1.4.2 Activity

Create a class called **Point** that has two data members: **x-** and **y**-coordinates of the point. Provide a no-argument and a 2-argument constructor. Provide separate get and set functions for the each of the data members i.e. **getX**, **getY**, **setX**, **setY**. The getter functions should return the corresponding values to the calling function. Provide a **display** method to display the point in (x, y) format. Make appropriate functions **const**.

2.4.1 Activity

Reuse **Complex** class given in **section 2.3.2 (C++)**, .

perform arithmetic with complex numbers. Note that addition and printing is already done in given sections. Add the following public methods to perform complex subtraction and multiplication as well:

a) **Input:** Write class function input() to take complex number real and imaginary parts from user on runtime. Note: input takes no arguments and returns nothing.

b) **Subtract two Complex numbers:** Write class function subCom() taking two complex objects c1 and c2. Difference is computed as following: the real part of the right operand is subtracted from the real part of the left operand, and the imaginary part of the right operand is subtracted from the imaginary part of the left operand. Note: return type of subCom is void.

c) **Multiply two Complex numbers:** Write class function mulCom() taking two complex objects c1 and c2. Product is computed as following: Suppose you are trying to compute the product of two complex numbers a + bi and c + di. The real part will be ac – bd, while the imaginary part will be ad+ bc. Note: return type of subCom is void.

4.5.1 Activity

a) **C++:** Create a class called employee. This class maintains information about name (char*), department (char*), salary (double), and period of service in years (double).

1. Provide a parameterless constructor to initialize the data members to some fixed values.
2. Provide a 4-argument parameterized constructor to initialize the members to values sent from calling function.
   a. (You have to make dynamic allocation for both name and department data members in constructor.)

3. Provide a copy-constructor that performs the deep copy of the data members.
4. Provide an input function that takes all the values from user during run-time.
5. Provide a show function that shows all the information about a specific employee to user.
6. Provide a destructor to free the memory allocated to name and department in constructor.

Write all the member function outside a C++ class. Write a driver program to test the functionality of the above-mentioned class.

5.4.1 Activity

Reuse Point class of Lab 1 (Activity 1.4.2) as base class. Make the member data protected. Write all class member functions outside Point class.

Derive a class Circle from this Point class that has an additional data member: radius of the circle. The point from which this circle is derived represents the center of circle. Provide a no-argument constructor to initialize the radius and center coordinates to 0. Provide a 2-argument constructor: one argument to initialize the radius of circle and the other argument to initialize the center of circle (provide an object of point class in the second argument). Provide a 3-argument constructor that takes three floats to initialize the radius, x-, and y-coordinates of the circle. Provide setter and getter functions for radius of the circle. Provide two functions to determine the radius and circumference of the circle. Write all class member functions outside Circle class.

Also derive another class Cylinder from the Point class. This class contains an additional data member: radius and height of cylinder. Provide appropriate constructors to initialize the center, radius, and height of the cylinder. Provide functions to determine the area and volume of the cylinder. Area of a cylinder is 2πr*(r + h). Volume of cylinder is 2πr*r*h. Use the findCircum() of circle class where required. Write a main function to test these classes.

6.3.1 Activity [Multilevel Inheritance]

Create a class **First.** It contains one protected data member *f* and one public input function *f_Input()*. Use the function to take *f* from user on runtime.

Next, create a derived class **Second** from **First** class. This class also contains only one protected data member *s* and one public input function *s_Input*(). Call *f_Input*() function inside *s_Input*() and then take *s* from user on runtime.

Finally, create another derived class **Third** from **Second** class. This class contains one protected data member *t*. It contains three public functions. An input function *t_Input()* that takes *t* from user on runtime, a max function *max*() that finds maximum of *f*, *s*, and *t* and displays the maximum, and show function that displays *f*, *s*, and *t*. Note, call *s_Input*() inside t_Input() and then take *t* from user.

Write main function to test the functionality. Create an object of **Third.** Call t_input(), show(), and max() functions according to test case given in 6.4.

7.4.1 Activity

Define an abstract base class **shape** that includes protected data members for area and volume of a shape, public methods for computing area and volume of a shape (make these functions virtual), and a display function to display the information about an object. Make this class abstract by making display function pure virtual.

Derive a concrete class **point** from the **shape** class. This **point** class contains two protected data members that hold the position of **point**. Provide no-argument and 2-argument constructors. Override the appropriate functions of base class.

Derive a class **Circle** publicly from the **point** class. This class has a protected data member of radius. Provide a no-argument constructor to initialize the fields to some fixed values. Provide a 3-argument constructor to initialize the data members of **Circle** class to the values sent from outside. Override the methods of base class as required.

Derive another class **Cylinder** from the **Circle** class. Provide a protected data member for height of cylinder. Provide a no-argument constructor for initializing the data members to default values. Provide a 4-argument constructor to initialize x- and y-coordinates, radius, and height of cylinder. Override the methods of base class.

Write a driver program to check the polymorphic behavior of this class.

8.4.1 Activity

Create a class called **Distance** containing two members feet and inches. This class represents distance measured in feet and inches. For this class, provide the following functions:

a) A **no-argument constructor** that initializes the data members to some fixed values.

b) A **2-argument constructor** to initialize the values of feet and inches to the values sent from the calling function at the time of creation of an object of type Distance.

c) A **showDistance()** to show the distance in feet and inches.

d) Overloaded arithmetic operators

   a. **operator+** to add two distances: Feet and inches of both objects should add in their corresponding members. 12 inches constitute one feet. Make sure that the result of addition doesn't violate this rule.

   b. **operator+=** for addition of two distances.

i) Overloaded relational operators

   a. **operator >** should return a variable of type **bool** to indicate whether 1st distance is greater than 2nd or not.

   b. **operator <** should return a variable of type **bool** to indicate whether 1st distance is smaller than 2nd or not.

   c. **operator >=** should return a variable of type **bool** to indicate whether 1st distance is greater than or equal to 2nd or not.

   d. **operator <=** should return a variable of type **bool** to indicate whether 1st distance is smaller than or equal to 2nd or not.

j) Overloaded equality operators

   a. **operator==**should return a variable of type **bool** to indicate whether 1st Distance is

equal to the 2nd distance or not.

   b. **Operator!=** should a **true** value if both the distances are not equal and return a **false** if both are equal.

k) Overloaded operators for **pre- and post- increment**. These increment operators should add a 1 to the **inches**. Keep track that **inch** should not exceed 12.

l) Overload operators for **pre- and post- decrement**. These decrement operators should subtract a 1 from **inches**. If number of inches goes below 0, take appropriate actions to make this value valid.