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:	**Sarika Laxmikant Galphade**
Roll No :	**36**

Title:

To perform Operator Overloading using C++ for

- adding 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:

Explain about operator overloading with respect to:

- constructor,
- methods and
- operators.

Constructor : Operator overloading in c++ is one of the best features that is used to overload most of the operators like “+” “–” “*” “/” “=” “.” “,” etc in c++.

- Overloaded constructors essentially have the same name (exact name of the class) and differ by number and type of arguments.
- 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:

Method overloading is the process of overloading the method that has the same name but different parameters. C++ provides this method of overloading features. Method overloading allows users to use the same name to another method, but the parameters passed to the methods should be different.

Faculty: Ms. Deepali Kayande

Operators:
In C++, we can make operators to work for user defined classes. This means C++ has the ability to provide the operators with a special meaning for a data type, this ability is known as operator overloading. For example, we can overload an operator '+' in a class like String so that we can concatenate two strings by just using +.

adding 2 complex numbers

Algorithm :	Step 1: Start the program. Step 2: Create a class as complex Step 3: Create a constructor and perform addition of two numbers. Step 4: Use destruction fuction. Step 5: End of the program

Program:	# include<iostream>
	using namespace std;
	
	class complex
	{
	    float real;
	    float img;
	
	public:
	void get_elements(); //take numbers from user
	complex operator *(complex c1); //operator overloading
	void display(); //print the result
	};
	void complex::get_elements()
	{
	        cout<<"Enter the real and img of complex no.\n";
	        cout<<"Real :";

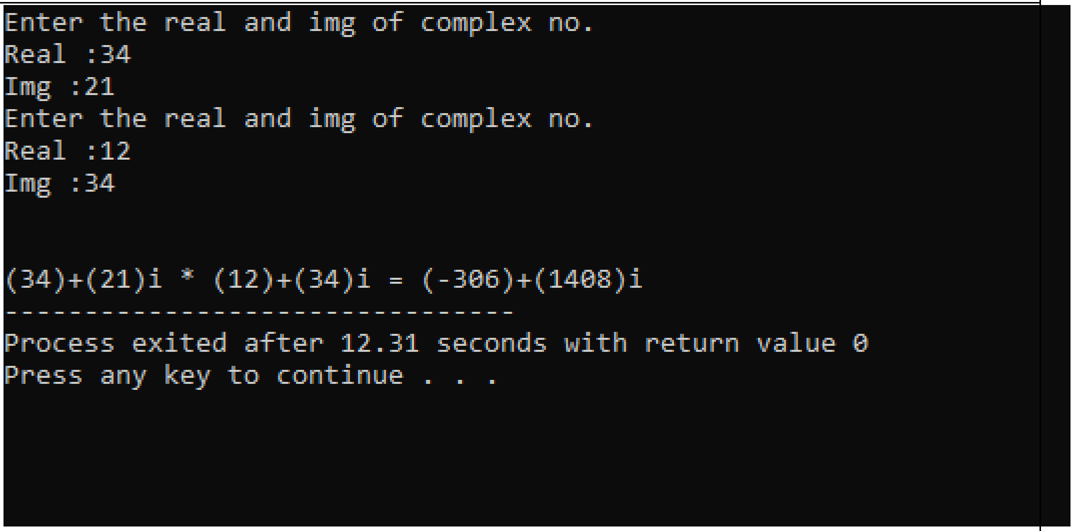
```
            cin>>real;
            cout<<"Img :";
            cin>>img;


}
void complex::display()
{
cout<<"("<<real<<")"<<"+"<<"("<<img<<")"<<"i";
}


complex complex::operator*(complex c1)
{
            complex mul;
   mul.real = ((real*c1.real)-(img*c1.img));
   mul.img = ((real*c1.img)+(c1.real*img));
     return(mul);
}
int main()
{
      complex obj1,obj2,obj3;
     obj1.get_elements();
    obj2.get_elements();
    obj3= obj1*obj2;


    cout<<"\n\n";
    obj1.display();
    cout<<" * ";
    obj2.display();
```

	`cout<<" = ";`
	`obj3.display();`
	
	`}`

Input given:	Real:34 Img: 21 Real: 12 Img: 34
Output Screenshot :	

```
Enter the real and img of complex no.
Real :34
Img :21
Enter the real and img of complex no.
Real :12
Img :34


(34)+(21)i * (12)+(34)i = (-306)+(1408)i
--------------------------------
Process exited after 12.31 seconds with return value 0
Press any key to continue . . .
```

adding matrices

Algorithm :	1. Define functions for get_matrix(), display_matrix(), and overload the '+' operator. 2. Take user input for matrices. 3. Decide on two variables of the Matrix type. 4. Use the get_matrix() function to receive the matrix 5. Use the display_matrix() function to display the matrices. 6. Add them using the overloaded '+' operator. 7. Print the result.

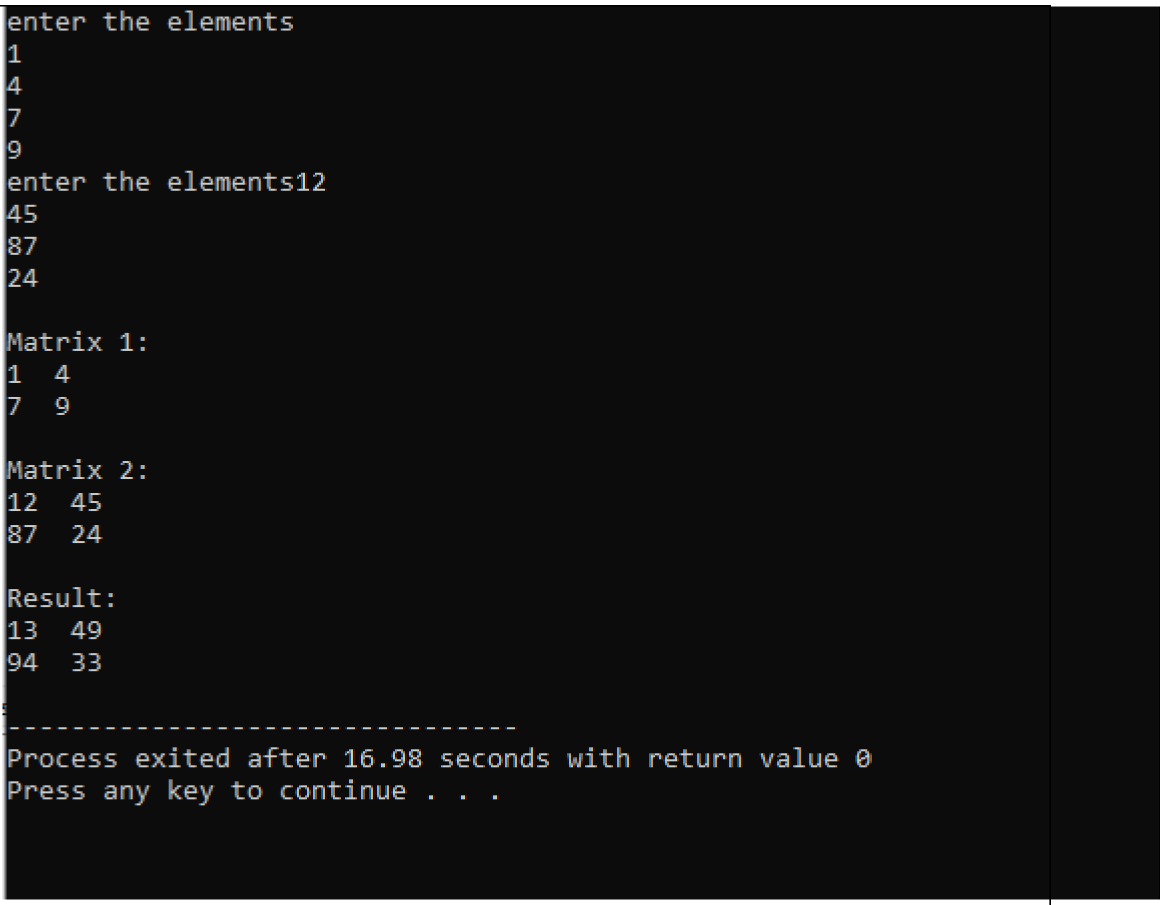
Program:	`# include<iostream>`
	`using namespace std;`
	`class matrices`
	`{`

```
int a[2][2];
int b[2][2];
int c[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(int i=0;i<2;i++)     //for row
{
for(int j=0;j<2;j++)     //for columns
cin>>a[i][j];
}
}
void matrices:: display()
{
for(int i=0;i<2;i++)
{
for(int j=0;j<2;j++)
cout<<a[i][j]<<"  ";
cout<<endl;
}
}
```

	```
	matrices matrices::operator+(matrices m2)
	{
	matrices m3;
	for(int i=0;i<2;i++)
	{
	for(int j=0;j<2;j++)

	m3.a[i][j]=a[i][j]+m2.a[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<<"\nResult:\n";
	ob1.display();
	}
	```
Input given:	**1 4 7 9** **12 45 87 24**

Output Screenshot:	

```
enter the elements
1
4
7
9
enter the elements12
45
87
24

Matrix 1:
1  4
7  9

Matrix 2:
12  45
87  24

Result:
13  49
94  33

--------------------------------
Process exited after 16.98 seconds with return value 0
Press any key to continue . . .
```