node<T>\* parent, Node<T>\* current, T value) {

if (!current) return false;

if (current->value == value) {

if (current->left == NULL || current->right == NULL) {

Node<T>\* temp = current->left;

if (current->right) temp = current->right;

if (parent) {

if (parent->left == current) {

parent->left = temp;

} else {

parent->right = temp;

}

} else {

this->root = temp;

}

} else {

Node<T>\* substitute = current->right;

while (substitute->left) {

substitute = substitute->left;

}

T temp = current->value;

current->value = substitute->value;

substitute->value = temp;

return deleteValueRecursive(current, current->right, temp);

}

delete current;

return true;

}

return deleteValueRecursive(current, current->left, value) ||

deleteValueRecursive(current, current->right, value);

}

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

template <class T>

void BinaryTree<T>:: add(T val) {

if (root) {

this->addRecursive(root, val);

} else {

root = new Node<T>(val);

}

}

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

template <class T>

void BinaryTree<T>:: print() {

printRecursive(this->root);

cout<<"\n";

}

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

template <class T>

int BinaryTree<T>:: nodesCount() {

return nodesCountRecursive(root);

}

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

template <class T>

int BinaryTree<T>:: height() {

return heightRecursive(this->root);

}

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

template <class T>

bool BinaryTree<T>:: deleteValue(T value) {

return this->deleteValueRecursive(NULL, this->root, value);

}

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

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

// insert code here...

BinaryTree<int> \*bst1=new BinaryTree<int>();

bst1->add(5);

bst1->add(4);

bst1->add(7);

bst1->add(2);

bst1->add(9);

bst1->add(8);

bst1->print();

bst1->deleteValue(5);

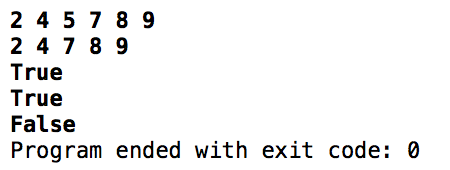
bst1->print();

bst1->is\_present\_user(0);

bst1->is\_present\_user(1);

bst1->is\_present\_user(7);

return 0;

}

// Yousef Zoumot

// main.cpp

// Coen70HW7.1 Chapter 10 Problem#8

//

// Created by Yousef Zoumot on 3/13/16.

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

//

#include <iostream>

#include <math.h>

#include <list>

using namespace std;

template <class T>

struct Node {

T value;

Node \*left;

Node \*right;

Node(T val) {

this->value = val;

}

Node(T val, Node<T> left, Node<T> right) {

this->value = val;

this->left = &left;

this->right = &right;

}

};

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

template <class T>

class BinaryTree {

private:

Node<T> \*root;

void addRecursive(Node<T> \*root, T val);

void printRecursive(Node<T> \*root);

int nodesCountRecursive(Node<T> \*root);

int heightRecursive(Node<T> \*root);

bool deleteValueRecursive(Node<T>\* parent, Node<T>\* current, T value);

public:

Node<T>\* getRoot(){return root;};

void add(T val);

void print();

int nodesCount();

int height();

bool deleteValue(T value);

};

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

template<class T>

class LinkedList: public BinaryTree<T>{

list<T> head;

public:

LinkedList(BinaryTree<T>);

void linkedListRecursion(Node<T> \*root);

void pushBack(T);

void printLinkedList();

};

template<class T>

void LinkedList<T>:: pushBack(T val){

head.push\_back(val);

}

template<class T>

LinkedList<T>::LinkedList(BinaryTree<T> bT){

linkedListRecursion(bT.getRoot());

}

template<class T>

void LinkedList<T>:: linkedListRecursion(Node<T> \*root){

if(!root){

return;

}

linkedListRecursion(root->left);

pushBack(root->value);

linkedListRecursion(root->right);

}

template<class T>

void LinkedList<T>:: printLinkedList(){

typename list<T>:: iterator it;

for(it=head.begin(); it!= head.end(); it++){

cout<< \*it <<" ";

}

cout<<"\n";

}

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

template <class T>

void BinaryTree<T>:: addRecursive(Node<T> \*root, T val) {

if (root->value > val) {

if (!root->left) {

root->left = new Node<T>(val);

} else {

addRecursive(root->left, val);

}

} else {

if (!root->right) {

root->right = new Node<T>(val);

} else {

addRecursive(root->right, val);

}

}

}

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

template <class T>

void BinaryTree<T>:: printRecursive(Node<T> \*root) {

if (!root) return;

printRecursive(root->left);

cout<<root->value<<' ';

printRecursive(root->right);

}

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

template <class T>

int BinaryTree<T>:: nodesCountRecursive(Node<T> \*root) {

if (!root) return 0;

else return 1 + nodesCountRecursive(root->left) + nodesCountRecursive(root->right);

}

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

template <class T>

int BinaryTree<T>:: heightRecursive(Node<T> \*root) {

if (!root) return 0;

else return 1 + max(heightRecursive(root->left), heightRecursive(root->right));

}

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

template <class T>

bool BinaryTree<T>:: deleteValueRecursive(Node<T>\* parent, Node<T>\* current, T value) {

if (!current) return false;

if (current->value == value) {

if (current->left == NULL || current->right == NULL) {

Node<T>\* temp = current->left;

if (current->right) temp = current->right;

if (parent) {

if (parent->left == current) {

parent->left = temp;

} else {

parent->right = temp;

}

} else {

this->root = temp;

}

} else {

Node<T>\* substitute = current->right;

while (substitute->left) {

substitute = substitute->left;

}

T temp = current->value;

current->value = substitute->value;

substitute->value = temp;

return deleteValueRecursive(current, current->right, temp);

}

delete current;

return true;

}

return deleteValueRecursive(current, current->left, value) ||

deleteValueRecursive(current, current->right, value);

}

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

template <class T>

void BinaryTree<T>:: add(T val) {

if (root) {

this->addRecursive(root, val);

} else {

root = new Node<T>(val);

}

}

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

template <class T>

void BinaryTree<T>:: print() {

printRecursive(this->root);

cout<<"\n";

}

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

template <class T>

int BinaryTree<T>:: nodesCount() {

return nodesCountRecursive(root);

}

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

template <class T>

int BinaryTree<T>:: height() {

return heightRecursive(this->root);

}

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

template <class T>

bool BinaryTree<T>:: deleteValue(T value) {

return this->deleteValueRecursive(NULL, this->root, value);

}

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

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

// insert code here...

BinaryTree<int> \*bst1=new BinaryTree<int>();

bst1->add(5);

bst1->add(4);

bst1->add(7);

bst1->add(2);

bst1->add(9);

bst1->add(8);

bst1->print();

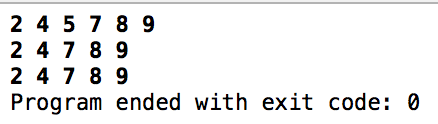
bst1->deleteValue(5);

bst1->print();

LinkedList<int> \*linkedL= new LinkedList<int>(\*bst1);

linkedL->printLinkedList();

return 0;

}

// Yousef Zoumot

// main.cpp

// Coen70HW7.2 Chapter 10 Problem 9

//

// Created by Yousef Zoumot on 3/13/16.

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

//

#include<iostream>

#include<cstdio>

#include<sstream>

#include<algorithm>

#include <list>

//#define pow2(n) (1 << (n))

using namespace std;

struct avl\_Node

{

int data;

struct avl\_Node \*left;

struct avl\_Node \*right;

};

class avlTree

{

public:

avlTree()

{

root = NULL;

}

avlTree(list<int>);

int height(){return heightRecursive(root);};

void insert(int value){root=insertRecursive(root , value);};

void remove(int value){root=removeRecursive(root, value);};

void display();

void inOrder();

void preOrder();

void postOrder();

private:

avl\_Node \*root;

int heightRecursive(avl\_Node \*);

int heightDifferenceRecursive(avl\_Node \*);

avl\_Node \*rightright\_rotationRecursive(avl\_Node \*);

avl\_Node \*leftleft\_rotationRecursive(avl\_Node \*);

avl\_Node \*leftright\_rotationRecursive(avl\_Node \*);

avl\_Node \*rightleft\_rotationRecursive(avl\_Node \*);

avl\_Node\* balanceRecursive(avl\_Node \*);

avl\_Node\* insertRecursive(avl\_Node \*, int );

avl\_Node\* removeRecursive(avl\_Node \*, int );

void displayRecursive(avl\_Node \*, int);

void inOrderRecursive(avl\_Node \*);

void preOrderRecursive(avl\_Node \*);

void postOrderRecursive(avl\_Node \*);

avl\_Node\* minValueNode(avl\_Node\* node);

};

avlTree::avlTree(list<int> source){

root=NULL;

typename list<int>::iterator it;

for(it=source.begin(); it!=source.end(); it++){

insert(\*it);

}

}

void avlTree:: display(){

if(root ==NULL)

cout<<"This AVL Tree is empty"<< "\n";

else{

displayRecursive(root, 1);

}

cout<<"\n";

cout<<"\n";

cout<<"\n";

cout<<"\n";

cout<<"\n";

cout<<"\n";

}

void avlTree:: preOrder(){

preOrderRecursive(root);

}

void avlTree:: inOrder(){

inOrderRecursive(root);

}

void avlTree:: postOrder(){

postOrderRecursive(root);

}

avl\_Node\* avlTree:: minValueNode(avl\_Node\* node)

{

avl\_Node\* current = node;

/\* loop down to find the leftmost leaf \*/

while (current->left != NULL)

current = current->left;

return current;

}

//\* Height of AVL Tree

int avlTree::heightRecursive(avl\_Node \*temp)

{

int h = 0;

if (temp != NULL)

{

int l\_height = heightRecursive (temp->left);

int r\_height = heightRecursive (temp->right);

int max\_height = max (l\_height, r\_height);

h = max\_height + 1;

}

return h;

}

// \* Height Difference

int avlTree::heightDifferenceRecursive(avl\_Node \*temp)

{

int l\_height = heightRecursive (temp->left);

int r\_height = heightRecursive (temp->right);

int b\_factor= l\_height - r\_height;

return b\_factor;

}

// Right- Right rotationRecursive

avl\_Node \*avlTree::rightright\_rotationRecursive(avl\_Node \*parent)

{

avl\_Node \*temp;

temp = parent->right;

parent->right = temp->left;

temp->left = parent;

return temp;

}

// Left- Left rotationRecursive

avl\_Node \*avlTree::leftleft\_rotationRecursive(avl\_Node \*parent)

{

avl\_Node \*temp;

temp = parent->left;

parent->left = temp->right;

temp->right = parent;

return temp;

}

// Left - Right rotationRecursive

avl\_Node \*avlTree::leftright\_rotationRecursive(avl\_Node \*parent)

{

avl\_Node \*temp;

temp = parent->left;

parent->left = rightright\_rotationRecursive (temp);

return leftleft\_rotationRecursive (parent);

}

// Right- Left rotationRecursive

avl\_Node \*avlTree::rightleft\_rotationRecursive(avl\_Node \*parent)

{

avl\_Node \*temp;

temp = parent->right;

parent->right = leftleft\_rotationRecursive (temp);

return rightright\_rotationRecursive (parent);

}

// Balancing AVL Tree

avl\_Node \*avlTree::balanceRecursive(avl\_Node \*temp)

{

int bal\_factor = heightDifferenceRecursive (temp);

if (bal\_factor > 1)

{

if (heightDifferenceRecursive (temp->left) > 0)

temp = leftleft\_rotationRecursive (temp);

else

temp = leftright\_rotationRecursive (temp);

}

else if (bal\_factor < -1)

{

if (heightDifferenceRecursive (temp->right) > 0)

temp = rightleft\_rotationRecursive (temp);

else

temp = rightright\_rotationRecursive (temp);

}

return temp;

}

//insertRecursive Element into the tree

avl\_Node \*avlTree::insertRecursive(avl\_Node \*root, int value)

{

if (root == NULL)

{

root = new avl\_Node;

root->data = value;

root->left = NULL;

root->right = NULL;

return root;

}

else if (value < root->data)

{

root->left = insertRecursive(root->left, value);

root = balanceRecursive (root);

}

else if (value >= root->data)

{

root->right = insertRecursive(root->right, value);

root = balanceRecursive (root);

}

return root;

}

//removes element from tree

avl\_Node \*avlTree:: removeRecursive(avl\_Node \*root, int key)

{

// PERFORM STANDARD BST DELETE

if (root == NULL)

return root;

// If the key to be deleted is smaller than the root's key,

// then it lies in left subtree

if ( key < root->data )

root->left = removeRecursive(root->left, key);

// If the key to be deleted is greater than the root's key,

// then it lies in right subtree

else if( key > root->data )

root->right = removeRecursive(root->right, key);

// if key is same as root's key, then This is the node

// to be deleted

else

{

// node with only one child or no child

if( (root->left == NULL) || (root->right == NULL) )

{

avl\_Node\* temp = root->left ? root->left : root->right;

// No child case

if(temp == NULL)

{

temp = root;

root = NULL;

}

else // One child case

\*root = \*temp; // Copy the contents of the non-empty child

delete temp;

}

else

{

// node with two children: Get the inorder successor (smallest

// in the right subtree)

avl\_Node\* temp = minValueNode(root->right);

// Copy the inorder successor's data to this node

root->data = temp->data;

// Delete the inorder successor

root->right = removeRecursive(root->right, temp->data);

}

}

// If the tree had only one node

if (root == NULL)

return root;

// GET THE BALANCE FACTOR OF THIS NODE (to check whether

// this node became unbalanced)

int balance = heightDifferenceRecursive(root);

// If unbalanced, there are 4 cases

// Left Left Case

if (balance > 1 && heightDifferenceRecursive(root->left) >= 0)

return leftleft\_rotationRecursive(root);

// Left Right Case

if (balance > 1 && heightDifferenceRecursive(root->left) < 0)

{

return leftright\_rotationRecursive(root);

}

// Right Right Case

if (balance < -1 && heightDifferenceRecursive(root->right) <= 0)

return rightright\_rotationRecursive(root);

// Right Left Case

if (balance < -1 && heightDifferenceRecursive(root->right) > 0)

{

return rightleft\_rotationRecursive(root);

}

return root;

}

//displayRecursive AVL Tree

void avlTree::displayRecursive(avl\_Node \*ptr, int level)

{

int i;

if (ptr!=NULL)

{

displayRecursive(ptr->right, level + 1);

printf("\n");

if (ptr == root)

cout<<"Root -> ";

for (i = 0; i < level && ptr != root; i++)

cout<<" ";

cout<<ptr->data;

displayRecursive(ptr->left, level + 1);

}

}

//inOrderRecursive Traversal of AVL Tree

void avlTree::inOrderRecursive(avl\_Node \*tree)

{

if (tree == NULL)

return;

inOrderRecursive (tree->left);

cout<<tree->data<<" ";

inOrderRecursive (tree->right);

}

// preOrderRecursive Traversal of AVL Tree

void avlTree::preOrderRecursive(avl\_Node \*tree)

{

if (tree == NULL)

return;

cout<<tree->data<<" ";

preOrderRecursive (tree->left);

preOrderRecursive (tree->right);

}

// postOrderRecursive Traversal of AVL Tree

void avlTree::postOrderRecursive(avl\_Node \*tree)

{

if (tree == NULL)

return;

postOrderRecursive ( tree ->left );

postOrderRecursive ( tree ->right );

cout<<tree->data<<" ";

}

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

avlTree avlt1 = avlTree();

avlt1.insert(5);

avlt1.insert(4);

avlt1.insert(3);

avlt1.insert(2);

avlt1.insert(1);

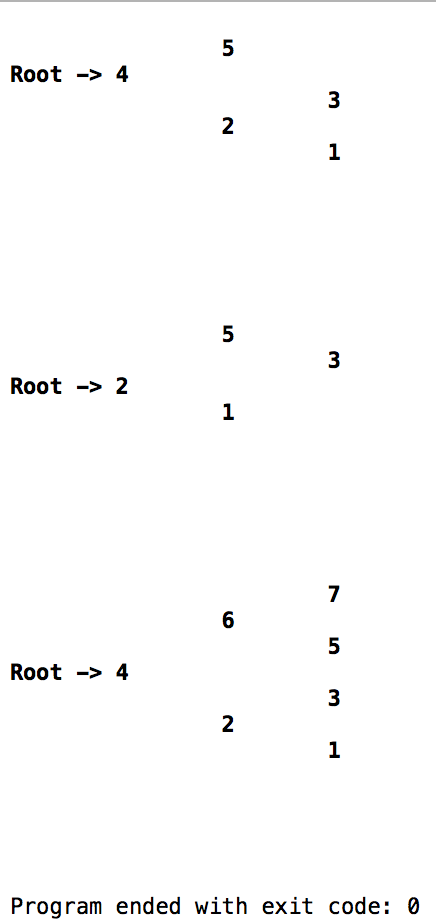
avlt1.display();

avlt1.remove(4);

avlt1.display();

list<int> myList;

myList.push\_back(1);

 myList.push\_back(2);

myList.push\_back(3);

myList.push\_back(4);

myList.push\_back(5);

myList.push\_back(6);

myList.push\_back(7);

avlTree avl2=avlTree(myList);

avl2.display();

}

// Yousef Zoumot

// main.cpp

// Coen70HW7.2 Chapter 10 Problem 12

//

// Created by Yousef Zoumot on 3/13/16.

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

//

#include<iostream>

#include<cstdio>

#include<sstream>

#include<algorithm>

#include <list>

#include <vector>

#include <string>

#include <cstring>

#include "string.h"

using namespace std;

class Book

{

public:

struct BookNode

{

BookNode(string nam, string auth, int isbn, string dat);

string name;

string author;

int iSBN;

string date;

struct BookNode \*left;

struct BookNode \*right;

};

Book()

{

root = NULL;

}

Book(list<int>);

int height(){return heightRecursive(root);};

void insert(BookNode source){root=insertRecursive(root ,source);};

void display();

private:

BookNode \*root;

int heightRecursive(BookNode \*);

int heightDifferenceRecursive(BookNode \*);

BookNode \*rightright\_rotationRecursive(BookNode \*);

BookNode \*leftleft\_rotationRecursive(BookNode \*);

BookNode \*leftright\_rotationRecursive(BookNode \*);

BookNode \*rightleft\_rotationRecursive(BookNode \*);

BookNode\* balanceRecursive(BookNode \*);

BookNode\* insertRecursive(BookNode \*, BookNode source);

BookNode\* removeRecursive(BookNode \*, int );

void displayRecursive(BookNode \*, int);

BookNode\* minValueNode(BookNode\* node);

};

bool lowercase(char c1, char c2){

return tolower(c1) < tolower(c2);

}

Book::BookNode::BookNode(string nam, string auth, int isbn, string dat){

name=nam;

author=auth;

iSBN=isbn;

date=dat;

}

void Book:: display(){

if(root ==NULL)

cout<<"This AVL Tree is empty"<< "\n";

else{

displayRecursive(root, 1);

}

cout<<"\n";

cout<<"\n";

cout<<"\n";

cout<<"\n";

cout<<"\n";

cout<<"\n";

}

void Book:: preOrder(){

preOrderRecursive(root);

}

void Book:: inOrder(){

inOrderRecursive(root);

}

void Book:: postOrder(){

postOrderRecursive(root);

}

Book::BookNode\* Book:: minValueNode(BookNode\* node)

{

BookNode\* current = node;

/\* loop down to find the leftmost leaf \*/

while (current->left != NULL)

current = current->left;

return current;

}

//\* Height of AVL Tree

int Book::heightRecursive(BookNode \*temp)

{

int h = 0;

if (temp != NULL)

{

int l\_height = heightRecursive (temp->left);

int r\_height = heightRecursive (temp->right);

int max\_height = max (l\_height, r\_height);

h = max\_height + 1;

}

return h;

}

// \* Height Difference

int Book::heightDifferenceRecursive(BookNode \*temp)

{

int l\_height = heightRecursive (temp->left);

int r\_height = heightRecursive (temp->right);

int b\_factor= l\_height - r\_height;

return b\_factor;

}

// Right- Right rotationRecursive

Book::BookNode \*Book::rightright\_rotationRecursive(BookNode \*parent)

{

BookNode \*temp;

temp = parent->right;

parent->right = temp->left;

temp->left = parent;

return temp;

}

// Left- Left rotationRecursive

Book::BookNode \*Book::leftleft\_rotationRecursive(BookNode \*parent)

{

BookNode \*temp;

temp = parent->left;

parent->left = temp->right;

temp->right = parent;

return temp;

}

// Left - Right rotationRecursive

Book::BookNode \*Book::leftright\_rotationRecursive(BookNode \*parent)

{

BookNode \*temp;

temp = parent->left;

parent->left = rightright\_rotationRecursive (temp);

return leftleft\_rotationRecursive (parent);

}

// Right- Left rotationRecursive

Book::BookNode \*Book::rightleft\_rotationRecursive(BookNode \*parent)

{

BookNode \*temp;

temp = parent->right;

parent->right = leftleft\_rotationRecursive (temp);

return rightright\_rotationRecursive (parent);

}

// Balancing AVL Tree

Book::BookNode \*Book::balanceRecursive(BookNode \*temp)

{

int bal\_factor = heightDifferenceRecursive (temp);

if (bal\_factor > 1)

{

if (heightDifferenceRecursive (temp->left) > 0)

temp = leftleft\_rotationRecursive (temp);

else

temp = leftright\_rotationRecursive (temp);

}

else if (bal\_factor < -1)

{

if (heightDifferenceRecursive (temp->right) > 0)

temp = rightleft\_rotationRecursive (temp);

else

temp = rightright\_rotationRecursive (temp);

}

return temp;

}

//insertRecursive Element into the tree

Book::BookNode \*Book::insertRecursive(BookNode \*root, BookNode source)

{

if (root == NULL)

{

root = new BookNode(source.name, source.author, source.iSBN, source.date);

root->left = NULL;

root->right = NULL;

return root;

}

string temp1=source.author;

string temp2=root->author;

for(int i=0; i<temp1.size(); i++){

temp1[i]=tolower(temp1[i]);

}

for(int i=0; i<temp2.size(); i++){

temp2[i]=tolower(temp2[i]);

}

if( temp1 < temp2)

{

root->left = insertRecursive(root->left, source);

root = balanceRecursive (root);

}

else if (temp1 >= temp2)

{

root->right = insertRecursive(root->right, source);

root = balanceRecursive (root);

}

return root;

}

//displayRecursive AVL Tree

void Book::displayRecursive(BookNode \*ptr, int level)

{

int i;

if (ptr!=NULL)

{

displayRecursive(ptr->right, level + 1);

printf("\n");

if (ptr == root)

cout<<"Root -> ";

for (i = 0; i < level && ptr != root; i++)

cout<<" ";

cout<<ptr->author;

displayRecursive(ptr->left, level + 1);

}

}

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

Book::BookNode \*b1= new Book::BookNode("name", "author1", 1234, "3/6/2001");

Book::BookNode \*b2= new Book::BookNode("name", "author2", 123456, "3/6/2001");

Book::BookNode \*b3= new Book::BookNode("name", "author3", 12345, "3/6/2001");

Book::BookNode \*b4= new Book::BookNode("name", "author4", 12346, "3/6/2001");

Book::BookNode \*b5= new Book::BookNode("name", "author5", 12356, "3/6/2001");

Book::BookNode \*b6= new Book::BookNode("name", "author6", 12456, "3/6/2001");

Book::BookNode \*b7= new Book::BookNode("name", "author7", 13456, "3/6/2001");

Book::BookNode \*b8= new Book::BookNode("name", "author4", 23456, "3/6/2001");

Book avlt1 = Book();

avlt1.insert(\*b1);

avlt1.insert(\*b2);

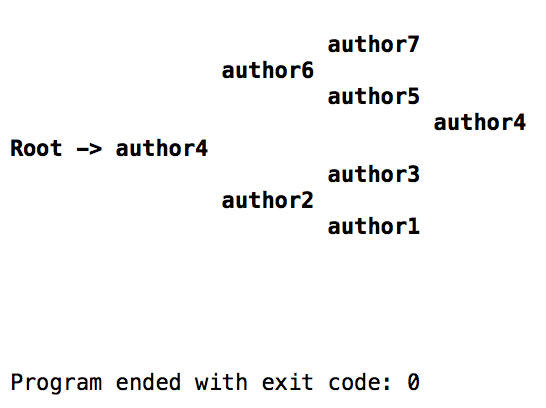
avlt1.insert(\*b3);

avlt1.insert(\*b4);

avlt1.insert(\*b5);

avlt1.insert(\*b6);

avlt1.insert(\*b7);

 avlt1.insert(\*b8);

avlt1.display();

}