Lab Number:	**5**
Student Name:	**Shreyas Sanjay Nanaware**
Roll No :	**39**

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:

ECL304.2	Comprehend building blocks of OOPs language, inheritance, package and interfaces

Theory:

In simple terms , to overload is to load an **excessive** amount in or on something, such as an overload of electricity which shorts out the circuits. Overloading causes a "Too much!" situation. To overload is to push something or someone too far. Like for example , a supervisor can **overload** an employee by assigning too much work. When two or more methods in the same class have the same name but different parameters, it's called Overloading.

Constructor overloading:

In C++, We can have more than one constructor in a class with same name, as long as each has a different list of arguments. This concept is known as

Constructor Overloading and is quite similar to function overloading. ... A constructor is called depending upon the number and type of arguments passed. The constructor overloading can be defined as the concept of having more than one constructor with different parameters so that every constructor can perform a different task.

Method overloading:

C++ allows you to specify more than one definition for a function name or an operator in the same scope, which is called function overloading and operator overloading respectively.
An overloaded declaration is a declaration that is declared with the same name as a previously declared declaration in the same scope, except that both declarations have different arguments and obviously different definition (implementation).
When you call an overloaded function or operator, the compiler determines the most appropriate definition to use, by comparing the argument types you have used to call the function or operator with the parameter types specified in the definitions. The process of selecting the most appropriate overloaded function or operator is called overload resolution.

Operator overloading:

In computer programming, operator overloading, is a specific case of **polymorphism**, where different operators have different implementations depending on their arguments. **Operator overloading** is generally defined by a programming language, a programmer, or both.
Operator Overloading Syntax:
Syntax:

```
return_type operator operator_symbol (parameters)
{
statement1;
statement2;
}
```

The keyword operator defines a new action or operation to the operator

For example, we can overload an operator '+' in a class like String so that we can concatenate two strings by just using +.

. It is used to perform the operation on the user-defined data type.

1) **Write a C++ program for multiplying two complex numbers using the concept of operator overloading.**

Algorithm :

1) Create a class and declare two variables representing the real and the imaginary part of the complex number.
2) Take two complex numbers from the user and store them in order to perform their multiplication.
3) Use operator overloading for multiplication.
4) Display the multiplication of the two complex numbers as output.

Program:

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

#include<iostream>
using namespace std;
class Complex   //creating the class complex
{
int real, imag;   //declaring the variables
public:   //Access Specifier
Complex operator*(Complex c)   //overloading the multiplication operator
{
```

```
int real1,real2;
real1=real;
real2=c.real;
real=(real*c.real)-(imag*c.imag);     //calculation part
imag=(real1*c.imag)+(imag*real2);     //calculation part

Complex temp;
temp.real=real;
temp.imag=imag;
return temp;
}
void display()   //method for displaying the output
{
cout<<"("<<real << " + " <<imag<<"i"<< ")"<<" \n";
}
void input()   //method for taking input
{
cout<<"Enter Real part: \n";   //asking for user input
cin>>real;          //storing the user input
cout<<" Enter Imaginary part: \n";    //asking for user input
cin>>imag;    //storing the user input
}
};
int main()
{
```

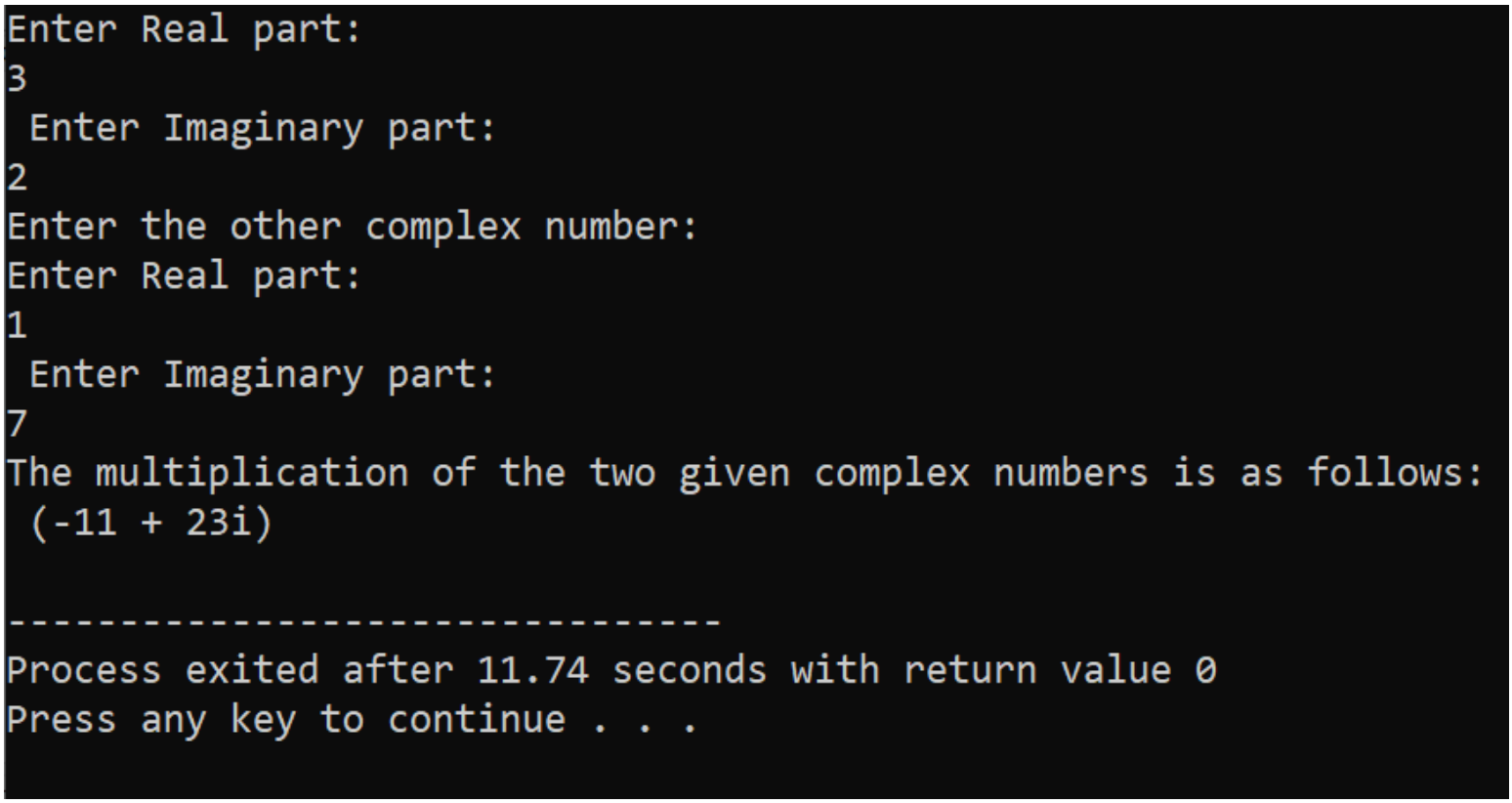
```
Complex c1,c2;     //creating the objects of the class
c1.input();    //calling the methods
cout<<"Enter the other complex number: "<<"\n";
c2.input();    //calling the methods
c1*c2;
cout<<"The multiplication of the two given complex numbers is as follows: \n
";
c1.display();  //calling the methods


}
```

Input given:

First complex number : 3+2i

Second Complex number : 1+7i

Output Screenshot:



2) Write a C++ program to add two matrices using operator overloading.

Algorithm:

1. Create a class and declare 3 variables for 3 matrices with their dimensions.
2. Take the elements of the matrix as the input from the user.
3. Add two matrices using operator overloading

   Resultant[i][j]=Matrix 1[i][j] + Matrix 2[i][j];

4. Display result sum[i][j].

Program:

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

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

class matrices  //creating a class
{
    int x[3][3];   //class attributes
    int y[3][3];   //class attributes
    int z[3][3];   //class attributes
    public:       //access specifier
        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()  //method for getting the input elements
{
    cout<<"Enter the  Matrix elements \n";
    for(int i=0;i<3;i++)         //for row
    {
        for(int j=0;j<3;j++)         //for columns
        cin>>x[i][j];
    }
}

void matrices:: display()   //method to display the resultant matrix
{
    for(int i=0;i<3;i++)

    {
        for(int j=0;j<3;j++)
        cout<<x[i][j]<<"  ";

        cout<<endl;
    }
}
```

```
matrices matrices::operator+(matrices m2)  //overloading '+' operator

{

        matrices m3;

        for(int i=0;i<3;i++)

        {
                for(int j=0;j<3;j++)
                m3.x[i][j]=x[i][j]+m2.x[i][j];

        }
        return(m3);
}

int main()
{
        matrices object1,object2,object3;
        object1.get_elements();   //calling the methods
        object2.get_elements();    //calling the methods
        cout<<"\nMatrix 1 is as follows :\n";
        object1.display();       //calling the methods
        cout<<"\nMatrix 2 is as follows :\n";
        object2.display();       //calling the methods
```

```
        object3=object1+object2;
        cout<<"\nResultant Matrix is as follows :\n";


        object3.display();      //calling the methods
}
```

Input Given:

Matrix 1:

1 2 3

4 5 6

7 8 9

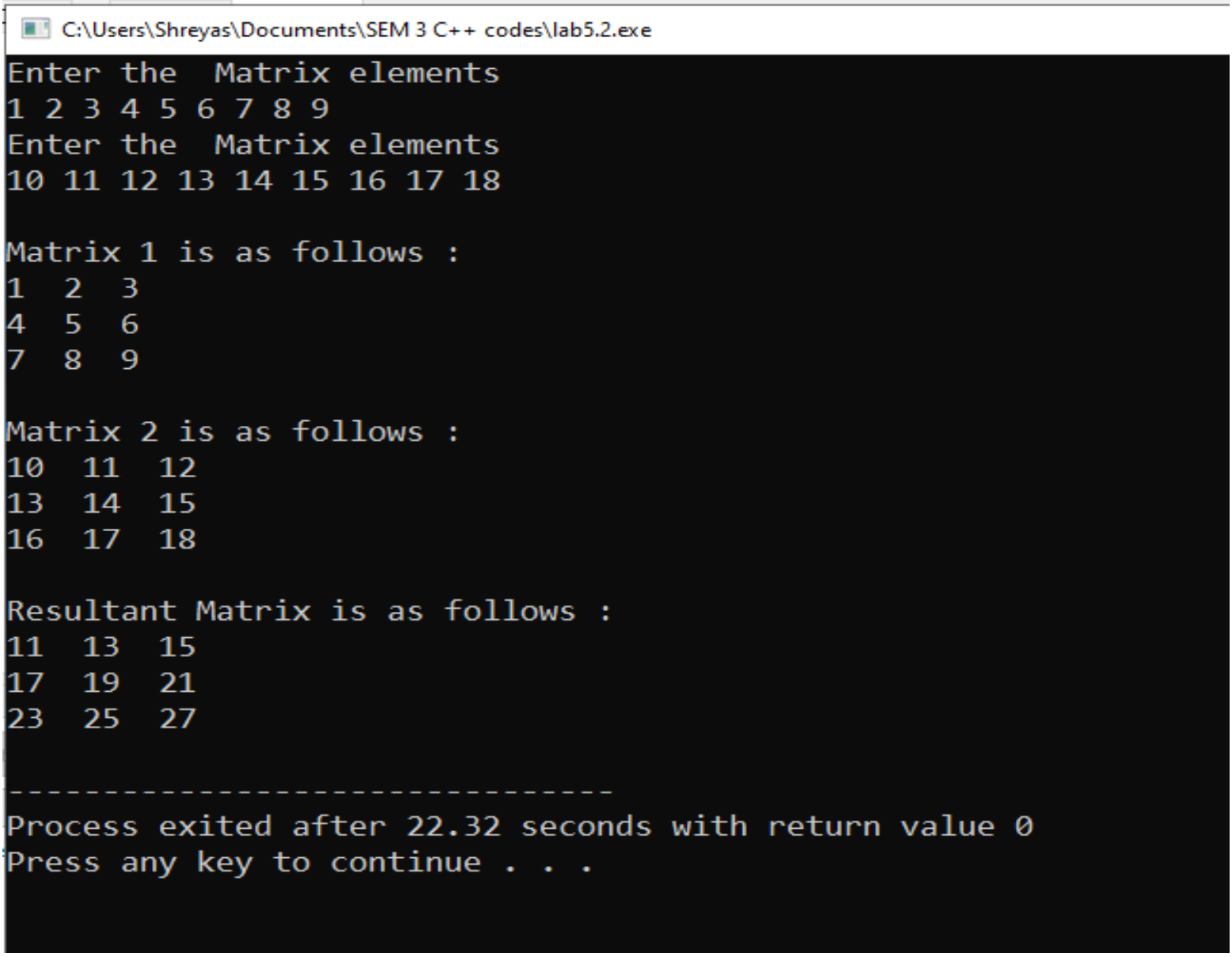
Matrix 2:

10 11 12

13 14 15

16 17 18

Output:


C:\Users\Shreyas\Documents\SEM 3 C++ codes\lab5.2.exe
Enter the Matrix elements
1 2 3 4 5 6 7 8 9
Enter the Matrix elements
10 11 12 13 14 15 16 17 18
Matrix 1 is as follows :
1 2 3
4 5 6
7 8 9
Matrix 2 is as follows :
10 11 12
13 14 15
16 17 18
Resultant Matrix is as follows :
11 13 15
17 19 21
23 25 27

Process exited after 22.32 seconds with return value 0
Press any key to continue . . .