Lab No. 08

Composition

OBJECTIVE:

Things that will be covered in today's lab:

- Composition

THEORY:

Composition (aggregation) is another way to relate two classes. In composition (aggregation), one or more members of a class are objects of another class type. Composition is a "has a" relation; for example "every person has a date of birth".

```
class DOB
{
      // member variables/ functions
};
class person
{
      DOB date; // Declare object of date of birth
       // member variable/ functions
}student;
```

Access nested members:

- You can access the members of "*person*" class using dot operator.
- As *"date"*isthe object of *"DOB"* which is a member of *"person"*class, so Members of *"DOB"* can be accessed using "*person*" class object.

```
// Access member variables
Student.person_members
Student.date.DOB_members
```

Now, let us discuss how the constructors of the objects of date are invoked.Recall that a class constructor is automatically executed when a class object enters itsscope. Suppose that we have the following statement:

```
person student;
```

When the object "student" enters its scope, the objects "date", which is themember of the student, also enters their scope. As a result, one of their constructors is executed. We, therefore, need to know how to pass arguments to the constructors of the member objects "date", which occurs when we give the definitions of the constructors of the class. Recall that constructors do not have a type and so cannot be called like other functions. The arguments to the constructor of a member object are specified in the heading part of the definition of the constructor of the class.

Furthermore, member objects of a class are constructed (that is, initialized) in the order they are declared (not in the order they are listed in the constructor's member initialization list) and before the containing class objects are constructed. Thus, in our case, the object "date" is initialized first and then "student".

.

The following statements illustrate how to pass arguments to the constructors of the member objects DOB:

```
      person::person(string f_n, string l_n, int m, int d, int y)
                    :date(m, d, y)
      {
            // Definition of Constructor person
      }
```

Example: What should be the output of this program?

```
class Birthday
{
public:
    Birthday(int cmonth, int cday, int cyear){
        month = cmonth;
        day = cday;
        year = cyear;
    }
void printDate(){ cout<<month<<"/"<<day <<"/"<<year;}
private:
int month;
int day;
int year;
};
class People
{
public:
    People(string cname, Birthday cdateOfBirth)
           :name(cname),dateOfBirth(cdateOfBirth){}
void printInfo(){
        cout<<name <<" was born on: ";
        dateOfBirth.printDate();
    }
private:
    string name;
    Birthday dateOfBirth;
};
int main()
{
    Birthday birthObject(7,9,97);
    People infoObject("Lenny the Cowboy", birthObject);
    infoObject.printInfo();
}
```

Exercise 1: [illegible])

Every Circle has a center and a radius. Create a class **CircleType** that can store the center, the radius, and the color of the circle. Since the center of a circle is a point in the x-y plane, create a class **PointType** to store the x and y coordinate. Use class *PointType* to define the class *CircleType*.

Provide *constructors* that enable objects of these classes to be initialized when they are declared. The constructors should contain default values in case no initializes are provided.
The definition of class *CircleType* and class *PointType* is as under: (you may define additional functions if you require any).

```
	class PointType
	{
		int x;
		int y;
public:
		//default constructor
		//constructor so that objects are initialized
		//print the x and y- coordinates
		//determine the quadrant in which the point lies
		//getter and setter functions
	};
	class CircleType
	{
		double radius;
		char * color;
		PointType center;
	public:
		//default constructor
		//constructor so that objects are initialized
		//print the radius, center, color
		//calculate area
		//calculate circumference
		//getter and setter functions
		//destructor
	};
```

Your program should run for the following main.

```
int main()
{
	CircleType C(21,2,3.5,"blue");
	cout<<"\n************************\n"<<endl;
	C.print();
	cout<<"\n************************\n"<<endl;
	cout<<" Area of circle is   "<<C.calc_area();
	cout<<"\n\n************************\n\n"<<endl;
```

```
        PointType P(-20,3);
        int p=P.checkquad();
        P.print();

        switch (p)
        {
        case 0:
                cout<<"Point lies at center"<<endl;
                break;
        case 1:
                cout<<"Point lies in I quadrant"<<endl;
                break;
        case 2:
                cout<<"Point lies in II quadrant"<<endl;
                break;
        case 3:
                cout<<"Point lies in III quadrant"<<endl;
                break;
        case 4:
                cout<<"Point lies in IV quadrant"<<endl;
                break;
        default:
                cout<<"INVALID";
                break;
        }

        int x;
inty;
        double r;
        char col[9];

        CircleType circ(2,5,4.89,"purple");
        cout<<"\n\n**********************\n\n";
        circ.print();

        cout<<"\n Enter radius \n";
        cin>>r;
        cout<<"\n Enter the coordinates where the center lies \n";
        cin>>x>>y;
        cout<<"\n Enter color \n";
        cin>>col;

        circ.setparam(x,y,r,col);
        cout<<"\n\n**********************\n\n";
        circ.print();
        cout<<"\n\n**********************\n\n";
}
```

Post Lab:

Given the following two classes:

```
class course
{
int courseNumber;
int creditHours;
public:
void setCourse (int x,int y);
};

class section
{
int secNumber;
course c;//composition
public:
void setSection (int,int,int);
};
```

- Provide a meaningful implementation for class course and section.
- Write a main program that declares an array of 7 objects of type section and set their values:
    - Course number: 117 for all sections.
    - Credit hours: 3 for all sections.
    - Section number: give each section a unique number from 1-7.