Chittagong University of Engineering and Technology

Department of Electronics and Telecommunication Engineering

Name of the Experiment :

Operator Overloading in C++.

Course No. : CSE 284

Course Title : Object Oriented Programming

Experiment No. : 07

Date of Experiment : 04-12-2024

Date of Submission : 18-12-2024

Remarks	Submitted by,
	Name : Belayet Hossain Saimun **ID** : 2108032 **Level** : 2 **Term** : 2 **Group** : 2

Name of the Experiment: Operator Overloading in C++.

Objectives:

1. To understand operator overloading in C++.
2. To implement operator overloading using the Friend function.

Problem Statement-01: Define a class Distance with distances in feet and inch and with a print function to print the distance.
a) overload < operator to compare two distances using member function.
b) overload + operator to add two Distances using friend function.

Code:

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

class Distance
{
private:
    int feet;
    int inch;

public:
    Distance()
    {
        feet = 0;
        inch = 0;
    }

    void getData()
    {
        cout << "Enter Distance in feet: ";
        cin >> feet;
        cout << "Enter Distance in inch: ";
        cin >> inch;
    }

    bool operator<(Distance d)
    {
        if (feet < d.feet)
        {
            return true;
        }
        if (feet == d.feet && inch < d.inch)
        {
            return true;
        }

        return false;
    }

    void print()
```

```
    {
        cout << "Summation of this two distance is: " << feet << " Feet
    " << inch << " Inch" << endl;
    }
    friend Distance operator+(Distance, Distance);
};

Distance operator+(Distance d1, Distance d2)
{
    Distance temp;
    temp.feet = d1.feet + d2.feet;
    temp.inch = d1.inch + d2.inch;
    return temp;
}

int main()
{
    Distance d1, d2, result;

    cout << "First Distance:" << endl;
    d1.getData();
    cout << "Second Distance:" << endl;
    d2.getData();

    if (d1 < d2)
    {
        cout << "First Distance is less than Second Distance" << endl;
    }
    else
    {
        cout << "First Distance is greater than Second Distance" <<
   endl;
    }

    result = d1 + d2;
    result.print();
}
```

Output:

```
"B:\BS_32 Codes\CSE_284

First Distance:
Enter Distance in feet: 22
Enter Distance in inch: 6
Second Distance:
Enter Distance in feet: 10
Enter Distance in inch: 12
First Distance is greater than Second Distance
Summation of this two distance is: 32 Feet 18 Inch

Process returned 0 (0x0)   execution time : 22.483 s
Press any key to continue.
```

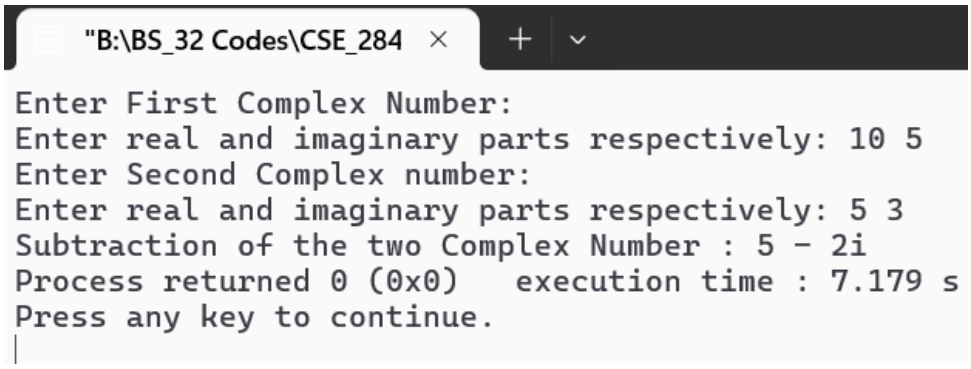
Problem Statement-02: Write a C++ program to Overloaded operator to subtract two complex number.

Code:

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

class Complex
{
private:
    int real;
    int imag;

public:
    Complex()
    {
        real = 0;
        imag = 0;
    }

    void input()
    {
        cout << "Enter real and imaginary parts respectively: ";
        cin >> real;
        cin >> imag;
    }

    Complex operator-(Complex c)
    {
        Complex temp;
        temp.real = real - c.real;
        temp.imag = imag - c.imag;
        return temp;
    }

    void output()
    {
        if (imag < 0)
            cout << "Subtraction of the Complex Number: " << real << "+
   " << imag << "i";
        else
            cout << "Subtraction of the two Complex Number : " << real
   << " - " << imag << "i";
    }
};

int main()
{
    Complex c1, c2, result;

    cout << "Enter First Complex Number:\n";
    c1.input();
    cout << "Enter Second Complex number:\n";
    c2.input();
```

```
    result = c1 - c2;
    result.output();
}
```

Output:



Discussion:

In this experiment, I learned the fundamental concepts of operator overloading in C++ through various examples. Implementing overloaded operators such as + and ++ gave practical experience in simplifying operations for user-defined types, such as adding complex numbers or customizing increment operations. I also learned to use member and friend functions for overloading, which helped to understand the importance of encapsulation and access control in object-oriented programming.