Aim:

To write a program to create a class `employee` with following specifications :

- `calculate()` : A member function to net salary
- `havedata()` : A member function to accept values
- `displaydata()` : A member function to display all data members

Theory:

The building block of C++ that leads to Object Oriented programming is a `class`. It is a user defined data type, which holds its own data members and member functions, which can be accessed and used by creating an instance of that class. A class is like a blueprint for an object. An object is an instance of a Class. When a class is defined, no memory is allocated but when it is instantiated (i.e. an object is created) memory is allocated. The public data members are also accessed in the same way given however the private data members are not allowed to be accessed directly by the object. Accessing a data member depends solely on the access control of that data member. This access control is given by Access modifiers in C++. There are three access modifiers : public, private and protected. The member functions defined inside the class are inline functions and declared outside are treated as normal functions.

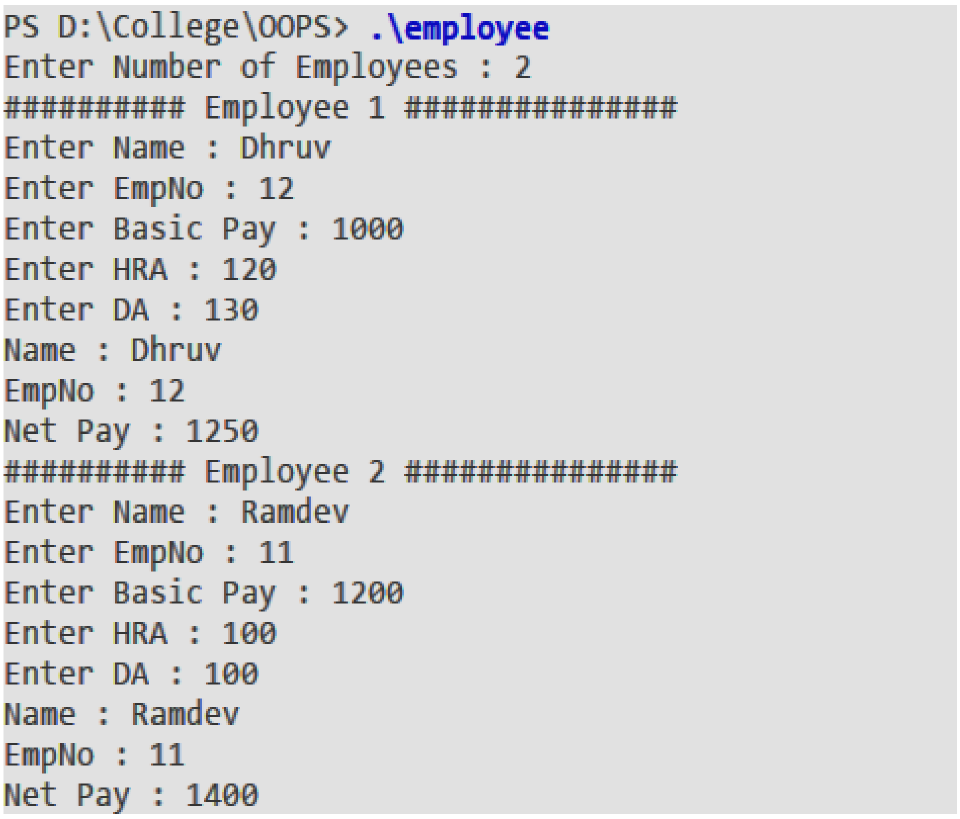
Syntax :

```
// Class Declaration
class name{
    private:
    //data members and member functions
    public:
    //data members and member functions
    protected:
    //data members and member functions
};

// Object Declaration
name object;
```

Code :

```
#include <iostream>
// #include <string>

using namespace::std;

class Employee {
    char name[50] ;
    int empno ;
    float basic ;
    float hra ;
    float da ;
    float netpay ;

    float calculate() {
        return basic + da + hra ;
    }

    public :

    void havedata() {
        cout << "Enter Name : " ;
        cin >> name ;
        cout << "Enter EmpNo : " ;
        cin >> empno ;
        cout << "Enter Basic Pay : " ;
        cin >> basic ;
        cout << "Enter HRA : " ;
        cin >> hra ;
        cout << "Enter DA : " ;
        cin >> da ;
        netpay = calculate() ;
    }

    void displaydata() {
        cout << "Name : "  << name << endl ;
        cout << "EmpNo : "  << empno << endl ;
        cout << "Net Pay : "  << netpay << endl ;
    }
};


int main () {
    int n ;
    cout << "Enter Number of Employees : " ;
    cin >> n ;

    Employee emp[n] ;

    int i = 0 ;
    while(i < n) {
        cout << "##### Employee " << i+1 << " ######" << endl ;
        emp[i].havedata() ;
        emp[i++].displaydata() ;
    }
    return 0 ;
}
```

Output :



```
PS D:\College\OOPS> .\employee
Enter Number of Employees : 2
########## Employee 1 ###############
Enter Name : Dhruv
Enter EmpNo : 12
Enter Basic Pay : 1000
Enter HRA : 120
Enter DA : 130
Name : Dhruv
EmpNo : 12
Net Pay : 1250
########## Employee 2 ###############
Enter Name : Ramdev
Enter EmpNo : 11
Enter Basic Pay : 1200
Enter HRA : 100
Enter DA : 100
Name : Ramdev
EmpNo : 11
Net Pay : 1400
```

Discussion :

In the above program we have created a class `Employee` in which number, name and salary are data member and there are two member function for input and output. Now in the `main` program we have created an array of objects which means we have created a number of objects with each object can be represented as an empty which stores the info of each employee and ask user to enter the info of each
employee. Thus, displaying information of each employee as shown in output.

Learning Outcomes :

By studying classes, we learnt the following things: -

- Classes contain, data members and member functions, and the access of these data members and variable depends on the access specifiers .

- Class member functions can be defined inside the class definition or outside the class definition.

- Classes in C++ are similar to structures in C, the only difference being, class defaults to private access control, where as structure defaults to public.
- All the features of OOPS, revolve around classes in C++.
- Objects of class holds separate copies of data members. We can create as many objects of a class as we need.

Aim:

Write a program to add, subtract, multiply and divide two complex number using **Operator Overloading** .

Theory:

Operator overloading is an important concept in C++. It is a type of polymorphism in which an operator is overloaded to give user defined meaning to it. Overloaded operator is used to perform operation on
user-defined data type. For example, ‘+’ operator can be overloaded to perform addition on various data types, like for Integer, String(concatenation) etc.
Almost any operator can be overloaded in C++. However, there are few operator which can not be overloaded. Operator that are not overloaded are follows

- Scope Resolution Operator - `::`
- Member Access or Dot operator - `.`
- Pointer-to-member Operator - `*`
- Ternary or Conditional Operator - `?:`

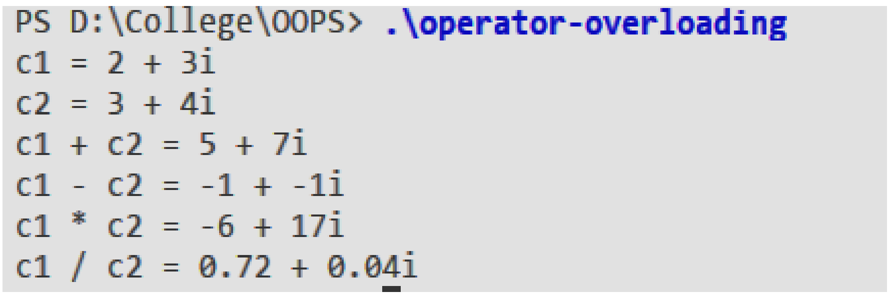
Syntax :

```
data_type classname :: operator symbol (arguments){
    //function body
}
```

Code :

```
#include <iostream>
using namespace std;

class Complex {
    float real ;
    float imag ;

    public :
    Complex () {
        real = 0 ;
        imag = 0 ;
    }
    Complex (float x , float y) {
        real = x ;
        imag = y ;
    }
    void display(){
        cout << real << " + " << imag << "i" << endl ;
    }
    Complex operator + ( Complex x) {
        return Complex( real + x.real , imag + x.imag );
    }
    Complex operator - ( Complex x) {
        return Complex( real - x.real , imag - x.imag );
    }
    Complex operator * ( Complex x) {
        return Complex( real * x.real - imag * x.imag  , real * x.imag + imag * x.real )
    }
    Complex operator / ( Complex x) {
        return Complex( (real * x.real + imag * x.imag) / ( x.real * x.real + x.imag * x
                        (imag * x.real - real * x.imag) / ( x.real * x.real + x.imag * x
    }
} ;

int main(){

    Complex c1(2.0 , 3.0 ) , c2(3.0 , 4.0) , c3 ;

    cout << "c1 = " ;
    c1.display();
    cout << "c2 = " ;
    c2.display();

    cout << "c1 + c2 = " ;
    c3 = c1 + c2 ;
    c3.display();

    cout << "c1 - c2 = " ;
    c3 = c1 - c2 ;
    c3.display();

    cout << "c1 * c2 = " ;
    c3 = c1 * c2 ;
    c3.display();

    cout << "c1 / c2 = " ;
    c3 = c1 / c2 ;
    c3.display();

    return 0 ;
}
```

Output :



```
PS D:\College\OOPS> .\operator-overloading
c1 = 2 + 3i
c2 = 3 + 4i
c1 + c2 = 5 + 7i
c1 - c2 = -1 + -1i
c1 * c2 = -6 + 17i
c1 / c2 = 0.72 + 0.04i
```

Discussion :

In the above program we created a class complex which is used for carrying operation on complex numbers like addition, subtraction, multiplication and division with the help of operator overloading.
Operator overloading refers to operator polymorphism. For example, + operator can be used to add two number but also can be used to add two strings together. So similar above we overload +, -, *, / with each operator represents its respective operation on complex numbers operators for two complex numbers.

Learning Outcomes :

- We have learned that by using overloading operator our program will be more understandable. However, there are three methods to implement operator overloading that are: -
- Member Function
- Non-Member Function
- Friend Function
- Operator overloading function can be a member function if the Left operand is an Object of that class, but if the Left operand is different, then Operator overloading function must be a non-member function. Operator overloading function can be made friend function if it needs access to the private and protected members of class.
- However, we cannot overload some of the operators that are given below: -
- Scope Resolution Operator - `::`
- Member Access or Dot operator - `.`
- Pointer-to-member Operator - `*`
- Ternary or Conditional Operator - `?:`

Aim:

Write a program to add, subtract, multiply and divide two complex number using **Operator Overloading** .

Theory:

Operator overloading is an important concept in C++. It is a type of polymorphism in which an operator is overloaded to give user defined meaning to it. Overloaded operator is used to perform operation on user-defined data type. For example, ‘+’ operator can be overloaded to perform addition on various data types, like for Integer, String(concatenation) etc. Almost any operator can be overloaded in C++. However, there are few operator which can not be overloaded. Operator that are not overloaded are follows :

- Scope Resolution Operator - `::`
- Member Access or Dot operator - `.`
- Pointer-to-member Operator - `*`
- Ternary or Conditional Operator - `?:`

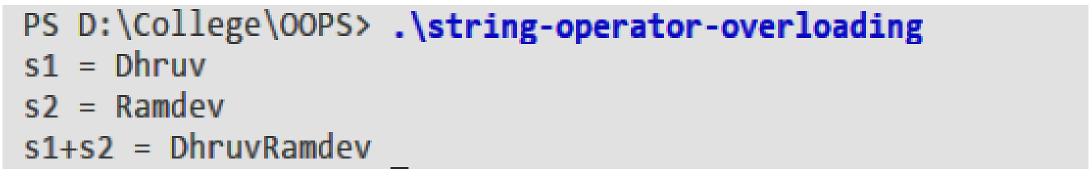
Syntax :

```
data_type classname :: operator symbol (arguments){
    //function body
}
```

Code :

```
#include <iostream>
#include <string.h>

using namespace std;

class String {
    int len ;
    char* str ;

    public :
    String () {
        len = 1 ;
        str = new char ;
        str[0] = '\0' ;

    }
    String (char* s) {
        len = strlen(s) ;
        str = new char[len + 1];
        strcpy( str , s );
    }
    void display(){
        cout << str << endl ;
    }
    String operator + ( String s ) {
        char* t = new char[len + s.len - 1] ;
        strcpy( t , str ) ;
        strcat( t , s.str ) ;
        return String(t) ;
    }

} ;

int main(){

   String s1 = "Dhruv" ;
   String s2 = "Ramdev" ;
   String s3 = s1+s2 ;
   cout << "s1 = " ; s1.display() ;
   cout << "s2 = " ; s2.display() ;
   cout << "s1+s2 = " ; s3.display() ;
    return 0 ;

}
```

Output :



```
PS D:\College\OOPS> .\string-operator-overloading
s1 = Dhruv
s2 = Ramdev
s1+s2 = DhruvRamdev _
```

Discussion :

In the above program we have created a class `String` to concatenate two strings by using operator overloading. We have overload `+` operator for adding two strings. First we created two objects of String
class and initialize two strings for two different objects and then we overload + operator or we can say that we invoke the operator function and then we created a string of third type of object which has
size of string one and string two and concatenate in it by using `strcpy()` function.

Learning Outcomes :

- We have learned that by using overloading operator our program will be more understandable. However, there are three methods to implement operator overloading that are: -
  - Member Function
  - Non-Member Function
  - Friend Function

- Operator overloading function can be a member function if the Left operand is an Object of that class, but if the Left operand is different, then Operator overloading function must be a non-member function. Operator overloading function can be made friend function if it needs access to the private and protected members of class.
- However, we cannot overload some of the operators that are given below: -
- Scope Resolution Operator - `::`
- Member Access or Dot operator - `.`
- Pointer-to-member Operator - `*`
- Ternary or Conditional Operator - `?:`

Aim:

To understand the concept of Multilevel Inheritance and implement it with the help of on example.

Theory:

One of the most important concepts in object-oriented programming is that of inheritance. Inheritance allows us to define a class in terms of another class, which makes it easier to create and maintain an application. This also provides an opportunity to reuse the code functionality and fast implementation time. When creating a class, instead of writing completely new data members and member functions, the programmer can designate that the new class should inherit the members of an existing class. This existing class is called the **base class** , and the new class is referred to as the **derived class** . In Multilevel Inheritance, a derived class can also inherited by another class. For instance, if we inherit Base class in Derived1 class and inherit Derived1 class in Derived2 class. So when a derived is inherited by another class then it creates Multiple Levels and this is known as **Multi-level Inheritance** .

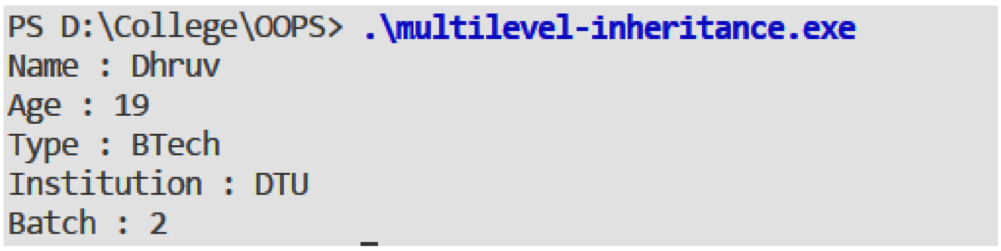
Syntax :

```
class A {
    // Base class
};
class B: public A {
    // B derived from A
};
class C: public B {
   // C derived from B
};
```

Code :

```
#include <iostream>
#include <string.h>
using namespace std ;

class Person {

    protected :
    char name[50] ;
    int age ;

    public :
    Person( char* s , int a ) {
        strcpy( name,s );
        age = a ;
    }

    void display() {
        cout << "Name : " << name << endl ;
        cout << "Age : " << age << endl ;
    }

};

class Student : public Person  {

    protected :
    char type[10] ;
    char clg_name[50] ;

    public :
    Student( char *s  , int a , char *t , char *cn ) : Person( s , a ) {
        strcpy( type , t );
        strcpy( clg_name , cn );
    }

    void display() {
        Person :: display() ;
        cout << "Type : " << type << endl ;
        cout << "Institution : " << clg_name << endl ;
    }

} ;

class BTech : public Student {
    private :
    int batch ;

    public :
    BTech( char *s  , int a , char *t , char *cn , int b  ) : Student( s, a, t ,cn  ){
        batch = b ;
    }

    void display() {
        Student :: display () ;
        cout << "Batch : " << batch <<endl ;
    }
};

int main(){

    BTech var( "Dhruv" , 19 , "BTech" , "DTU" , 2  );
    var.display();

}
```

Output :



```
PS D:\College\OOPS> .\multilevel-inheritance.exe
Name : Dhruv
Age : 19
Type : BTech
Institution : DTU
Batch : 2
```

Discussion :

The above program illustrates the concept of multilevel inheritance. In this program, a `Person` class is made which has marks as a data member and a constructor to accept data. Class `Person` is inherited publically in derived class `Student` which accepts data and calls the constructor of `Person` . `Student` is inherited by `BTech` . This way a multilevel inheritance is carried out.

Learning Outcomes :

- Multilevel inheritance allows user to inherit classes in a level wise order.
- Reusability - facility to use public methods of base class without rewriting the same.
- Extensibility - extending the base class logic as per business logic of the derived class.
- Data Hiding - base class can decide to keep some data private so that it cannot be altered by the derived class
- Overriding -With inheritance, we will be able to override the methods of the base class so that meaningful implementation of the base class method can be designed in the derived class.

Aim:

To understand the concept of Hybrid Inheritance and implement it with the help of on example.

Theory:

One of the most important concepts in object-oriented programming is that of inheritance. Inheritance allows us to define a class in terms of another class, which makes it easier to create and maintain an application. This also provides an opportunity to reuse the code functionality and fast implementation time. When creating a class, instead of writing completely new data members and member functions, the programmer can designate that the new class should inherit the members of an existing class. This existing class is called the base class, and the new class is referred to as the derived class. Hybrid Inheritance is a type of inheritance. It is implemented by combining more than one type of inheritance. For example: Combining Hierarchical inheritance and Multiple Inheritance.

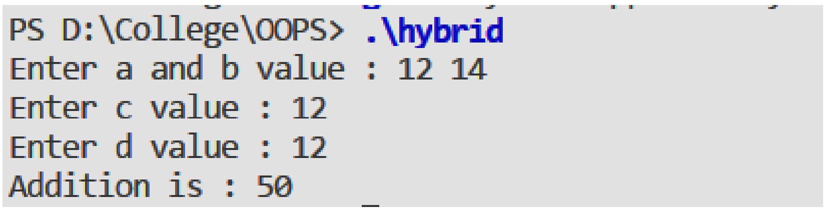
Syntax :

```
class A {

};
class B: public A {

};
class C: {

};
class D: public C public B {

};
```

Code :

```
#include<iostream>
using namespace std;

class A {
    protected:
        int a , b ;
    public:
        void getab() {
            cout<<"Enter a and b value : ";
            cin>>a>>b;
        }
};
class B:public A {
    protected:
        int c ;
    public:
        void getc() {
            cout<<"Enter c value : ";
            cin>>c;
        }
};
class C {
    protected:
        int d ;
    public:
        void getd() {
            cout<<"Enter d value : ";
            cin>>d;
        }
};
class D:public B,public C {
    protected:
        int e ;
    public:
        void result() {
            getab();
            getc();
            getd();
            e=a+b+c+d;
            cout<<"Addition is : "<<e<<endl;
        }
};
int main() {
    D d1;
    d1.result();
    return 0;
}
```

Output :



```
PS D:\College\OOPS> .\hybrid
Enter a and b value : 12 14
Enter c value : 12
Enter d value : 12
Addition is : 50
```

Discussion :

In the above program there are four classes which are showing hybrid inheritance. Classes A and B show multilevel and classes B,C and D show multiple inheritance. So it is a type of hybrid inheritance having
a combination of multilevel and multiple inheritance. However, the invoking method is same as of previous program.

Learning Outcomes :

Inheritance is the major feature of object oriented programming we learn the following terms from inheritance: -

- Reusability : Facility to use public methods of base class without rewriting the same
- Extensibility : Extending the base class logic as per business logic of the derived class

- Data hiding : Base class can decide to keep some data private so that it cannot be altered by the derived class
- Overriding : With inheritance, we will be able to override the methods of the base class so that meaningful implementation of the base class method can be designed in the derived class.