// 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);

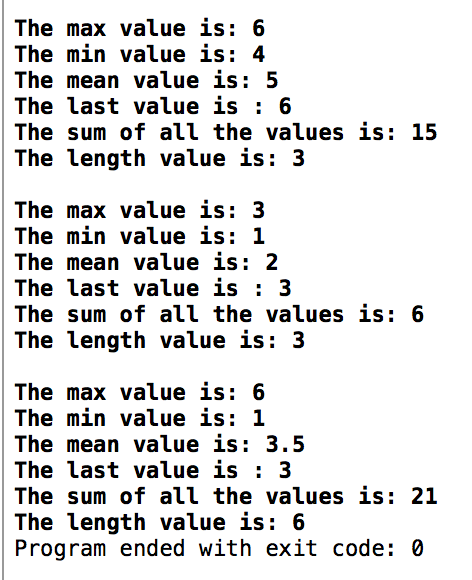
s2.printValues();

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);

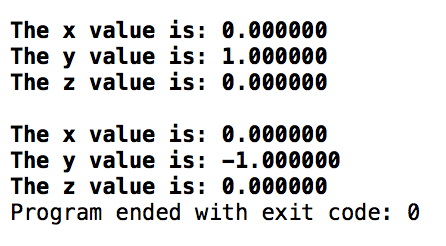
p.shiftZ(1);

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;

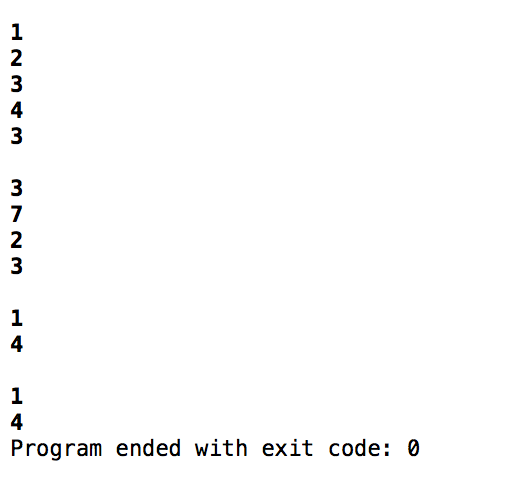
c=b-b2;

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];

}

