ENSF 480 Lab 2 Report

Cover Page

Names: Beau McCartney, Quentin Jennings

Course Name: Principles of Software Design

Lab Section: B02 (Dr. Moshirpour)

Course Code: ENSF 480 Fall 2021

Assignment Number: Lab 2

Submission Date and Time: 24/09/2021 - 12:00 p.m.

Preface - How we got the output

To paste the output into this report, we redirected the output of each exercise to a `.txt` file. For example, `./a.out > a_output.txt`. The contents of these .txt files was then copied and pasted into the report.

Exercise Solutions

Exercise A

Source code:

```

// lookuptable.cpp

// ENSF 480 - Lab 2 - Exercise A

// Completed by: Beau McCartney, Quentin jennings

#include <assert.h>
#include <cstddef>
#include <iostream>
#include <ostream>
#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)
{
}

char& Node::operator[](size_t index) const {
    assert(index >= 0 && index < datumM.length());
    return datumM[index];
}

ostream& operator<<(ostream& os, const Node& node) {
    os << node.datumM.c_str();
    return os;
}

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;

}

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)
{
    if (sizeM == 0) {
        cursorM = 0;
```

```
            return;
        }

        Node *p;
        for(p = headM; p != 0 && p -> keyM < keyA; p = p -> nextM)
            ;

        cursorM = p != 0 && p -> keyM == keyA ? p : 0;
    }


    void DictionaryList::destroy()
    {
      if (sizeM > 0) {
          for (Node *temp = headM->nextM; temp != 0; temp = temp -> nextM) {
              delete headM;
              headM = temp;
          }
          delete headM;

      }
      sizeM = 0;
      headM = 0;
      cursorM = 0;
    }


    void DictionaryList::copy(const DictionaryList& source)
    {
        sizeM = 0;
        headM = 0;

        Node *src = source.headM;

        while (src != 0)
        {
            const Key keyN = src->keyM;
            const Datum datumN = src->datumM;
            insert(keyN, datumN);

            src = src->nextM;
        }

        if (source.cursor_ok()) {
            Key cursorKey = source.cursor_key();
            find(cursorKey);
        }

        assert(sizeM == source.size());
    }

    Node& DictionaryList::operator[](const size_t index) const {
        assert(index >= 0 && index < size());
        Node *temp = headM;
        for (size_t i = 0; i < index; i++)
            temp = temp->nextM;

        return *temp;
    }

    ostream& operator<<(ostream& os, const DictionaryList& dictionaryList) {
        for (size_t i = 0; i < dictionaryList.size(); i++)
            os << dictionaryList[i] << endl;

        return os;
    }


    /*
     * File Name: mystring_B.cpp
     * Assignment: Lab 2 Exercise A
     * Completed By: Beau McCartney, Quentin Jennings
     * Submission Date: September 30th, 2021
     */

    // ENSF 480 - Lab 2 - Exercise A
    #include "mystring_B.h"
    #include <cassert>
    #include <cstddef>
    #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;
}

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::isNotEqual (const Mystring& s)const
{
  return (strcmp(charsM, s.charsM)!= 0);
}

int Mystring::isEqual (const Mystring& s)const
{
  return (strcmp(charsM, s.charsM)== 0);
}


int Mystring::isGreaterThan (const Mystring& s)const
{
  return (strcmp(charsM, s.charsM)> 0);
}

int Mystring::isLessThan (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);
    }
}

ostream& operator << (ostream& os, const Mystring& myString) {
    os << myString.charsM;
    return os;
}

bool operator==(const Mystring& lhs, const Mystring &rhs) {
    return lhs.isEqual(rhs);
}

bool operator!=(const Mystring& lhs, const Mystring &rhs) {
    return lhs.isNotEqual(rhs);
}

bool operator < (const Mystring& lhs, const Mystring &rhs) {
    return lhs.isLessThan(rhs);
}

bool operator > (const Mystring& lhs, const Mystring &rhs) {
    return lhs.isGreaterThan(rhs);
}

bool operator<=(const Mystring& lhs, const Mystring &rhs) {
    return !lhs.isGreaterThan(rhs);
}

bool operator>=(const Mystring& lhs, const Mystring &rhs) {
    return !lhs.isLessThan(rhs);
}

char& Mystring::operator[](const size_t index) const {
    assert(index >= 0 && index < length());
    return charsM[index];
}
```

Output:

```
Printing list just after its creation ...
  List is EMPTY.

Printing list after inserting 3 new keys ...
```

```
  8001  Dilbert
  8002  Alice
  8003  Wally

Printing list after removing two keys and inserting PointyHair ...
  8003  Wally
  8004  PointyHair

Printing list after changing data for one of the keys ...
  8003  Sam
  8004  PointyHair

Printing list after inserting 2 more keys ...
  8001  Allen
  8002  Peter
  8003  Sam
  8004  PointyHair
***----Finished dictionary tests---------------------------***

Printing list--keys should be 315, 319
  315  Shocks
  319  Randomness
Printing list--keys should be 315, 319, 335
  315  Shocks
  319  Randomness
  335  ParseErrors
Printing list--keys should be 315, 335
  315  Shocks
  335  ParseErrors
Printing list--keys should be 319, 335
  319  Randomness
  335  ParseErrors
Printing list--keys should be 315, 319, 335
  315  Shocks
  319  Randomness
  335  ParseErrors
***----Finished tests of copying----------------------***


Testig a few comparison and insertion operators.

Peter is greater than or equal Allen
Allen is less than Peter
Peter is not equal to Allen
Peter is greater than Allen
Peter is not less than Allen
Peter is not equal to Allen

Using square bracket [] to access elements of Mystring objects.
The socond element of Peter is: e
The socond element of Poter is: o

Using << to display key/datum pairs in a Dictionary list:
Allen
```

```
Peter
Sam
PointyHair


Using [] to display the datum only:
Allen
Peter
Sam
PointyHair


Using [] to display sequence of charaters in a datum:
A
l
l
e
n


***----Finished tests for overloading operators ----------***
```

Exercise B

Source code:

```
/*
 * File Name: graphicsworld.cpp
 * Assignment: Lab 2 Exercise A
 * Completed By: Beau McCartney, Quentin Jennings
 * Submission Date: September 30th, 2021
 */


#include "point.h"
#include "shape.h"
#include "square.h"
#include "rectangle.h"
#include "graphicsworld.h"
#include "iostream"
using namespace std;

//function modified from Lab 2 pdf
https://d2l.ucalgary.ca/d2l/le/content/399720/viewContent/4905854/View
void GraphicsWorld::run(){
#if 1               // Change 0 to 1 to test Point

Point m (6, 8);

Point n (6,8);
```

```
n.setX(9);
cout << "\nExpected to dispaly the distance between m and n is: 3";
cout << "\nThe distance between m and n is: " <<
m.distance(n);
cout << "\nExpected second version of the distance function also print: 3";
cout << "\nThe distance between m and n is again: "
           << Point::distance(m, n);
#endif             // end of block to test Point

#if 1               // Change 0 to 1 to test Square
    cout << "\n\nTesting Functions in class Square:" <<endl;
    Square s(5, 7, 12, "SQUARE - S");
    s.display();
#endif             // end of block to test Square

#if 1               // Change 0 to 1 to test Rectangle

cout << "\nTesting Functions in class Rectangle:\n";

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);
cout <<"Distance between square a, and b is: " << d << "\n" << endl;

Rectangle rec1 = a;

rec1.display();

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

Rectangle rec2 (3, 4, 11, 7, "RECTANGLE rec2");
rec2.display();
rec2 = a;
a.set_side_b(200);
a.set_side_a(100);

cout << "\nExpected to display the following values for objec rec2: " << endl;
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"
;
cout << "\nIf it doesn't there is a problem with your assignment operator.\n" <<
endl;
rec2.display();

cout << "\nTesting copy constructor in class Rectangle:" <<endl;
Rectangle rec3 (a);
```

```
rec3.display();
a.set_side_b(300);
a.set_side_a(400);
cout << "\nExpected to display the following values for objec rec2: " << endl;
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" ;
cout << "\nIf it doesn't there is a problem with your assignment operator.\n" <<
endl;
rec3.display();
#endif              // end of block to test Rectangle
#if 1               // Change 0 to 1 to test using array of pointer and
polymorphism
cout << "\nTesting array of pointers and polymorphism:" <<endl;
Shape* sh[4];
sh[0] = &s;
sh[1] = &b;
sh [2] = &rec1;
sh [3] = &rec3;
sh [0]->display();
sh [1]->display();
sh [2]->display();
sh [3]->display();

#endif             // end of block to test array of pointer and polymorphism
}

/*
 * File Name: graphicsworld.h
 * Assignment: Lab 2 Exercise A
 * Completed By: Beau McCartney, Quentin Jennings
 * Submission Date: September 30th, 2021
 */


#ifndef GRAPHICSWORLD_H
#define GRAPHICSWORLD_H

//testing function
class GraphicsWorld {
    public:
        static void run();
        // PROMISES: a main function testing our classes
};
#endif

/*
 * File Name: main.cpp
 * Assignment: Lab 2 Exercise A
 * Completed By: Beau McCartney, Quentin Jennings
 * Submission Date: September 30th, 2021
 */
#include "graphicsworld.h"
```

```cpp
int main (int argc,char *argv[])
{
    GraphicsWorld::run();
    return 0;
}

/*
 * File Name: point.cpp
 * Assignment: Lab 2 Exercise A
 * Completed By: Beau McCartney, Quentin Jennings
 * Submission Date: September 30th, 2021
 */

#include "point.h"

int Point::pointCount = 0; //initializes the static point count

void Point::display() const {
    cout << fixed << "X-coordinate: " << setw(8) << setprecision(2) << x << endl;
    cout << fixed << "Y-coordinate: " << setw(8) << setprecision(2) << y << endl;
}

double Point::distance(const Point& the_point, const Point& other) {
    double dx = the_point.getX() - other.getX();
    double dy = the_point.getY() - other.getY();

    return sqrt(dx * dx + dy * dy);
}

double Point::distance(const Point& other) const {
    return distance(*this, other);
}

/*
 * File Name: point.h
 * Assignment: Lab 2 Exercise A
 * Completed By: Beau McCartney, Quentin Jennings
 * Submission Date: September 30th, 2021
 */

#ifndef POINT_H
#define POINT_H

#include <iostream>
#include <iomanip>
#include <cmath>
using namespace std;

//Represents a 1D point with an x and y coordinate
//Comes with functions to calculate the distance between points
//and a counter of the total number of points
class Point {
    private:
        static int pointCount; //the current number of active points
        double x, y; //the x and y coordinates of the points
        int id; //each point object has a unique id, starting at 1001
        friend class Shape;
    public:
        // TODO: how to make pointCount start at 0?
        Point(double x, double y) : x(x), y(y) {
            pointCount++;
            id = pointCount + 1000;
        }
        // PROMISES: increments pointCount
        // PROMISES: sets id to 1000 + pointCount

        ~Point() { pointCount--; }
        // PROMISES: decrements the point count

        //getters and setters
        void setX(const double x) { this->x = x; }
        // PROMISES: sets x to the passed in value

        void setY(const double y) { this->y = y; }
        // PROMISES: sets y to the passed in value

        double getX() const { return x; }
        // PROMISES: returns x

        double getY() const { return y; }
        // PROMISES: returns y

        static int counter() { return pointCount; }
        // PROMISES: returns the total number of point objects in the program

        void display() const;
        // PROMISES: displays the x and y coord

        static double distance(const Point& the_point, const Point& other);
        //PROMISES: returns the distance between any 2 points (static)

        double distance(const Point& other) const;
        //PROMISES: returns the distance between this point and another
};
#endif

/*
 * File Name: rectangle.cpp
 * Assignment: Lab 2 Exercise A
 * Completed By: Beau McCartney, Quentin Jennings
 * Submission Date: September 30th, 2021
 */

#include "rectangle.h"

//constructor
Rectangle::Rectangle(const double x, const double y, const double side_a, const
double side_b,
    const char* shapeName) : Square(x, y, side_a, shapeName)
{
    this->side_b = side_b;
}

void Rectangle::display() const {
    cout << "Rectangle Name: " << getName() << endl;
    origin.display();
    cout << "Side a: " << side_a << endl;
    cout << "Side b: " << side_b << endl;
    cout << "Area: " << area() << endl;
    cout << "Perimeter: " << perimeter() << endl;
}

/*
 * File Name: rectangle.h
 * Assignment: Lab 2 Exercise A
 * Completed By: Beau McCartney, Quentin Jennings
 * Submission Date: September 30th, 2021
 */

#ifndef RECT_H
#define RECT_H

#include "square.h"
#include <iostream>
using namespace std;

//child of Square and grandchild of Shape, represents a Rectangle
//side_b defines the length of the second side (side_a and side_b)
class Rectangle : public Square {
    private:
        double side_b;

    public:
        //constructor + big 3
        Rectangle(const double x, const double y, const double side_a,
                const double side_b, const char* shapeName);
        // REQUIRES: shapeName points to the first character of a c-string
        // PROMISES: initializes rectangle with the passed in values

        double area() const { return side_a * side_b; }
        // PROMISES: returns area of rectangle

        double perimeter() const { return 2 * (side_a + side_b); }
        // PROMISES: returns perimeter of rectangle

        //getters/setters
        double get_side_b() const { return side_b; }
        // PROMISES: returns side b

        void set_side_b(double side_b) { this->side_b = side_b; }
        // PROMISES: sets side_b to the passed in value
        void display() const;
        // PROMISES: displays rectangle name, x/y coords, side a/b, area and
perimeter
};

#endif

/*
 * File Name: shape.cpp
 * Assignment: Lab 2 Exercise A
 * Completed By: Beau McCartney, Quentin Jennings
 * Submission Date: September 30th, 2021
 */

#include "shape.h"

//constructor
Shape::Shape(const Point origin, const char *shapeName) :
    origin(origin), shapeName(new char[strlen(shapeName) + 1])
{
    strcpy(this->shapeName, shapeName);
}

//assignment operator
Shape& Shape::operator=(const Shape& s) {
    if(this != &s) {
        origin = Point(s.getOrigin().getX(), s.getOrigin().getY());
        shapeName = new char[strlen(s.getName() + 1)];
        strcpy(shapeName, s.getName());
    }
    return *this;
}

//copy constructor
Shape::Shape(const Shape& s) :
    origin(Point(s.getOrigin().getX(), s.getOrigin().getY()))
{
    shapeName = new char[strlen(s.getName() + 1)];
    strcpy(shapeName, s.getName());
}

//displays the shape name and coordinates of its origin
void Shape::display() const {
    cout << "Shape Name: " << shapeName << endl;
    origin.display();
}

double Shape::distance(Shape& other) const {
    return Point::distance(this->origin, other.origin);
}

double Shape::distance(Shape& the_shape, Shape& other) {
    return Point::distance(the_shape.getOrigin(), other.getOrigin());
}
```

```
void Shape::move(double dx, double dy) {
    origin.x += dx;
    origin.y += dy;
}

/*
 * File Name: shape.h
 * Assignment: Lab 2 Exercise A
 * Completed By: Beau McCartney, Quentin Jennings
 * Submission Date: September 30th, 2021
 */

#ifndef SHAPE_H
#define SHAPE_H

#include <string.h>
#include "point.h"
using namespace std;

//represents a Shape and acts as the base class for Square/Rectangle
//has an origin point and a name and functions related to the shape
class Shape {
    protected:
        Point origin;
        char *shapeName; // dynamically allocated by the constructor

    public:
        //constructor + big 3
        Shape(const Point origin, const char *shapeName); //constructor
        // REQUIRES: shapeName points to the first character of a c-string
        // PROMISES: initializes members of shape to the passed in values

        ~Shape() { delete [] shapeName; }
        // PROMISES: frees shapeName

        Shape& operator=(const Shape& s); //assignment operator
        // REQUIRES: s is a reference to a shape object
        // PROMISES: make this object a copy of s, freeing memory as necessary,
freeing memory as necessary

        Shape(const Shape& s); //copy constructor
        // REQUIRES: s is a reference to a shape object
        // PROMISES: construct a copy of s, allocating memory as necessary

        const Point& getOrigin() const { return origin; }
        // PROMISES: returns an immutable reference to the origin of the shape

        char* getName() const { return shapeName; }
        // PROMISES: returns a pointer to the shapeName

        virtual void display() const;
        // PROMISES: displays the shape name and origin coordinates
        double distance(Shape& other) const;
        // PROMISES: returns the distance between this shape and another

        static double distance(Shape& the_shape, Shape& other);
        // PROMISES: returns the distance between any 2 shapes

        void move(double dx, double dy);
        // PROMISES: Sets shape's origin to the passed in coordinates
};
#endif

/*
 * File Name: square.cpp
 * Assignment: Lab 2 Exercise A
 * Completed By: Beau McCartney, Quentin Jennings
 * Submission Date: September 30th, 2021
 */

#include "square.h"

//constructor
Square::Square(const double x, const double y, const double side_a, const char*
shapeName)
    : Shape(Point(x,y), shapeName)
{
    this->side_a = side_a;
}

void Square::display() const {
    cout << "Square Name: " << getName() << endl;
    origin.display();
    cout << "Side a: " << side_a << endl;
    cout << "Area: " << area() << endl;
    cout << "Perimeter: " << perimeter() << endl;
}

/*
 * File Name: square.h
 * Assignment: Lab 2 Exercise A
 * Completed By: Beau McCartney, Quentin Jennings
 * Submission Date: September 30th, 2021
 */
#ifndef SQUARE_H
#define SQUARE_H

#include "shape.h"
#include <iostream>
using namespace std;

//child class of Shape, represents a Square
//has one side defined (squares have all sides equal)
class Square : public Shape {
    protected:
        double side_a;
    public:
        //constructor + big 3
        Square(const double x, const double y, const double side_a, const char*
shapeName);
        // REQUIRES: shapeName points to the first character of a c-string
        // PROMISES: initializes members of square to the passed in values

        //getters/setters
        double get_side_a() const { return side_a; }
        // PROMISES: returns side_a

        void set_side_a(double side_a) { this->side_a = side_a; }
        // PROMISES: sets side_a to passed in value

        virtual double area() const { return side_a * side_a; }
        // PROMISES: returns the area of the square

        virtual double perimeter() const { return 4 * side_a; }
        // PROMISES: returns the perimeter of the square

        virtual void display() const;
        // PROMISES: prints the square's name, x/y coords, side length, area and
perimeter
};
#endif
```

Output:

```


Expected to dispaly the distance between m and n is: 3
The distance between m and n is: 3
Expected second version of the distance function also print: 3
The distance between m and n is again: 3

Testing Functions in class Square:
Square Name: SQUARE - S
X-coordinate:     5.00
Y-coordinate:     7.00
Side a: 12.00
Area: 144.00
Perimeter: 48.00

Testing Functions in class Rectangle:
Rectangle Name: RECTANGLE A
X-coordinate:     5.00
Y-coordinate:     7.00
Side a: 12.00
Side b: 15.00
Area: 180.00
Perimeter: 54.00
Rectangle Name: RECTANGLE B
```

```
X-coordinate:    16.00
Y-coordinate:     7.00
Side a: 8.00
Side b: 9.00
Area: 72.00
Perimeter: 34.00
Distance between square a, and b is: 11.00

Rectangle Name: RECTANGLE A
X-coordinate:     5.00
Y-coordinate:     7.00
Side a: 12.00
Side b: 15.00
Area: 180.00
Perimeter: 54.00

Testing assignment operator in class Rectangle:
Rectangle Name: RECTANGLE rec2
X-coordinate:     3.00
Y-coordinate:     4.00
Side a: 11.00
Side b: 7.00
Area: 77.00
Perimeter: 36.00

Expected to display the following values for objec rec2:
Rectangle Name: RECTANGLE A
X-coordinate: 5
Y-coordinate: 7
Side a: 12
Side b: 15
Area: 180
Perimeter: 54

If it doesn't there is a problem with your assignment operator.

Rectangle Name: RECTANGLE A
X-coordinate:     5.00
Y-coordinate:     7.00
Side a: 12.00
Side b: 15.00
Area: 180.00
Perimeter: 54.00

Testing copy constructor in class Rectangle:
Rectangle Name: RECTANGLE A
X-coordinate:     5.00
Y-coordinate:     7.00
Side a: 100.00
Side b: 200.00
Area: 20000.00
Perimeter: 600.00

Expected to display the following values for objec rec2:
```

```
Rectangle Name: RECTANGLE A
X-coordinate: 5
Y-coordinate: 7
Side a: 100
Side b: 200
Area: 20000
Perimeter: 600

If it doesn't there is a problem with your assignment operator.

Rectangle Name: RECTANGLE A
X-coordinate:     5.00
Y-coordinate:     7.00
Side a: 100.00
Side b: 200.00
Area: 20000.00
Perimeter: 600.00

Testing array of pointers and polymorphism:
Square Name: SQUARE - S
X-coordinate:     5.00
Y-coordinate:     7.00
Side a: 12.00
Area: 144.00
Perimeter: 48.00
Rectangle Name: RECTANGLE B
X-coordinate:    16.00
Y-coordinate:     7.00
Side a: 8.00
Side b: 9.00
Area: 72.00
Perimeter: 34.00
Rectangle Name: RECTANGLE A
X-coordinate:     5.00
Y-coordinate:     7.00
Side a: 12.00
Side b: 15.00
Area: 180.00
Perimeter: 54.00
Rectangle Name: RECTANGLE A
X-coordinate:     5.00
Y-coordinate:     7.00
Side a: 100.00
Side b: 200.00
Area: 20000.00
Perimeter: 600.00
```