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:	**02**
Student Name:	SHIVAM SHARMA
Roll No :	E-19

Title:

To perform operator overloading using C++ for multiplying two complex numbers.

Learning Objective:

- Students will be able to write C++ and java program for operator overloading and take input from user.

Learning Outcome:

- Ability to execute a simple C++ and Java program with and without any inputs tothe program.
- Understanding the overloading in C++ and Java.

Theory:

C++ allows specification of more than one function of the same name in the same scope. These functions are called *overloaded* functions. Overloaded functions enable you to supply different semantics for a function, depending on the types and number of arguments.

For example, a print function that takes a std::string argument might perform very different tasks than one that takes an argument of type double. Overloading saves you from having to use names such as print_string or print_double. At compile time, the compiler chooses which overload to use based on the type of arguments passed in by the caller. If you call print(42.0), then the void print(double d) function will be invoked. If you call print("hello world"), then the void print(std::string) overload will be invoked.
You can overload both member functions and non-member functions. The following table shows what parts of a function declaration C++ uses to differentiate between groups of functions with the same name in the same scope.

Function overloading is a feature of object oriented programming where two or more functions can have the same name but different parameters.
When a function name is overloaded with different jobs it is called Function Overloading.
In Function Overloading "Function" name should be the same and the arguments should be different.
Function overloading can be considered as an example of polymorphism feature in C++.

Function overloading is one of the important features of object-oriented programming. It allows users to have more than one function having the same name but different properties. Overloaded functions enable users to supply different semantics for a function, depending on the signature of functions.

Advantages of function overloading are as follows:

- The main advantage of function overloading is that it improves code readability and allows code reusability.
- The use of function overloading is to save memory space, consistency, and readability.
- It speeds up the execution of the program
- Code maintenance also becomes easy.
- Function overloading brings flexibility to code.
- The function can perform different operations and hence it eliminates the use of different function names for the same kind of operations.
-

Disadvantage of function overloading are as follows:

- Function declarations that differ only in the return type cannot be overloaded

Why is function overloading in C++ is used?

Function overloading is similar to polymorphism that helps us to get different behaviour, with the same name of the function. Function overloading in C++ is used for code reusability and to save memory.

Algorithm :

STEP 1: Start
STEP 2: Define functions for get_matrix(), display_matrix(), and overload the '+' operator.
STEP 3: Take user input for matrices.
STEP 4: Decide on two variables of the Matrix type.
STEP 5: Use the get_matrix() function to receive the matrix
STEP 6: Use the display_matrix() function to display the matrices.
STEP 7: Add them using the overloaded '+' operator.
STEP 8: End

INPUT

```
# include<iostream>
using namespace std;
class matrices
{
int a[2][2];
int b[2][2];
i 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();
```

OUTPUT: