MIDDLE EAST TECHNICAL UNIVERSITY

Department of
Computer Education and Instructional Technology

CEIT211: Programming Languages II

2015 Spring **Midterm Exam II** **100 minutes**

Surname	**Name**	**Student ID**	**Total**

Grading:

Question	1	2	3	4
Points				

Q1. (10 points) Consider the following class definitions:

```
class bClass
{
public:
   void setX(int a);
     //Postcondition: x = a;
   void print() const;

private:
   int x;
};

class dClass: public bClass
{
public:
   void setXY(int a, int b);
     //Postcondition: x = a; y = b;
   void print() const;

private:
   int y;
};
```

Complete the following to set the values of x and y:

Answer:

```
void dClass::setXY(int a, int b)
{
        bClass::setX(a);
        y = b;
}
```

Q2. (10 points) What is the output of the following program?

```
#include <iostream>

using namespace std;
class bClass
{
public:
   void print() const;
   bClass(int a = 0, int b = 0);
     //Postcondition: x = a; y = b;

private:
   int x;
   int y;
};

class dClass: public bClass
{
public:
   void print() const;
   dClass(int a = 0, int b = 0, int c =
0);
     //Postcondition: x = a; y = b; z =
c;

private:
   int z;
};
```

```
int main()
{
  bClass bObject(2, 3);
  dClass dObject(3, 5, 8);

  bObject.print();
  cout << endl;
  dObject.print();
  cout << endl;

  return 0 ;
}

void bClass::print() const
{
   cout << x << " " << y << endl;
}

bClass::bClass(int a, int b)
{
   x = a;
   y = b;
}

void dClass::print() const
{
   bClass:print();
   cout << " " << z << endl;
}

dClass::dClass(int a, int b, int c)
  : bClass(a, b)
{
  z = c;
}
```

Answer

2 3
3 5 8

Q3. (40 Points) Using the following OOP program and its implementation, derive the UML Diagram for the classes **shape**, **Rectangle** and **Circle** including the relations between them.

```
/* Header for Shape class (Shape.h) */
#ifndef SHAPE_H
#define SHAPE_H

#include <string>
using namespace std;

class Shape {
private:
   string color;   // Private data member

public:
   Shape(const string & color = "red");  // Constructor
   string getColor() const;           // Getter
   void setColor(const string & color);  // Setter
   // Virtual function, run subclass version if overridden
   virtual void print() const;
   // Pure virtual, to be implemented by subclass
   // You cannot create instance of Shape
   virtual double getArea() const = 0;
};

#endif


/* Implementation for Shape class (Shape.cpp) */
#include "Shape.h"
#include <iostream>

// Constructor
Shape::Shape(const string & color) {
   this->color = color;
}

// Getter
string Shape::getColor() const {
   return color;
}

// Setter
void Shape::setColor(const string & color) {
   this->color = color;
}

void Shape::print() const {
   std::cout << "Shape of color=" << color;
}
```

```
/* Header for Circle (Circle.h) */
#ifndef CIRCLE_H
#define CIRCLE_H

#include "Shape.h"

// The class Circle is a subclass of Shape
class Circle : public Shape {
private:
   int radius;   // Private data member

public:
   Circle(int radius = 1, const string & color = "red"); // Constructor
   int getRadius() const;               // Getter
   void setRadius(int radius);     // Setter
   void print() const;      // Override the virtual function
   double getArea() const;  // to implement virtual function
};

#endif
```

```
/* Implementation for Circle (Circle.cpp) */
#include "Circle.h"
#include <iostream>
#define PI 3.14159265

// Constructor
Circle::Circle(int radius, const string & color)
   : Shape(color), radius(radius) { }

// Getters
int Circle::getRadius() const {
   return radius;
}

// Setters
void Circle::setRadius(int radius) {
   this->radius = radius;
}

void Circle::print() const {
   std::cout << "Circle radius=" << radius << ", subclass of ";
   Shape::print();
}

// Implement virtual function inherited for superclass Shape
double Circle::getArea() const {
   return radius * radius * PI;
}
```

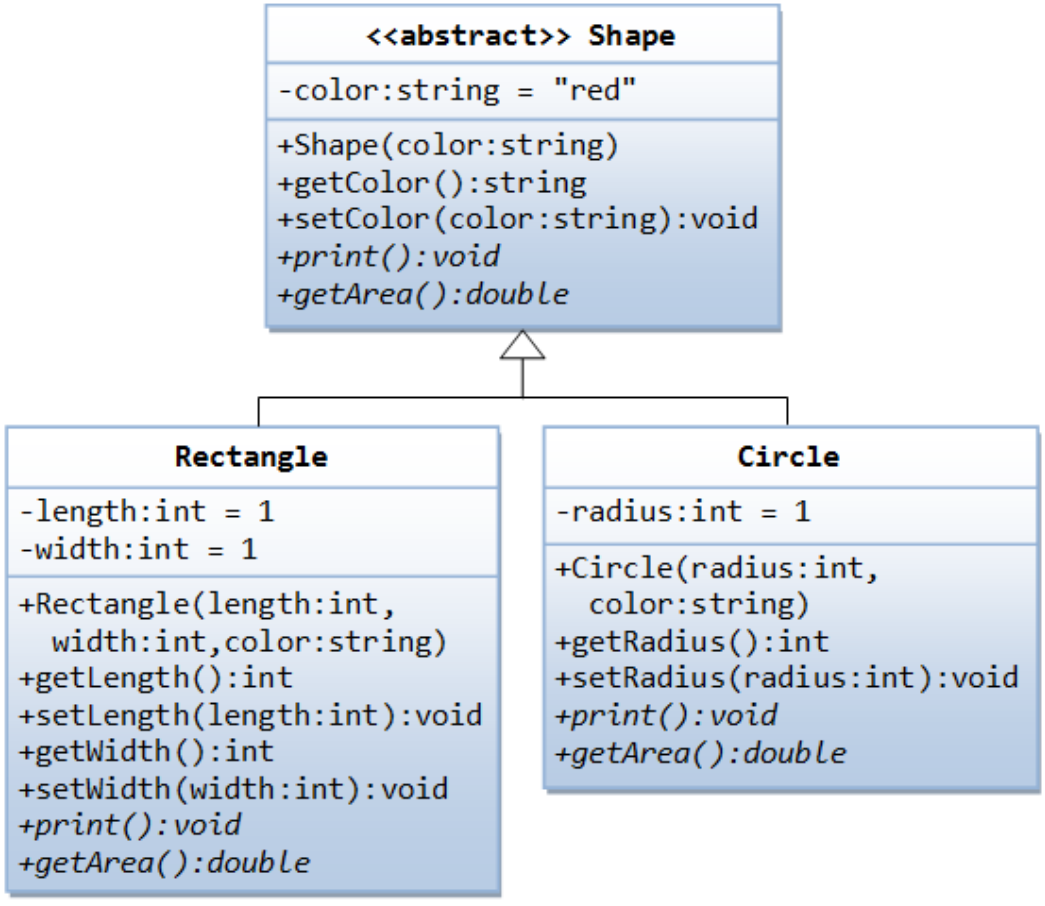
```
/* Header for Rectangle class (Rectangle.h) */
#ifndef RECTANGLE_H
#define RECTANGLE_H

#include "Shape.h"

// The class Rectangle is a subclass of Shape
class Rectangle : public Shape {
private:
   int length;
   int width;

public:
   Rectangle(int length = 1, int width = 1, const string & color = "red");
   int getLength() const;
   void setLength(int length);
   int getWidth() const;
   void setWidth(int width);
   void print() const;      // Override the virtual function
   double getArea() const;  // to implement virtual function
};
#endif

/* Implementation for Rectangle (Rectangle.cpp) */
#include "Rectangle.h"
#include <iostream>

// Constructor
Rectangle::Rectangle(int length, int width, const string & color)
  : Shape(color), length(length), width(width) { }

// Getters
int Rectangle::getLength() const {
   return length; }
int Rectangle::getWidth() const {
   return width; }

// Setters
void Rectangle::setLength(int length) {
   this->length = length; }
void Rectangle::setWidth(int width) {
   this->width = width; }

void Rectangle::print() const {
   std::cout << "Rectangle length=" << length << " width=" << width << ",
subclass of ";
   Shape::print(); }

// Implement virtual function inherited from superclass Shape
double Rectangle::getArea() const {
   return length * width;
}
```

```
/* A test driver program for polymorphism (TestShape.cpp) */
#include "Circle.h"
#include "Rectangle.h"
#include <iostream>
using namespace std;

int main() {
   // Circle object
   Circle c1(5, "blue");
   c1.print();
   cout << endl;
   cout << "area=" << c1.getArea() << endl;

   // Rectangle object
   Rectangle r1(5, 6, "green");
   r1.print();
   cout << endl;
   cout << "area=" << r1.getArea() << endl;

// Shape s1;  // Cannot create instance of abstract class Shape

   // Polymorphism
   Shape * s1, * s2;  // Shape pointers

   s1 = new Circle(6);  // Dynamically allocate a subclass instance
   s1->print();   // Run subclass version
   cout << endl;
   cout << "area=" << s1->getArea() << endl;  // Run subclass version of getArea()

   s2 = new Rectangle(7, 8);  // Dynamically allocate a subclass instance
   s2->print();   // Run subclass version
   cout << endl;
   cout << "area=" << s2->getArea() << endl;  // Run subclass version of getArea()

   delete s1;
   delete s2;

// Shape s3 = Circle(6);  // error: cannot allocate an object of abstract type 'Shape'

   Circle c3(8);
   Shape & s3 = c3;    // Object reference
   s3.print();
   cout << endl;
   cout << "area=" << s3.getArea() << endl;

   Circle c4(9);
   Shape * s4 = &c4;   // Object pointer
   s4->print();
   cout << endl;
   cout << "area=" << s4->getArea() << endl;
}
```

Output

Circle radius=5, subclass of Shape of color=blue
area=78.5398
Rectangle length=5 width=6, subclass of Shape of color=green
area=30
Circle radius=6, subclass of Shape of color=red
area=113.097
Rectangle length=7 width=8, subclass of Shape of color=red
area=56
Circle radius=8, subclass of Shape of color=red
area=201.062
Circle radius=9, subclass of Shape of color=red
area=254.469

Answer:



Q4. (40 points) Use the UML diagram below to create a class called point and using the explanation below demonstrate the class by creating sample points in main and calculating the distance between points.

point
-x:double -y:double
+point(double, double): void +getX():double +getY():double +showPoint():void +calcDistance(point):double

The getX() member function returns the value stored in x. The getY() member function returns the value stored in y. The showPoint() member function displays the point in (x,y) format, for example: (4,3). Demonstrate the class by creating sample points in main and calculating the distance between points.

The distance between two points (x1,y1) and (x2,y2) is: $((x2-x1)^2 + (y2-y1)^2)^{(1/2)}$

Answer:

```
//Point program
#include <iostream>
#include <cmath>
using namespace std;

class point
{
private:  double x;
          double y;

public:   point(double, double);
          double getX();
          double getY();
          void showPoint();
          double calcDistance(point);
};

int main()
{
   point p1(-2,-3);
   point p2(-4,4);

   cout << "The distance bewteen:  " ;
   p1.showPoint();
   cout << " and ";
   p2.showPoint();
   cout << endl;
   cout << p1.calcDistance(p2) << endl;

   return 0;
}
```

```
point::point(double x_in, double y_in)
{
  x = x_in;
  y = y_in;
}

double point::getX()
{
  return x;
}

double point::getY()
{
  return y;
}

void point::showPoint()
{
  cout << "("  << x << ", " << y << ")";
}

double point::calcDistance(point p2)
{
   double xval, yval;
   //Formula broken up for easier reading.
   //May be combiuned into one line.
   xval = (x - p2.getX()) * (x - p2.getX());
   yval = (y - p2.getY()) * (y - p2.getY());
   return sqrt(xval+yval);
}
```