ENSF 480 - Lab 0 Solutions:

EXERCISE A:

```
//point.h

#ifndef POINT_H
#define POINT_H

class Point
{
public:
     Point(double a, double b);
     void display() const;
     double getx() const;
     void setx(double newvalue);
     double gety() const;
     void sety(double newvalue);
     double distance(const Point& other);
     static double distance (const Point& that, const Point& other);
    static int counter();
private:
    static int idCounter;
    int id;
    double x_coordinate;
     double y_coordinate;
};
#endif

//point.cpp

#include <assert.h>
#include <iostream>
#include <stdlib.h>
#include <math.h>
#include "point.h"

using namespace std;
int Point::idCounter = 1001;

Point::Point(double a, double b):x_coordinate(a), y_coordinate(b)
{
    id = idCounter++;
}

void Point::display() const
{
     cout << "Point ID: " << id << "\n";
    cout << "X-coordinate: " << x_coordinate << "\n";
     cout << "Y-coordinate: " << y_coordinate << "\n";
}

double Point::getx() const
{
     return x_coordinate;
}

void Point::setx(double newvalue)
{
     x_coordinate = newvalue;
}

double Point::gety() const
{
     return y_coordinate;
}

void Point::sety(double newvalue)
{
     y_coordinate = newvalue;
}

double Point::distance(const Point& other)
{
     double dist_x = other.x_coordinate - x_coordinate;
     double dist_y = other.y_coordinate - y_coordinate;

     return (sqrt(pow(dist_x, 2) + pow(dist_y, 2)));
}

double Point::distance (const Point& that, const Point& other)
{
     double dist_x = other.x_coordinate - that.x_coordinate;
     double dist_y = other.y_coordinate - that.y_coordinate;
```

```
     return (sqrt(pow(dist_x, 2) + pow(dist_y, 2)));
}

int Point::counter() {
    return idCounter;
}

//shape.h

#ifndef SHAPE_H
#define SHAPE_H

#include "point.h"

class Shape
{
public:
     Shape(double x_origin, double y_origin, const char* name);
     virtual ~Shape();
     Shape(const Shape& source);
     Shape&  operator=(const Shape& rhs);
     Point getOrigin() const;
     char* getName() const;
     virtual void display() const;
     double distance(const Shape &other);
     static double distance (const Shape& the_shape, const Shape& other);
     void move(double dx, double dy);
     virtual double area() const = 0;
     virtual double perimeter() const = 0;
protected:
     Point origin;
     char* shapeName;
};
#endif

//shape.cpp
#include <assert.h>
#include <iostream>
#include <stdlib.h>
#include <math.h>
#include "shape.h"
using namespace std;

Shape::Shape(double x_origin, double y_origin, const char* name):origin(x_origin,y_origin)
{
  shapeName = new char[strlen(name)+1];
  if(shapeName == NULL){
       cerr << "Memory not available...";
       exit(1);
     }

     origin.setx(x_origin);
     origin.sety(y_origin);
     strcpy(shapeName,name);
}

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

Shape::Shape(const Shape& source):origin(source.origin), shapeName(new
char[strlen(source.shapeName)+1])
{
     if(shapeName == NULL){
       cerr << "Memory not available...";
       exit(1);
     }
     strcpy(shapeName, source.shapeName);
}

Shape& Shape::operator=(const Shape& rhs)
{
     if(this==&rhs)
       return *this;
     delete [] shapeName;
     shapeName = new char[strlen(rhs.shapeName)+1];

     if(shapeName == NULL){
       cerr << "Memory not available...";
       exit(1);
     }
     strcpy(shapeName, rhs.shapeName);
     origin = rhs.origin;
     return *this;
}
```

```
Point Shape::getOrigin() const
{
     return origin;
}

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

void Shape::display() const
{
     cout << "\nShape name: " << shapeName << "\n";
     origin.display();
}

double Shape::distance(const Shape& other)
{
     return origin.distance(other.origin);
}

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

void Shape::move(double dx, double dy)
{
     origin.setx(origin.getx()+dx);
     origin.sety(origin.gety()+dy);
}

//rectangle.h
#ifndef RECTANGLE_H
#define RECTANGLE_H
#include "square.h"

class Rectangle: public Square
{
public:
     Rectangle(double x_origin, double y_origin, double newlength_a, double newlength_b, const
char* name);
     Rectangle(const Rectangle& source);
     ~Rectangle(){}
     Rectangle&  operator=(const Rectangle& rhs);
     void set_side_b(double newlength);
     double get_side_b() const;
     double area() const;
     double perimeter() const;
     void display() const;

protected:
     double side_b;
};
#endif

//rectangle.cpp
#include <assert.h>
#include <iostream>
#include <stdlib.h>
#include <math.h>
#include "rectangle.h"
using namespace std;

Rectangle::Rectangle(double x_origin, double y_origin, double newlength_a, double newlength_b,
const char* name):side_b(newlength_b), Square(x_origin, y_origin, newlength_a,
name),Shape(x_origin, y_origin, name)
{
     cout << getName();
}

Rectangle::Rectangle(const Rectangle& source):Square(source), Shape(source)
{
    delete [] shapeName;
    shapeName = new char[sizeof(source.shapeName)+1];
    strcpy(shapeName, source.shapeName);
    origin = source.origin;
    side_a = source.side_a;
    side_b = source.side_b;
}

Rectangle& Rectangle::operator=(const Rectangle& rhs)
{
     if(this==&rhs)
       return *this;
     delete [] shapeName;
     shapeName = new char[sizeof(rhs.shapeName)+1];
```

```
     strcpy(shapeName, rhs.shapeName);
     origin = rhs.origin;
     side_a = rhs.side_a;
     side_b = rhs.side_b;
     return *this;
}


void Rectangle::set_side_b(double newlength)
{
     side_b = newlength;
}

double Rectangle::get_side_b() const
{
     return side_b;
}

double Rectangle::area() const
{
     return side_a * side_b;
}

double Rectangle::perimeter() const
{
     return side_a * 2 + side_b * 2;
}

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

//circle.h
#ifndef CIRCLE_H
#define CIRCLE_H
#include "shape.h"

class Circle: virtual public Shape
{
public:
     Circle(double x_origin, double y_origin, double newradius, const char* name);
     Circle(const Circle& source);
     ~Circle() {}
     Circle&  operator=(const Circle& rhs);
     void set_radius(double newradius);
     double get_radius() const;
     double area() const;
     double perimeter() const;
     void display() const;
protected:
     double radius;
};
#endif

//circle.cpp
#include <assert.h>
#include <iostream>
#include <stdlib.h>
#include <math.h>
#include "circle.h"
using namespace std;

Circle::Circle(double x_origin, double y_origin, double newradius, const char*
name):radius(newradius), Shape(x_origin, y_origin, name)
{
}

Circle::Circle(const Circle& source):radius(source.radius), Shape(source)
{
}

Circle& Circle::operator=(const Circle& rhs)
{
     if(this==&rhs)
       return *this;
     delete [] shapeName;
     shapeName = new char[sizeof(rhs.getName())+1];

     if(shapeName == NULL){
       cerr << "Memory not available...";
       exit(1);
```

```
     }
     strcpy(shapeName, rhs.shapeName);
     origin = rhs.origin;
     return *this;
}


void Circle::set_radius(double newradius)
{
     radius = newradius;
}

double Circle::get_radius() const
{
     return radius;
}

double Circle::area() const
{
     return 3.14 * pow(radius, 2);
}

double Circle::perimeter() const
{
     return 2 * 3.14 * radius;
}

void Circle::display() const
{
     cout << "\nCircle Name: " << getName() << "\n";
     origin.display();
    cout << "Radius: "  <<  radius << endl;
}


//square.h
#ifndef SQUARE_H
#define SQUARE_H
#include "shape.h"

class Square: virtual public Shape
{
public:
     Square(double x_origin, double y_origin, double newlength, const char* name);
     ~Square(){}
     Square(const Square& source);
     Square&  operator=(const Square& rhs);
     void set_side_a(double newlength);
     double get_side_a() const;
     double area() const;
     double perimeter() const;
     void display() const;

protected:
     double side_a;
};
#endif









//square.cpp
#include <assert.h>
#include <iostream>
#include <stdlib.h>
#include <math.h>
#include "square.h"
using namespace std;

Square::Square(double x_origin, double y_origin, double newlength, const char*
name):side_a(newlength), Shape(x_origin, y_origin, name)
{

}

Square::Square(const Square& source):side_a(source.get_side_a()), Shape(source)
{
}
```

```
Square& Square::operator=(const Square& rhs)
{
     if(this==&rhs)
       return *this;
     delete [] shapeName;
     shapeName = new char[sizeof(rhs.getName())+1];
     strcpy(shapeName, rhs.getName());
     origin = rhs.getOrigin();
     side_a = rhs.get_side_a();
     return *this;
}

void Square::set_side_a(double newlength)
{
     side_a = newlength;
}

double Square::get_side_a() const
{
     return side_a;
}

double Square::area() const
{
     return side_a * side_a;
}

double Square::perimeter() const
{
     return 4 * side_a;
}

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


/Curvecut.h

#ifndef CurveCUT_H
#define CurveCUT_H
#include "rectangle.h"
#include "circle.h"

class CurveCut: public Rectangle, public Circle
{
public:
     CurveCut(double x_origin, double y_origin, double newlength_a, double newlength_b, double
newradius, const char* name);
     ~CurveCut(){}
     CurveCut(const CurveCut& source);
     CurveCut& operator=(const CurveCut& rhs);

     double area() const;
     double perimeter() const;
     void display() const;
     char* getName();
     double distance(Shape& other);
};
#endif


//Curvecut.cpp

#include <assert.h>
#include <iostream>
#include <stdlib.h>
#include <math.h>
#include "curvecut.h"

using namespace std;

CurveCut::CurveCut(double x_origin, double y_origin, double newlength_a, double newlength_b, double
newradius, const char* name):Circle(x_origin, y_origin, newradius, name), Rectangle(x_origin,
y_origin, newlength_a, newlength_b, name),Shape(x_origin,y_origin,name)
{
     assert(radius <= newlength_a);
}

CurveCut& CurveCut::operator=(const CurveCut& rhs)
{
     if(this==&rhs)
```

```
      return *this;
     delete [] shapeName;
     shapeName = new char[sizeof(rhs.shapeName)+1];
     strcpy(shapeName, rhs.shapeName);
     origin = rhs.origin;
     side_a = rhs.side_a;
     side_b = rhs.side_b;
     radius = rhs.radius;
     return *this;
}

CurveCut::CurveCut(const CurveCut& source): Shape(source), Circle(source), Rectangle(source)
{

}

double CurveCut::area() const
{
     return (this->Rectangle::area()) - ((this->Circle::area())/4);
}

double CurveCut::perimeter() const
{
     return (this->Rectangle::perimeter()) - (2 * radius) + ((this->Circle::perimeter())/4);
}


void CurveCut::display() const
{
     cout << "CurveCut Name: " << Rectangle::getName() << "\n";
     Rectangle::origin.display();
     cout << "Width: " << get_side_a() << "\n";
     cout << "Length: " << get_side_b() << "\n";
     cout << "Radius: " << get_radius() << "\n";
}

char* CurveCut::getName()
{
     return this->Circle::getName();
}

double CurveCut::distance(Shape& other)
{
     return this->Circle::distance(other);
}


//graphicsworld.h
#ifndef GRAPHICSWORLD_H
#define GRAPHICSWORLD_H
#include "shape.h"

class GraphicsWorld
{
public:
    void run();
};
#endif
```