	AIR UNIVERSITY
	DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING
	EXPERIMENT NO 7

Lab Title: Revision lab (1-6)

Student Name: ______________________ **Reg. No:** ______________

Objective: ____

____

LAB ASSESSMENT:

Attributes	Excellent (5)	Good (4)	Average (3)	Satisfactory (2)	Unsatisfactory (1)
Ability to Conduct Experiment					
Ability to assimilate the results					
Effective use of lab equipment and follows the lab safety rules					

Total Marks: ______________ Obtained Marks: ______________

LAB REPORT ASSESSMENT:

Attributes	Excellent (5)	Good (4)	Average (3)	Satisfactory (2)	Unsatisfactory (1)
Data presentation					
Experimental results					
Conclusion					

Total Marks: ______________ Obtained Marks: ______________

Date: ______________ Signature: ______________

EXPERIMENT NO 7

Revision lab (1-6)

Objective:

- To revise all the concepts from lab 1 to lab 6

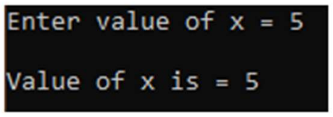
Equipment Required:

- Visual Studio/ Dev C++

Description:

- **Structures**

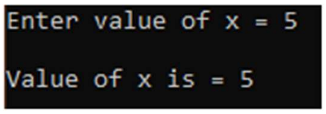
```
struct point{
        int x;                      // data member

        void input(){                       // member function
                cout << "Enter value of x = ";
                cin >> x;
        }
};

int main(){
        point p;                    // creating variable of type struct
        p.input();                          // calling input function
        cout << endl;
        cout << "Value of x is = " << p.x;      // accessing value of x
        return 0;
}
```



```
Enter value of x = 5

Value of x is = 5
```

- **Classes**

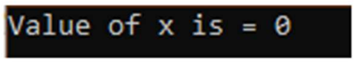
```
class point{
            int x;                          // private data member
            public:                         // public mode
            void input(){                                   // member function
                        cout << "Enter value of x = ";
                        cin >> x;
            }
            void display(){                             // member function
                        cout << "Value of x is = " << x;
            }
};

int main(){
            point p;                // creating object of class
            p.input();              // calling input function
            cout << endl;
            p.display();            // calling display function
            return 0;
}
```



```
Enter value of x = 5

Value of x is = 5
```

- **Default Constructor**

```
class point{
            int x;                          // private data member
            public:                         // public mode
            point(){                    // default constructor
                        x = 0;
            }
            void display(){                 // calling member function
                        cout << "Value of x is = " << x;
            }
};

int main(){
            point p;                    // p is object of class point
            p.display();            // calling member function
            return 0;
}
```



```
Value of x is = 0
```

- **Parametrized Constructor**

```
class point{
        int x;                  // private data member
        public:                   // public mode
        point(int x1){          // parametrized constructor
                x = x1;
        }
        void display(){                 // member function
                cout << "Value of x is = " << x;
        }
};

int main(){
        point p(5);                // creating object of class
        p.display();             // calling member function
        return 0;
}
```

```
Value of x is = 5
```

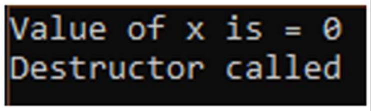
- **Copy Constructor**

```
class point{
        int x;                          // private data member
        public:                         // public mode
        point(int x1){                  // parametrized constructor
                x = x1;
        }
        Point(const point &p) {           // copy constructor
                x = p.x;
        }
        void display(){                   // member function
                cout << "Value of x is = " << x;
        }
};

int main(){
        point p(5);               // creating object of class
        p.display();             // calling member function
        cout << endl;
        point q = p;              // copy constructor
        q.display();              // calling member function
        return 0;
}
```

```
Value of x is = 5
Value of x is = 5
```

- **Destructor**

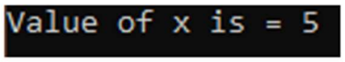
```
class point{
        int x;                      // private data member
        public:                     // public mode
        point(){                    // default constructor
                x = 0;
        }
        void display(){             // calling member function
                cout << "Value of x is = " << x;
        }
        ~point(){                   // destructor
                cout << endl << "Destructor called";
        }
};

int main(){
        point p;                    // p is object of class point
        p.display();                // calling member function
        return 0;
}
```



```
Value of x is = 0
Destructor called
```

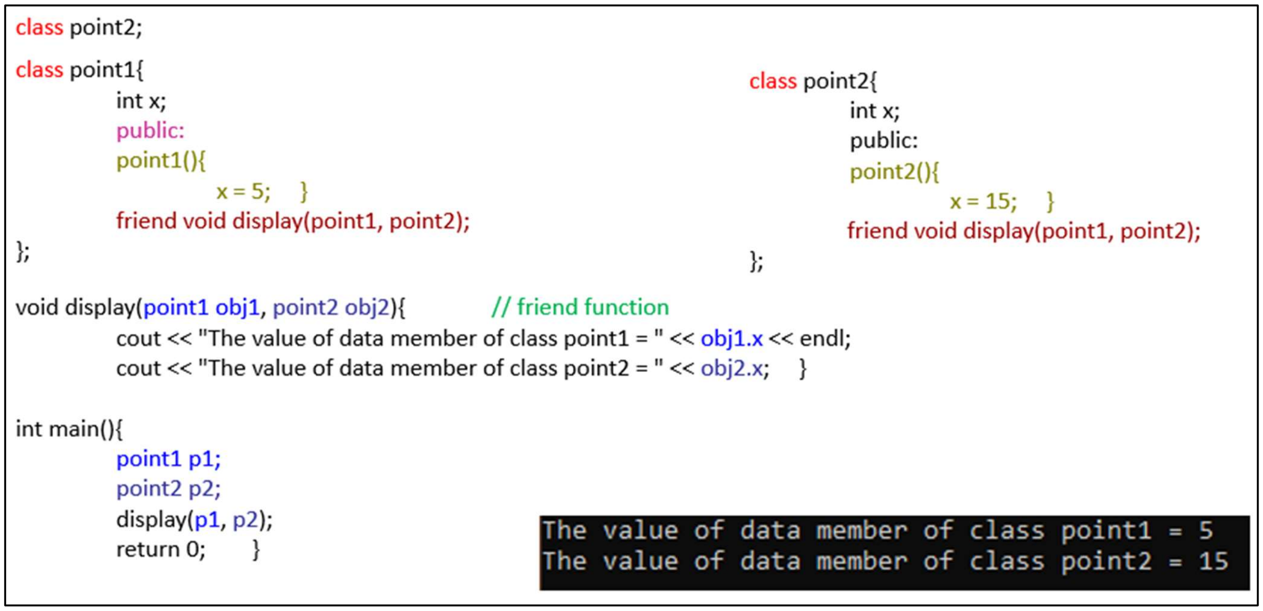
- **Friend Class**

```
class point{
        int x;                      // private data member
        public:                     // public mode
        point(){                    // default constructor
                x = 5;
        }
        friend class show;          // declaring class show as friend of class point
};
class show{
        public:
        void display(point obj){        // passing object of class point to function member
                cout << "Value of x is = " << obj.x;
        }
};
int main(){
        point p;                        // creating object of class point
        show s;                         // creating object of class show
        s.display(p);       // calling member function using object of class show and
                              passing object of class point as an argument to member function
        return 0;     }
```



```
Value of x is = 5
```

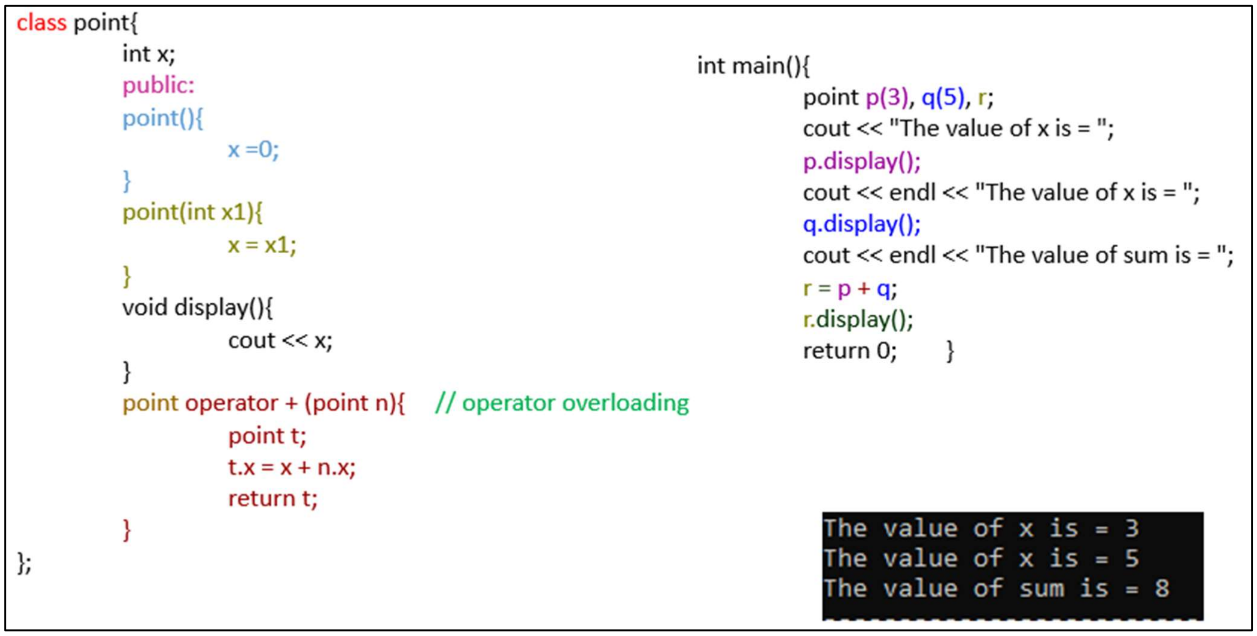
- **Friend Function**



```
class point2;
class point1{
        int x;
        public:
        point1(){
                x = 5;    }
        friend void display(point1, point2);
};

class point2{
        int x;
        public:
        point2(){
                x = 15;    }
        friend void display(point1, point2);
};

void display(point1 obj1, point2 obj2){          // friend function
        cout << "The value of data member of class point1 = " << obj1.x << endl;
        cout << "The value of data member of class point2 = " << obj2.x;    }

int main(){
        point1 p1;
        point2 p2;
        display(p1, p2);
        return 0;      }
```

```
The value of data member of class point1 = 5
The value of data member of class point2 = 15
```

- **Operator overloading**



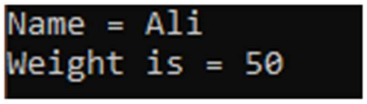
```
class point{
        int x;
        public:
        point(){
                x =0;
        }
        point(int x1){
                x = x1;
        }
        void display(){
                cout << x;
        }
        point operator + (point n){    // operator overloading
                point t;
                t.x = x + n.x;
                return t;
        }
};

int main(){
        point p(3), q(5), r;
        cout << "The value of x is = ";
        p.display();
        cout << endl << "The value of x is = ";
        q.display();
        cout << endl << "The value of sum is = ";
        r = p + q;
        r.display();
        return 0;      }
```

```
The value of x is = 3
The value of x is = 5
The value of sum is = 8
```

- **Composition**

```
class weight{
        int x1;
        public:
        weight(){
                x1 = 50;  }

        void display(){
                cout << "Weight is = " << x1;   }
};
// Every person "has" some weight
class person{          // class person is created using class weight
        string x;
        weight w;
        public:
        person(){
                x = "Ali";    }

        void display(){
                cout << "Name = " << x << endl;
                w.display();    }
};
```

```
int main(){
        person p;
        p.display();

        return 0;     }
```



```
Name = Ali
Weight is = 50
```

LAB TASK

1. Write a **structure** that stores the
   - Distance covered by a player
   - Minutes taken to cover the distance by a player
   - Seconds taken to cover the distance by a player

The structure contains the following member functions.

- A function to take values of data members from the user.
- A display function to show the value of data members.

The program should

- Input the record of two players and
- Display the record of the winner. (Hint: total time/ distance)

```
struct player{
        int distance, minutes, seconds;

        void input(){
        }

        void display(){
        }
};

int main(){
        player p1, p2;
        p1.input();
        p1.display();
        cout << endl;
        p2.input();
        p2.display();

        float t1, t2;
        t1 = total time / distance      // for player 1
        t2 = ……                         // for player 2
        if (t1 > t2 ){
                        ........
        }
        return 0;
}
```

2. Write a **class car** that contains the following data members.
   - The name of car
   - The direction of car (East, West, North, South)
   - The distance covered by car

The class contains the following member functions.

- A constructor to initialize the data members.
- Turn function to change the direction of car to one step right side (e.g., if the direction is to east, it should be changed to south and so on).
- Move function to change the position of car away from zero point. It should accept the distance as parameter.
- A display function to show the value of data members.

```
class car{
        string name, dir;
        int pos;
        public:
                car(){
                        name = "alto";
                        dir = "East";
                        pos = 0;
                }
                void turn(){
                        if (dir == "East")
                          dir = "West";
```

```
                    .....
            }
            void move(int m){
                    // add m to the current distance.
            }
            void display(){
            }

};

int main(){
        car c;
        c.display();
        cout << endl;
        c.turn();
        c.display();
        cout << endl;
        c.move(50);
        c.display();

        return 0;
}
```

3. Write a class **travel** that stores the data of two travelers. It contains the following data members.
   - Distance in kilometers
   - Time in hours

The class contains the following member functions.

- A constructor to initialize the data members to zero.
- A function to input the values.
- A function to display the values.
- A function that takes an object of type travel as argument, adds the distance and time of both travelers. This function returns the object of class type as well.

```
class travel{
        int km, hr;
        public:
                travel(){
                        km =0;
                        hr =0;
                }
                void input(){
                }
                void display(){
                }
                travel add(travel e){
                }
};

int main(){
        travel t1,t2, l;
        t1.input();
        t1.display();
        cout << endl;
        t2.input();
        t2.display();
        l = t1.add(t2);
        cout << endl;
        l.display();
        return 0;
}
```

4. Create a **class Time** which contains:

  - Hours

  - Minutes

  - Seconds

 Write a C++ program using operator overloading for the following:

  1. >> : To accept the time.

  2. << : To display the time.

  3. = = : To check whether two Time are same or not.

```
class time
{
private:
   int hr, min, sec;
public:
   time()
   {  hr = min = sec =0;}

   friend ostream & operator << (ostream &out, const time &c);
   friend istream & operator >> (istream &in,  time &c);

   friend bool operator==(time &t1, time &t2);
};

istream & operator >> (istream &in,  time &c)
{
   cout << "Enter hour ";
   in >> c.hr;
   . . . .
   return in;
}

ostream & operator << (ostream &out, const time &c)
{
    out << c.hr;
    . . . . .
    return out;
}

bool operator== (time &t1, time &t2)
{
    return ( t1.hr == t2.hr && . . . . . );
}

int main()
{
   time t1, t2;
   cin >> t1;
   cout << "The first time is ";
   cout << t1;
   cout << endl;
   cin >> t2;
   cout << "The second time is ";
   cout << t2;

   if(t1 == t2)
    {
        cout << "Both the time values are equal";
    }
    else
    {
        cout << "Both the time values are not equal";
    }

   return 0;
}
```