Don Bosco Institute of Technology, Kurla(W)
Department of Electronics and Tele-Communication Engineering
ECL304 - Skill Lab: C++ and Java Programming
Sem III
2021-22

Lab Number:	**5**
Student Name:	**Bhagyesh Subhash Manjrekar**
Roll No :	**07**

Title:

To perform Operator Overloading using C++ for

- multiplying 2 complex numbers
- adding matrices

Learning Objective:

- Students will be able to perform user-defined overloading of built-in operators.

Learning Outcome:

- Understanding the overloading concept on built-in operators.

Course Outcome:

- Comprehend building blocks of OOPs language, inheritance, package and interfaces

Faculty: Ms. Deepali Kayande

Theory:

Explain about operator overloading with respect to:

- Constructor:

  Constructors can be overloaded in a similar way as function overloading. Overloaded constructors have the same name as that of the class but the different number of arguments. Depending upon the number and type of arguments passed, the corresponding constructor will be call at that time.

  A constructor is called depending upon the number and type of arguments passed. While creating the object, arguments must be passed to let compiler know, which constructor needs to be called.

- Methods:

  Overloading by different methods is the process of having two or more function with the same name, but different in parameters in C++. In this, the function is redefined by using either different types of arguments or a different number of arguments. It is only through these differences compiler can differentiate between the functions.

The advantage of Fun it is, increases the readability of the program because you don't need to use different names for the same action.

- Operators:

  C++ provides a special function to change the current functionality of some operators within its class which is often called as operator overloading. Operator Overloading is the method by which we can change the function of some specific operators to do some different task.

  In C++, we can change the way operators work for user-defined types like objects and structures. This is known as operator overloading.

  For example, suppose we have created three object and result from a class that represents something in that code. Since operator overloading allows us to change how operators work, e.g we can redefine how the + operator works and use it to add the complex numbers.

Syntax for C++ Operator Overloading:

To overload an operator, we use a special operator function. We define the function inside the class or structure whose objects/variables we want the overloaded operator to work with.

```
class className
{
    public
      returnType operator symbol (arguments)
      {
       }
}
```

Here,

return-Type is the return type of the function.
operator is a keyword.
symbol is the operator we want to overload.
argument is the argument passed to the function.

Types of Operator overloading:

(1) Operator Overloading in Unary Operators:

Unary operators operate on only one operand. The increment operator (+) and decrement operator (-) are examples of unary operators.

(2) Operator Overloading in Binary Operators:

Binary operators work on two operands.

for example,
c =a+b;
Here, + is a binary operator that works on the operands a and b.

When we overload the binary operator for user-defined types by using the code:

obj3 = obj1 + obj2;

The operator function is called using the obj1 object and obj2 is passed as an argument to the function.

Operator overloading cannot change the precedence and associativity of operators. However, if we want to change the order of evaluation, parentheses should be used.

There are 4 operators that cannot be overloaded in C++.
They are:

1. :: (scope resolution)
2. . (member selection)
3. .* (member selection through pointer to function)
4. ?: (ternary operator)

PROGRAM 1:

Perform Operator Overloading using C++ for multiplying 2 complex numbers.

Algorithm :

Step 1: start

Step 2: create class complex

Step 3: create multiply function to multiply two complex numbers

Step 4: Overloading increment operator to increment complex number

Step 5: creating objects for two complex numbers and taking values from the user and printing it

Step 6: printing the output as multiplication of entered numbers

Step 7: exit

Program:

```
//To perform Operator Overloading using C++ for Multiplying 2 complex numbers


#include<iostream>


using namespace std;
```

```
class Complex
{
        public:
                int real;
                int img;

        //multiply function to multiply two complex numbers
   int multiply(Complex c1,Complex c2)
   {
     int x,y;
     x=(c1.real*c2.real)-(c1.img*c2.img);
     y=(c1.real*c2.img)+(c1.img*c2.real);

cout<<"\n("<<c1.real<<"+"<<c1.img<<"i)*("<<c2.real<<"+"<<c2.img<<"i)=("<<x<<"+"<<
y<<"i)";
   }
   //Overloading increment operator to increment complex number
   Complex operator ++()
  {
    Complex x;
    x.real=++real;
    x.img=++img;
    return x;
  }

};

int main ()
```

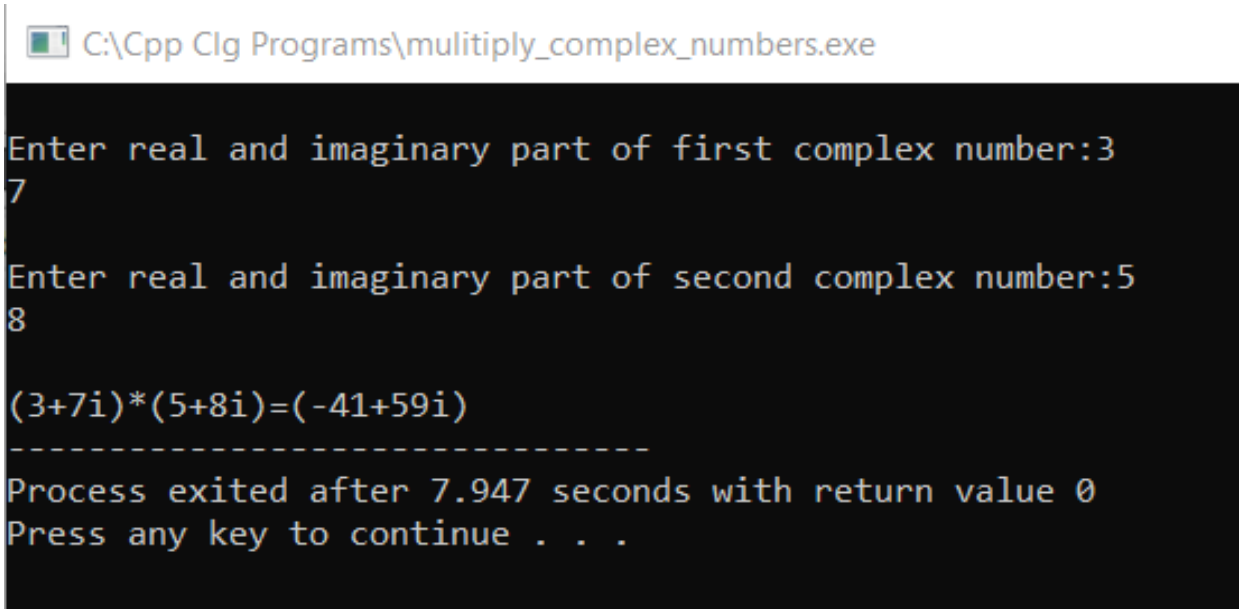
```
{

Complex a,b,d;
cout<<"\nEnter real and imaginary part of first complex number:";
cin>>a.real>>a.img;
cout<<"\nEnter real and imaginary part of second complex number:";
cin>>b.real>>b.img;

d.multiply(a,b);



return 0;
}
```

Input given:

1st complex number: 3+7i

2nd complex number: 5+8i

Output Screenshot:



PROGRAM 2:

Perform Operator Overloading using C++ for adding two matrices.

Algorithm:

Step 1: start

Step 2: crate class matrices

Step 3: create 3 matrices for 2*2 matrix

Step 4: creating get_elements() function, operator overloading, print the result as display sunction.

Step 5: create functions outside class, using scope resolution

Step 6: creating objects, displaying matrices, and printing the result as addition of two matrices.

Step 7: exit

Program:

//Write a C++ program to overload the '+' operator so that it can add two matrices.

```
# include<iostream>

using namespace std;

class matrices
{
        int x[2][2];
        int y[2][2];
        int z[2][2];
```

```
        public:
    void get_elements();      //take numbers from user
    matrices operator +(matrices m2);        //operator overloading
    void display();    //print the result
};
//functions outside class, using scope resolution
void matrices::get_elements()
{
  cout<<"Enter the elements for matrix:"<<endl;

  for(int i=0;i<2;i++)  //for row
  {
      for(int j=0;j<2;j++)        //for columns
      cin>>x[i][j];
  }

}

void matrices:: display()
{
  for(int i=0;i<2;i++)
   {
      for(int j=0;j<2;j++)
      cout<<x[i][j]<<"  ";
      cout<<endl;
        }

}
```

```
matrices matrices::operator+(matrices m2)
{
   matrices m3;
   for(int i=0;i<3;i++)
         {
      for(int j=0;j<2;j++)
      m3.x[i][j]=x[i][j]+m2.x[i][j];

         }

         return(m3);

}
int main()
{
         matrices ob1,ob2;
         ob1.get_elements();
         ob2.get_elements();
         cout<<"\nMatrix 1:\n";
         ob1.display();
         cout<<"\nMatrix 2:\n";
         ob2.display();
         ob1=ob1+ob2;
         cout<<"\nAddition of two matrices:\n";
         ob1.display();

}
```

Input given:

```
Enter the elements for matrix:
5
9
-7
8
Enter the elements for matrix:
45
20
13
2
```

Output Screenshot:

```
C:\Cpp Clg Programs\add_two_matices_.exe
Enter the elements for matrix:
5
9
-7
8
Enter the elements for matrix:
45
20
13
2

Matrix 1:
5  9
-7  8

Matrix 2:
45  20
13  2

Addition of two matrices:
50  29
6  10

--------------------------------
Process exited after 12.79 seconds with return value 0
Press any key to continue . . .
```