CMPT 135: Lab Work Week 2

1. What happens if we don't specify an access specifier inside a class? Analyze the following program and determine its output. What conclusion do you make? Based on this observation, clearly state the main difference between C++ structs and C++ classes with regards to access specifiers.

```
class Point
{
	float x, y;
	float getX()
	{
		return x;
	}
	float getY()
	{
		return y;
	}
	void setX(float newX)
	{
		x = newX;
	}
	void setY(float newY)
	{
		y = newY;
	}
	void print()
	{
		cout << "(" << x << ", " << y << ")";
	}
};
int main()
{
	Point p;
	p.x = 1;
	p.y = 2;
	cout << "Point p is "; p.print(); cout << endl;

	p.setX(3);
	p.setY(4);
	cout << "Now x coordinate of p is " << p.getX() << endl;
	cout << "And y coordinate of p is " << p.getY() << endl;

	system("Pause");
	return 0;
}
```

2. Consider the following program which has got lots of errors. Identify each error and classify it either as syntax error, run-time error, semantic error or linking error.

```
class Number
{
private:
    int value;
public:
    Number();
    Number(int);
    int getValue();
    void setValue(int);
};
Number::Number(){
    value = 0;
}
Number::Number(int newValue) { value = newValue;}
int Number::getValue()
{
    return value
}
int main()
{
    Number n1, n2;
    n2.setValue(5);
    cout << "n1 is " << n1.getValue() << " and n2 is " << n2.getValue() << endl;
    cout << n2.getValue() / n1.getValue() << endl;
    if (n1.getValue() > n2.getValue())
        cout << "n2 has larger value than n1." << endl;
    else
        cout << "n1 has larger value than n2." << endl;
    system("Pause");
    return 0;
}
```

3. Consider the following class definition:

```
class A
{
private:
  int v;
public:
  A()
  {
       setValue(0);
  }
  A(int value)
  {
       setValue(value);
  }
  int getValue()
  {
       return v;
  }
  void setValue(int v)
  {
       v = v;
  }
};
int main()
```

```
{
  A a1, a2(5);
  cout << a1.getValue() << endl;
  cout << a2.getValue() << endl;
  a1.setValue(2);
  cout << a1.getValue() << endl;
  system("Pause");
  return 0;
}
```

a. Does the class definition have any syntax error?
b. Does the class definition have any linking error?
c. Does the program have any run time error?
d. Correct the class definition so that the program runs with no any error.

4. The following program is designed to demonstrate the different ways member variables can be initialized in constructor member functions.

```
class Triangle
{
private:
  double s1, s2, s3;
public:
  Triangle();
  Triangle(double, double, double);
  double getSide1();
  double getSide2();
  double getSide3();
  void setSide1(double);
  void setSide2(double);
  void setSide3(double);
  double getArea();
};
Triangle::Triangle() : s1(1), s2(2), s3(sqrt(3.0)){}
Triangle::Triangle(double x1, double x2, double x3) : s1(x1)
{
  setSide2(x2);
  s3 = x3;
}
double Triangle::getSide1(){return s1;}
double Triangle::getSide2(){return s2;}
double Triangle::getSide3(){return s3;}
void Triangle::setSide1(double x) {s1 = x;}
void Triangle::setSide2(double x) {s2 = x;}
void Triangle::setSide3(double x) {s3 = x;}
double Triangle::getArea()
{
  double s = (s1 + s2 + s3)/2;
  return sqrt(s*(s-s1)*(s-s2)*(s-s3));
}

int main()
{
  Triangle t1, t2(3, 4, 5);
  cout << "t1 area is " << t1.getArea() << endl;
  cout << "t2 area is " << t2.getArea() << endl;
  system("Pause");
  return 0;
}
```

5. The Square class in the following program provides a non-default constructor but attempts to make use of the default constructor provided by C++ language whenever a class does not provide its own default constructor. Is there any syntax error in the program?

```
class Square
{
private:
  double s;
public:
  Square(double);
  double getSide();
  void setSide(double);
  double getArea();
};
Square::Square(double x) : s(x){}
double Square::getSide(){return s;}
void Square::setSide(double x) {s = x;}
double Square::getArea(){return s*s;}

int main()
{
  Square s1, s2(3);
  cout << "s1 area is " << s1.getArea() << endl;
  cout << "s2 area is " << s2.getArea() << endl;
  system("Pause");
  return 0;
}
```

6. Consider the Point class discussed in the lecture. Add to the class a member function named **getQuadrant** that takes no argument and returns one of the integers 1, 2, 3 or 4 which is the quadrant in 2D space where the Point is found. If the Point is at the origin or one of its coordinates is 0.0, then return 0.

Write a main program to test your **getQuadrant** member function. In your main program create few default and non-default Point objects and print their quadrants by calling **getQuadrant** member function.

7. Consider the Point class discussed in the lecture. Add to the class a member function named **getDistance** that takes no argument and returns the distance of the Point from the origin.

8. Given the Point class discussed in the lecture and the following program, get the program to work correctly by implementing the missing function.

```
#include <iostream>
#include <sstream>
#include <string>
int main()
{
  Point p(1,2);
  cout << "Point p is " << p.toString() << endl;      //This must print "Point p is (1, 2)

  system("Pause");
  return 0;
}
```

9. Given the Point class discussed in the lecture and the following program, get the program to work correctly by implementing the missing function.

```
int main()
{
```

```
  Point p1(1,5), p2(3,4);
  cout << "Point p1 is " << p1.toString() << endl;
  cout << "Point p2 is " << p2.toString() << endl;
  float d = distanceBetweenPoints(p1, p2);//This must assign 2.236068 to d
  cout << "The distance between points p1 and p2 is " << d << endl;

    system("Pause");
  return 0;
}
```

Is the function `distanceBetweenPoints` a member function? What do you call such a function?

10. Consider the Point class discussed in the lecture. Write a non-member function named **getFurthesetPoint** that takes an array of Point objects and its size; and that returns the Point object that is farthest from the origin among all the Point objects in the array. Test you function by writing a main program that reads a size from the user, creates a dynamic array of Points of size given by the user, sets the x and y coordinates of each of the Point object in the array to some random float in the range [-3.0, 5.0), prints each Point object in the array and its distance from the origin, calls the function to compute the furthest point from the origin, and finally prints the furthest Point object and its distance from the origin.

11. Implement **RationalNumber** as a C++ class. A RationalNumber has two integer member variables. Call the member variables **a** (to mean the numerator) and **b** (to mean the denominator). Add the following **member functions:**

- Default constructor: set a = 0 and b = 1
- Non-default constructors with only one argument: set a = the argument, and set b = 1
- Non-default constructor with two arguments: set a and b to the argument values
- Getters and setters: that return member variables and set the member variables
- **toDouble** that returns the RationalNumber object as double
- **standardize** as described in Week 1 Lab Work
- **reduce** as described in Week 1 Lab Work
- **print** to print the rational number in the form of a/b

Write a main program to test your class and member functions. You should be able to decide how many objects to create, what member functions to call, and what kind of outputs should you display in order to convince yourself your class is well designed and well tested for correctness.

12. Consider the Triangle geometrical object. Design a C++ class named **Triangle** to represent Triangle objects. The class must have three **PRIVATE** float type member variables named side1, sde2, side3. Add the following **PUBLIC** constructors and member functions to the class:

- Default constructor: Creates an equilateral triangle of length 1.0
- Non-default constructor with only one argument: Creates an equilateral triangle with the three sides given by the argument
- Non-default constructor with two arguments: Creates an isosceles triangle with its two equal sides given by the first argument and its third side given by the second argument.
- Non-default constructor with three arguments: Creates a scalene triangle with its three sides given by the three arguments
- Add getters and setters. You must have three getter functions for each of the sides and three setter functions for each of the sides. The names of these functions should be getSide1, getSide2, getSide3, setSide1, setSide2, and setSide3.

- Add the **getArea** and **getPerimeter** member functions that return the area of the triangle and the circumference of the triangle. NOTE:- Given a triangle with sides **a**, **b**, and **c**; its area is calculated as **sqrt(s*(s-a)*(s-b)*(s-c))** where **s = (a+b+c)/2**
- Add a member function named print that prints the triangle. In your print member function print the three sides of the triangle, the area of the triangle and the circumference of the triangle, and the type of the triangle (equilateral, isosceles or scalene).

Write a main program to test your **Triangle** class. In your main program create at least three different kinds of triangle objects with the three constructors provided and print each triangle using the print member function.

13. Consider a class representing a **Course** object. Answer the following questions:

a. What member variables would you consider for such a class
b. What member functions would you consider for such a class
c. Implement the Course class with the following attributes

Member Variables:

- name (string), **[make sure to #include <string>]**
- test (float value between 0.0 and 20.0 inclusive),
- midterm (float value between 0.0 and 30.0 inclusive),
- final (float value between 0.0 and 50.0 inclusive) and
- letterGrade (char)

Member Functions:

- Default constructor: Assigns 0 for all the assessments and empty string ("") for name. Also set the letter grade to 'F'.
- Non default constructor: that takes only one string argument. This constructor must assign the string argument to the name member variable and assign 0 for all the assessments. Also set the letter grade to 'F'.
- Non default constructor: that takes a string and three float arguments for the assessments. This constructor must assign the name and the assessments the values of the arguments. Also set the letter grade according to the scheme: [90, 100] = 'A', [75, 90) = 'B', [60, 75) = 'C', [50, 60) = 'D' and [0, 50) = 'F'.
- Getters for each member variable. (**getName**, **getTest**, **getMidterm**, **getFinal**, **getLetterGrade**)
- Setters for each member variable. Do not have a setter for the letter grade because it is set in the constructors. (**setName**, **setTest**, **setMidterm**, **setFinal**). However, each setter of the assessments must modify the letter grade when you call the setter to reflect the change in the total course mark.
- Print member function to print the name, assessments and the letter grade.

Now, write a main program to test your Course class. In your main program create at least three Course objects and construct objects using any of the three constructors provided. Finally print the course objects to the screen using your print member function.