### **Activity 01**

Create a class called **Time** that has separate int member data for hours, minutes, and seconds. Provide the following member functions for this class:

1. A **no-argument constructor** to initialize hour, minutes, and seconds to 0.
2. A **3-argument constructor** to initialize the members to values sent from the calling function at the time of creation of an object. Make sure that valid values are provided for all the data members. In case of an invalid value, set the variable to 0.
3. A member function **show** to display time in 11:59:59 format.
4. An overloaded **operator+** for addition of two Time objects. Each time unit of one object must add into the corresponding time unit of the other object. Keep in view the fact that minutes and seconds of resultant should not exceed the maximum limit (60). If any of them do exceed, subtract 60 from the corresponding unit and add a 1 to the next higher unit.
5. Overloaded operators for **pre- and post- increment**. These increment operators should add a 1 to the **seconds** unit of time. Keep track that **seconds** should not exceed 60.
6. Overload operators for **pre- and post- decrement**. These decrement operators should subtract a 1 from **seconds** unit of time. If number of seconds goes below 0, take appropriate actions to make this value valid.

A **main()** programs should create two initialized **Time** objects and one that isn’t initialized. Then it should add the two initialized values together, leaving the result in the third **Time** variable. Finally it should display the value of this third variable. Check the functionalities of ++ and -- operators of this program for both pre- and post-incrementation.

### **Activity 02**

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

1. A **no-argument constructor** that initializes the data members to some fixed values.
2. 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.
3. Overloaded arithmetic operators
4. **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.
5. **operator+=** for addition of two distances.
6. overloaded relational operators
7. **operator >**: should return a variable of type **bool** to indicate whether 1st distance is greater than 2nd or not.
8. **operator <**: should return a variable of type **bool** to indicate whether 1st distance is smaller than 2nd or not.
9. **operator >**=: should return a variable of type **bool** to indicate whether 1st distance is greater than or equal to 2nd or not.
10. **operator <=**: should return a variable of type **bool** to indicate whether 1st distance is smaller than or equal to 2nd or not.
11. Overloaded equality operators
12. **operator==:** should return a variable of type **bool** to indicate whether 1st Distance is equal to the 2nd distance or not.
13. **Operator!=**: should a **true** value if both the distances are not equal and return a **false** if both are equal.

### **Activity 03**

Create a **Point** class has two coordinates x and y. Provide the following member functions:

1. Two constructors: one for initializing to fixed values and one for initializing to the values sent from outside.
2. A function for displacing the object Point. Provide two values to this function which are added to the x and y coordinates as displacements.
3. Functions for overloading the addition and subtraction operators: +=, +, -=, and unary and binary-.
4. Function for overloading the equality operator.

Now use the overloaded operator + for adding displacement to a Point. In this case, the displacement will be represented by an object of type Point. So

P2=P1 + Disp; where P2, P1, and Disp all are instances of class **Point**.

### **Activity 04**

Create a class called **Martix** that represents a 3x3 matrix. Create a constructor for initializing the matrix with 0 values. Create another overloaded constructor for initializing the matrix to the values sent from outside. Overload the + and += operators for addition of two matrices, == operator for checking the equality of two matrices, and \*operator for finding the product of the two matrices. Define all the member functions outside the class.

### **Activity 05**

Create a class called **Complex** for performing arithmetic with complex numbers. Complex numbers have the form realPart + imaginaryPart \* i

where i is 

Write a program to test your class. Use floating-point variables to represent the private data of the class. Provide a constructor that enables an object of this class to be initialized when it is declared. Provide a no-argument constructor with default values in case no initializers are provided. Provide public methods that perform the following operations:

1. **Add two Complex numbers**: Overload the operators +, += for addition of two complex numbers.
2. **Subtract two Complex numbers**: overload the operators -, -= for subtraction of two complex numbers.
3. **Increment a complex number**: incrementing a complex number results in addition of 1 to the real portion. Overload both the pre- and post-incrementation operators.
4. **Decrement a complex number**: decrementing a complex number results in subtraction of 1from the real portion. Overload both the pre- and post-decrementation operators.
5. **Print** Complex numbers in the form (a, b), where a is the real part and b is the imaginary part.
6. **Multiply two complex number**: overload \* and \*= operators for multiplication of complex numbers.

### **Activity 06**

Let us model digital storage. Digital data is stored in the form of bits. 8 bits constitute one byte. Create a class **Storage** that specifies the size of a file. This class has two integer data members: bits and bytes.

1. Provide a **no-argument constructor** to initialize the size of some file to 0 bits and 0 bytes.
2. Provide a **two-argument constructor** to initialize the size of a file to values specified at the time of creation of an object.
3. Provide an overloaded **operator +** that is used to merge two files together in a third file.
4. Provide an overloaded **operator +=** that is used to concatenate one file at the end of the other.
5. Provide overloaded **post-increment and pre-increment** operators to increment the size of a file by one bit. (You must write the functions to accommodate statements like f2=++f1; and f2=f1++; where f1 and f2 are instances of the class Storage)
6. Provide an overloaded **operator >** to determine whether one file is larger in size than the other.

Write a test program to test the functionality of this class.