Alessandra Schiavi Paz and Muhammed Umar Khan

**Principles of Software Design**

**B02**

**ENSF 480**

Lab 2

Instructor: M. Moussavi

Sept 23, 2024

# Exercise A

## dictionaryList.cpp

/\*

\* dictionaryList.cpp

\* lab 2 Exercie A

\* Completed by: Alessandra Schiavi and Muhammed Umar Khan

\* Submission Date: Sept 23, 2024

\*/

#include <assert.h>

#include <iostream>

#include <stdlib.h>

#include "dictionaryList.h"

#include "mystring\_B.h"

using namespace std;

Node::Node(const Key& keyA, const Datum& datumA, Node \*nextA)

: keyM(keyA), datumM(datumA), nextM(nextA)

{

}

DictionaryList::DictionaryList()

: sizeM(0), headM(0), cursorM(0)

{

}

DictionaryList::DictionaryList(const DictionaryList& source)

{

copy(source);

}

DictionaryList& DictionaryList::operator =(const DictionaryList& rhs)

{

if (this != &rhs) {

destroy();

copy(rhs);

}

return \*this;

}

DictionaryList::~DictionaryList()

{

destroy();

}

int DictionaryList::size() const

{

return sizeM;

}

int DictionaryList::cursor\_ok() const

{

return cursorM != 0;

}

const Key& DictionaryList::cursor\_key() const

{

assert(cursor\_ok());

return cursorM->keyM;

}

const Datum& DictionaryList::cursor\_datum() const

{

assert(cursor\_ok());

return cursorM->datumM;

}

void DictionaryList::insert(const int& keyA, const Mystring& datumA)

{

// Add new node at head?

if (headM == 0 || keyA < headM->keyM) {

headM = new Node(keyA, datumA, headM);

sizeM++;

}

// Overwrite datum at head?

else if (keyA == headM->keyM)

headM->datumM = datumA;

// Have to search ...

else {

//POINT ONE

// if key is found in list, just overwrite data;

for (Node \*p = headM; p !=0; p = p->nextM)

{

if(keyA == p->keyM)

{

p->datumM = datumA;

return;

}

}

//OK, find place to insert new node ...

Node \*p = headM ->nextM;

Node \*prev = headM;

while(p !=0 && keyA >p->keyM)

{

prev = p;

p = p->nextM;

}

prev->nextM = new Node(keyA, datumA, p);

sizeM++;

}

cursorM = NULL;

}

Datum& DictionaryList::operator[](int index)

{// it recieves a dictionnaryList

if (index<0 || index >= sizeM){

cout<< "index is out of bounds"<<endl;

}

cursorM = headM;

i=0;

while (cursorM!=0 && i< index){

cursorM = cursorM->nextM;

i++;

}

return datumM;

}

ostream& operator <<(ostream& os, const DictionaryList &x){

x.cursorM = x.headM; // initialize the cursor to the headM

if(x.cursorM == nullptr){

os<< "The dictionary list is empty."<<endl;

return os;

}

while (dictList.cursorM != nullptr) {

os << x.cursorM->keyM<<" "<<x.cursorM->datumM.charsM; // Output the charsM

x.cursorM = dictList.cursorM->next; // Move to the next node

// Formatting: Add a newline after each entry

if (dictList.cursorM != nullptr) {

os << '\n'; // Add newline between entries

}

}

return os; // Return the output stream

}

void DictionaryList::remove(const int& keyA)

{

if (headM == 0 || keyA < headM -> keyM)

return;

Node \*doomed\_node = 0;

if (keyA == headM-> keyM) {

doomed\_node = headM;

headM = headM->nextM;

// POINT TWO

}

else {

Node \*before = headM;

Node \*maybe\_doomed = headM->nextM;

while(maybe\_doomed != 0 && keyA > maybe\_doomed-> keyM) {

before = maybe\_doomed;

maybe\_doomed = maybe\_doomed->nextM;

}

if (maybe\_doomed != 0 && maybe\_doomed->keyM == keyA) {

doomed\_node = maybe\_doomed;

before->nextM = maybe\_doomed->nextM;

}

}

if(doomed\_node == cursorM)

cursorM = 0;

delete doomed\_node; // Does nothing if doomed\_node == 0.

sizeM--;

}

void DictionaryList::go\_to\_first()

{

cursorM = headM;

}

void DictionaryList::step\_fwd()

{

assert(cursor\_ok());

cursorM = cursorM->nextM;

}

void DictionaryList::make\_empty()

{

destroy();

sizeM = 0;

cursorM = 0;

}

// The following function are supposed to be completed by the stuents, as part

// of the exercise B part II. the given fucntion are in fact place-holders for

// find, destroy and copy, in order to allow successful linking when you're

// testing insert and remove. Replace them with the definitions that work.

void DictionaryList::find(const Key& keyA)

{

cout << "\nDon't know how to find " << keyA << " (or any other key).\n";

cout << "... so exit is being called.\n";

exit(1);

}

void DictionaryList::destroy()

{

cout << "\nWARNING: DictionaryList::destroy() is abandoning nodes\n"

<< "when it should be deleting them!\n";

headM = 0;

}

void DictionaryList::copy(const DictionaryList& source)

{

cout << "\nDictionaryList::copy is not implemented properly,\n"

<< "so the program is calling exit.\n";

exit(1);

}

## DictionaryList.h

// dictionaryList.h

/\*

\* lab 2 Exercie A

\* Completed by: Alessandra Schiavi and Muhammed Umar Khan

\* Submission Date: Sept 23, 2024

\*/

#ifndef DICTIONARY\_H

#define DICTIONARY\_H

#include <iostream>

using namespace std;

// class DictionaryList: GENERAL CONCEPTS

//

// key/datum pairs are ordered. The first pair is the pair with

// the lowest key, the second pair is the pair with the second

// lowest key, and so on. This implies that you must be able to

// compare two keys with the < operator.

//

// Each DictionaryList object has a "cursor" that is either attached

// to a particular key/datum pair or is in an "off-list" state, not

// attached to any key/datum pair. If a DictionaryList is empty, the

// cursor is automatically in the "off-list" state.

#include "mystring\_B.h"

// Edit these typedefs to change the key or datum types, if necessary.

typedef int Key;

typedef Mystring Datum;

// THE NODE TYPE

// In this exercise the node type is a class, that has a ctor.

// Data members of Node are private, and class DictionaryList

// is declared as a friend. For details on the friend keyword refer to your

// lecture notes.

class Node {

friend class DictionaryList;

private:

Key keyM;

Datum datumM;

Node \*nextM;

// This ctor should be convenient in insert and copy operations.

Node(const Key& keyA, const Datum& datumA, Node \*nextA);

};

class DictionaryList {

public:

DictionaryList();

DictionaryList(const DictionaryList& source);

DictionaryList& operator =(const DictionaryList& rhs);

~DictionaryList();

int size() const;

// PROMISES: Returns number of keys in the table.

int cursor\_ok() const;

// PROMISES:

// Returns 1 if the cursor is attached to a key/datum pair,

// and 0 if the cursor is in the off-list state.

const Key& cursor\_key() const;

// REQUIRES: cursor\_ok()

// PROMISES: Returns key of key/datum pair to which cursor is attached.

const Datum& cursor\_datum() const;

// REQUIRES: cursor\_ok()

// PROMISES: Returns datum of key/datum pair to which cursor is attached.

void insert(const Key& keyA, const Datum& datumA);

// PROMISES:

// If keyA matches a key in the table, the datum for that

// key is set equal to datumA.

// If keyA does not match an existing key, keyA and datumM are

// used to create a new key/datum pair in the table.

// In either case, the cursor goes to the off-list state.

void remove(const Key& keyA);

// PROMISES:

// If keyA matches a key in the table, the corresponding

// key/datum pair is removed from the table.

// If keyA does not match an existing key, the table is unchanged.

// In either case, the cursor goes to the off-list state.

Datum& operator[](int index);

//receives a dictionaryList object and returns the datumM which contains charsM and lenghtM

ostream& operator <<(ostream& os, const DictionaryList &x);

//receives a dictionnary and os

// returns os containing the values of keyM and charsM of the dictionnary

// has a newline btween entries

void find(const Key& keyA);

// PROMISES:

// If keyA matches a key in the table, the cursor is attached

// to the corresponding key/datum pair.

// If keyA does not match an existing key, the cursor is put in

// the off-list state.

void go\_to\_first();

// PROMISES: If size() > 0, cursor is moved to the first key/datum pair

// in the table.

void step\_fwd();

// REQUIRES: cursor\_ok()

// PROMISES:

// If cursor is at the last key/datum pair in the list, cursor

// goes to the off-list state.

// Otherwise the cursor moves forward from one pair to the next.

void make\_empty();

// PROMISES: size() == 0.

private:

int sizeM;

Node \*headM;

Node \*cursorM;

void destroy();

// Deallocate all nodes, set headM to zero.

void copy(const DictionaryList& source);

// Establishes \*this as a copy of source. Cursor of \*this will

// point to the twin of whatever the source's cursor points to.

};

#endif

## Mystring\_B.h

/\* File: mystring\_B.h

\* lab 2 Exercie A

\* Completed by: Alessandra Schiavi and Muhammed Umar Khan

\* Submission Date: Sept 23, 2024

\*/

// ENSF 480 - Lab 2 - Exercise A

#include <iostream>

#include <string>

using namespace std;

#ifndef MYSTRING\_H

#define MYSTRING\_H

class Mystring {

public:

Mystring();

// PROMISES: Empty string object is created.

Mystring(int n);

// PROMISES: Creates an empty string with a total capacity of n.

// In other words, dynamically allocates n elements for

// charsM,sets the lengthM to zero, and fills the first

// element of charsM with '\0'.

Mystring(const char \*s);

// REQUIRES: s points to first char of a built-in string.

// REQUIRES: Mystring object is created by copying chars from s.

~Mystring(); // destructor

Mystring(const Mystring& source); // copy constructor

Mystring& operator =(const Mystring& rhs); // assignment operator

// REQUIRES: rhs is reference to a Mystring as a source

// PROMISES: to make this-object (object that this is pointing to, as a copy

// of rhs.

bool operator >= (const Datum& x);

bool operator <= (const Datum& x);

bool operator > (const Datum& x);

bool operator < (const Datum& x);

char& operator [](int index);// receives datum value

ostream& operator <<(ostream& os, const Datum& x);

int length() const;

// PROMISES: Return value is number of chars in charsM.

char get\_char(int pos) const;

// REQUIRES: pos >= 0 && pos < length()

// PROMISES:

// Return value is char at position pos.

// (The first char in the charsM is at position 0.)

const char \* c\_str() const;

// PROMISES:

// Return value points to first char in built-in string

// containing the chars of the string object.

void set\_char(int pos, char c);

// REQUIRES: pos >= 0 && pos < length(), c != '\0'

// PROMISES: Character at position pos is set equal to c.

Mystring& append(const Mystring& other);

// PROMISES: extends the size of charsM to allow concatenate other.charsM to

// to the end of charsM. For example if charsM points to "ABC", and

// other.charsM points to XYZ, extends charsM to "ABCXYZ".

//

void set\_str(char\* s);

// REQUIRES: s is a valid C++ string of characters (a built-in string)

// PROMISES:copys s into charsM, if the length of s is less than or equal lengthM.

// Othrewise, extends the size of the charsM to s.lengthM+1, and copies

// s into the charsM.

int isEqual (const Mystring& s)const;

// REQUIRES: s refers to an object of class Mystring

// PROMISES: retruns true if charsM equal s.charsM.

private:

int lengthM; // the string length - number of characters excluding \0

char\* charsM; // a pointer to the beginning of an array of characters, allocated dynamically.

void memory\_check(char\* s);

// PROMISES: if s points to NULL terminates the program.

};

#endif

## Mystring\_B.cpp

/\*

\* mystring\_B.cpp

\* lab 2 Exercie A

\* Completed by: Alessandra Schiavi and Muhammed Umar Khan

\* Submission Date: Sept 23, 2024

\*/

// ENSF 480 - Lab 2 - Exercise A

#include "mystring\_B.h"

#include <string.h>

#include <iostream>

using namespace std;

Mystring::Mystring()

{

charsM = new char[1];

// make sure memory is allocated.

memory\_check(charsM);

charsM[0] = '\0';

lengthM = 0;

}

Mystring::Mystring(const char \*s)

: lengthM(strlen(s))

{

charsM = new char[lengthM + 1];

// make sure memory is allocated.

memory\_check(charsM);

strcpy(charsM, s);

}

Mystring::Mystring(int n)

: lengthM(0), charsM(new char[n])

{

// make sure memory is allocated.

memory\_check(charsM);

charsM[0] = '\0';

}

Mystring::Mystring(const Mystring& source):

lengthM(source.lengthM), charsM(new char[source.lengthM+1])

{

memory\_check(charsM);

strcpy (charsM, source.charsM);

}

Mystring::~Mystring()

{

delete [] charsM;

}

int Mystring::length() const

{

return lengthM;

}

char Mystring::get\_char(int pos) const

{

if(pos < 0 && pos >= length()){

cerr << "\nERROR: get\_char: the position is out of boundary." ;

}

return charsM[pos];

}

const char \* Mystring::c\_str() const

{

return charsM;

}

void Mystring::set\_char(int pos, char c)

{

if(pos < 0 && pos >= length()){

cerr << "\nset\_char: the position is out of boundary."

<< " Nothing was changed.";

return;

}

if (c != '\0'){

cerr << "\nset\_char: char c is empty."

<< " Nothing was changed.";

return;

}

charsM[pos] = c;

}

Mystring& Mystring::operator =(const Mystring& S)

{

if(this == &S)

return \*this;

delete [] charsM;

lengthM = (int)strlen(S.charsM);

charsM = new char [lengthM+1];

memory\_check(charsM);

strcpy(charsM,S.charsM);

return \*this;

}

bool Mystring::operator >= (const Datum& x)

{

int compare = strcmp(storageM, x.storageM);

bool result = (compare >= 0)? true : false;

return result;

}

bool Mystring::operator <= (const Datum& x)

{

int compare = strcmp(storageM, x.storageM);

bool result = (compare < 0)? true : false;

return result;

}

bool Mystring::operator < (const Datum& x)

{

int compare = strcmp(storageM, x.storageM);

bool result = (compare < 0)? true : false;

return result;

}

bool Mystring::operator > (const Datum& x)

{

int compare = strcmp(storageM, x.storageM);

bool result = (compare > 0)? true : false;

return result;

}

bool Mystring::operator == (const Datum& x)

{

int compare = strcmp(storageM, x.storageM);

bool result = (compare == 0)? true : false;

return result;

}

bool Mystring::operator != (const Datum& x)

{

int compare = strcmp(storageM, x.storageM);

bool result = (compare != 0)? true : false;

return result;

}

char& Mystring::operator[](int index)

{// it recieves a datum&

if (index<0||index >= lengthM){

cout<< "index is out of bounds"<<endl;

}

return charsM[index];

}

ostream& operator <<(ostream& os, const Datum& x){

os<<x.charsM;

return os;

}

Mystring& Mystring::append(const Mystring& other)

{

char \*tmp = new char [lengthM + other.lengthM + 1];

memory\_check(tmp);

lengthM+=other.lengthM;

strcpy(tmp, charsM);

strcat(tmp, other.charsM);

delete []charsM;

charsM = tmp;

return \*this;

}

void Mystring::set\_str(char\* s)

{

delete []charsM;

lengthM = (int)strlen(s);

charsM=new char[lengthM+1];

memory\_check(charsM);

strcpy(charsM, s);

}

int Mystring::isEqual (const Mystring& s)const

{

return (strcmp(charsM, s.charsM)== 0);

}

void Mystring::memory\_check(char\* s)

{

if(s == 0)

{

cerr <<"Memory not available.";

exit(1);

}

}

# Exercise B

## Point.cpp

/\*

\* Point.cpp

\* lab 2 Exercie B

\* Completed by: Alessandra Schiavi and Muhammed Umar Khan

\* Submission Date: Sept 23, 2024

\*/

#include <iostream>

#include <cmath>

#include "Point.h"

using namespace std;

int Point::count = 1001;

Point::Point(double xVal, double yVal) : x(xVal), y(yVal) {

id = count++;

}

Point::~Point() {}

void Point::setX(double xVal) { x = xVal; }

void Point::setY(double yVal) { y = yVal; }

int Point::getId() const { return id; }

void Point::display() const {

cout << "X-coordinate: " << x << endl;

cout << "Y-coordinate: " << y << endl;

}

double Point::distance(const Point& other) const {

return sqrt(pow(x - other.x, 2) + pow(y - other.y, 2));

}

double Point::distance(const Point& p1, const Point& p2) {

return sqrt(pow(p1.x - p2.x, 2) + pow(p1.y - p2.y, 2));

}

int Point::counter() {

return count - 1001; // Return the total number of objects created

}

## Point.h

/\*

\* Point.h

\* lab 2 Exercie B

\* Completed by: Alessandra Schiavi and Muhammed Umar Khan

\* Submission Date: Sept 23, 2024

\*/

#ifndef POINT\_H

#define POINT\_H

class Point{

private:

double x,y;

int id;

static int count;// to track the number of points and auto-increment IDs

public:

Point(double xVal, double yVal);

~Point();

double getX() const { return x; }

double getY() const { return y; }

void setX(double xVal);

void setY(double yVal);

int getId() const;

void display() const;

double distance(const Point& other) const;

static double distance(const Point& p1, const Point& p2);

static int counter(); // Return count of objects

}

## Shape.h

/\*

\* Point.h

\* lab 2 Exercie B

\* Completed by: Alessandra Schiavi and Muhammed Umar Khan

\* Submission Date: Sept 23, 2024

\*/

#ifndef POINT\_H

#define POINT\_H

class Point{

private:

double x,y;

int id;

static int count;// to track the number of points and auto-increment IDs

public:

Point(double xVal, double yVal);

~Point();

double getX() const { return x; }

double getY() const { return y; }

void setX(double xVal);

void setY(double yVal);

int getId() const;

void display() const;

double distance(const Point& other) const;

static double distance(const Point& p1, const Point& p2);

static int counter(); // Return count of objects

}

## Shape.cpp

/\*

\* Square.h

\* lab 2 Exercie B

\* Completed by: Alessandra Schiavi and Muhammed Umar Khan

\* Submission Date: Sept 23, 2024

\*/

#include <iostream>

/\*

\* Shape.cpp

\* lab 2 Exercie B

\* Completed by: Alessandra Schiavi and Muhammed Umar Khan

\* Submission Date: Sept 23, 2024

\*/

#include <string>

#include "shape.h"

#include "point.h"

#include <cmath>

using namespace std;

Shape::Shape(int x, int y, char\*s){

origin.set\_x(x);

origin.set\_y(y);

size\_t length =strlen(s);

shapeName= new char[length +1];

strcpy(shapeName, s)

}

Shape::~Shape()

{

delete[] shapeName;

}

char\* Shape::getName()const {return \*shapeName};

char\* Shape::get\_x()const {return origin.get\_x()};

char\* Shape::get\_y()const {return origin.get\_y()};

double Shape::distance(Shape &other){

double x\_squared = pow(other.origin.x, 2);

double y\_sqared = pow(other.origin.y, 2);

double sum = x\_squared + y\_sqared;

double distance = sqrt(sum);

return distance;

}

static double Shape:: distance(Shape& the\_shape, Shape& other){

}

virtual void Shape:: move(double dx, double dy){

origin.set\_x(origin.x + dx);

origin.set\_y(origin.y + dy);

}// virtual function

virtual void Shape::display() const{

std::cout << "Square Name: " << getName() << std::endl; // Access name using getName()

std::cout << "X-coordinate: " << getX() << std::endl; // Access x using getX()

std::cout << "Y-coordinate: " << getY() << std::endl; // Access y using getY()

}// virtual funciton

## Square.h

/\*

\* Square.h

\* lab 2 Exercie B

\* Completed by: Alessandra Schiavi and Muhammed Umar Khan

\* Submission Date: Sept 23, 2024

\*/

#ifndef SQUARE\_H

#define SQUARE\_H

#include <iostream>

#include <string>

#include "Shape.h"

class Square : public Shape {

protected:

double side; // Keep only the side variable

public:

Square(double side\_a, double x, double y, std::string name);

// Virtual methods that Rectangle will override

virtual double area() const;

virtual double perimeter() const;

// Getters and setters

double get\_side() const;

void set\_side(double side\_a);

virtual void display() const;

};

#endif

## Square.cpp

/\*

\* Square.cpp

\* lab 2 Exercie B

\* Completed by: Alessandra Schiavi and Muhammed Umar Khan

\* Submission Date: Sept 23, 2024

\*/

#include <iostream>

#include <string>

#include "shape.h"

#include "point.h"

#include "square.h"

using namespace std;

// Constructor for Square class

Square::Square(double side\_a, double x, double y, std::string name)

: Shape(name.c\_str(), x, y), side(side\_a) { } // Correctly initialize base class

// Area calculation

double Square::area() const {

return side \* side; // Calculate area

}

// Perimeter calculation

double Square::perimeter() const {

return 4 \* side; // Calculate perimeter

}

// Getter for side length

double Square::get\_side() const {

return side; // Return side length

}

// Setter for side length

void Square::set\_side(double side\_a) {

this->side = side\_a; // Set side length

}

// Display method

void Square::display() const {

std::cout << "Square Name: " << getName() << std::endl; // Access 'name' via getter

std::cout << "X-coordinate: " << getOrigin().getX() << std::endl; // Access 'x' via Point

std::cout << "Y-coordinate: " << getOrigin().getY() << std::endl; // Access 'y' via Point

std::cout << "Side: " << side << std::endl; // Display side length

std::cout << "Area: " << area() << std::endl; // Display area

std::cout << "Perimeter: " << perimeter() << std::endl; // Display perimeter

}

## Rectangle.h

/\*

\* rectangle.h

\* lab 2 Exercie B

\* Completed by: Alessandra Schiavi and Muhammed Umar Khan

\* Submission Date: Sept 23, 2024

\*/

// Rectangle.h

#ifndef RECTANGLE\_H

#define RECTANGLE\_H

#include "Square.h"

class Rectangle : public Square {

protected:

double side\_b;

public:

Rectangle(double x = 0, double y = 0, double side\_a = 1, double side\_b = 1, const std::string& name = "Rectangle");

void set\_side\_a(double side\_a); // Should return void

void set\_side\_b(double side\_b);

double get\_side\_a() const;

double get\_side\_b() const;

virtual double area() const;

virtual double perimeter() const;

virtual void display() const;

double side\_a; // Consider removing this if it's unnecessary

};

#endif

## Rectangle.cpp

/\*

\* rectangle.cpp

\* lab 2 Exercie B

\* Completed by: Alessandra Schiavi and Muhammed Umar Khan

\* Submission Date: Sept 23, 2024

\*/

#include "Rectangle.h"

#include <iostream>

Rectangle::Rectangle(double x, double y, double side\_a, double side\_b, const std::string& name)

: Square(side\_a, x, y, name), side\_b(side\_b) {}

// Correcting set\_side\_a to have a void return type

void Rectangle::set\_side\_a(double side\_a) {

this->side = side\_a; // Set the side of the square

}

// No need for a return value here

void Rectangle::set\_side\_b(double side\_b) {

this->side\_b = side\_b; // Set side\_b

}

double Rectangle::get\_side\_a() const {

return side; // Get the side of the square

}

double Rectangle::get\_side\_b() const {

return side\_b; // Return side\_b

}

double Rectangle::area() const {

return side \* side\_b; // Area calculation using both sides

}

double Rectangle::perimeter() const {

return 2 \* (side + side\_b); // Perimeter calculation

}

void Rectangle::display() const {

Square::display(); // Call the Square's display method

std::cout << "Rectangle sides: " << get\_side() << " and " << side\_b << "\nArea: " << area() << "\nPerimeter: " << perimeter() << std::endl;

}

## graphicsWorld.h

/\*

\* rectangle.h

\* lab 2 Exercie B

\* Completed by: Alessandra Schiavi and Muhammed Umar Khan

\* Submission Date: Sept 23, 2024

\*/

#ifndef GRAPHICSWORLD\_H

#define GRAPHICSWORLD\_H

class GraphicsWorld {

public:

void run();

};

#endif

## graphicsWorld.cpp

/\*

\* rectangle.h

\* lab 2 Exercie B

\* Completed by: Alessandra Schiavi and Muhammed Umar Khan

\* Submission Date: Sept 23, 2024

\*/

#include "graphicsWorld.h"

#include "Point.h"

#include "Square.h"

#include "Rectangle.h"

#include "Shape.h"

#include <iostream>

void GraphicsWorld::run() {

std::cout << "Program by Muhammed Umar Khan" << std::endl;

// Test for Point class:

Point m(6, 8);

Point n(6, 8);

n.setX(9);

std::cout << "\nExpected to display the distance between m and n is: 3";

std::cout << "\nThe distance between m and n is: " << m.distance(n);

std::cout << "\nExpected second version of the distance function also print: 3";

std::cout << "\nThe distance between m and n is again: " << Point::distance(m, n);

// Test for Square class:

std::cout << "\n\nTesting Functions in class Square:" << std::endl;

Square s(5, 7, 12, "SQUARE - S");

s.display();

// Test for Rectangle class:

std::cout << "\nTesting Functions in class Rectangle:" << std::endl;

Rectangle a(5, 7, 12, 15, "RECTANGLE A");

a.display();

Rectangle b(16, 7, 8, 9, "RECTANGLE B");

b.display();

double d = a.distance(b);

std::cout << "\nDistance between rectangle a, and b is: " << d << std::endl;

// Test copy constructor and assignment operator

Rectangle rec1 = a; // copy constructor

rec1.display();

std::cout << "\nTesting assignment operator in class Rectangle:" << std::endl;

Rectangle rec2(3, 4, 11, 7, "RECTANGLE rec2");

rec2.display();

rec2 = a; // assignment operator

a.set\_side\_b(200);

a.set\_side\_a(100);

std::cout << "\nExpected to display the following values for object rec2:" << std::endl;

std::cout << "Rectangle Name: RECTANGLE A\n"

<< "X-coordinate: 5\n"

<< "Y-coordinate: 7\n"

<< "Side a: 12\n"

<< "Side b: 15\n"

<< "Area: 180\n"

<< "Perimeter: 54\n";

rec2.display();

std::cout << "\nTesting copy constructor in class Rectangle:" << std::endl;

Rectangle rec3(a); // copy constructor

rec3.display();

a.set\_side\_b(300);

a.set\_side\_a(400);

std::cout << "\nExpected to display the following values for object rec3:" << std::endl;

std::cout << "Rectangle Name: RECTANGLE A\n"

<< "X-coordinate: 5\n"

<< "Y-coordinate: 7\n"

<< "Side a: 100\n"

<< "Side b: 200\n"

<< "Area: 20000\n"

<< "Perimeter: 600\n";

rec3.display();

// Test for polymorphism:

std::cout << "\nTesting array of pointers and polymorphism:" << std::endl;

Shape\* sh[4];

sh[0] = &s;

sh[1] = &b;

sh[2] = &rec1;

sh[3] = &rec3;

for (int i = 0; i < 4; ++i) {

sh[i]->display(); // Polymorphism in action

}

}