Experiment Name :

Operator Overloading in C++.

Objectives:

- Understand operator overloading in C++ .
- Implement operator overloading using Friend function.

Example-1 :

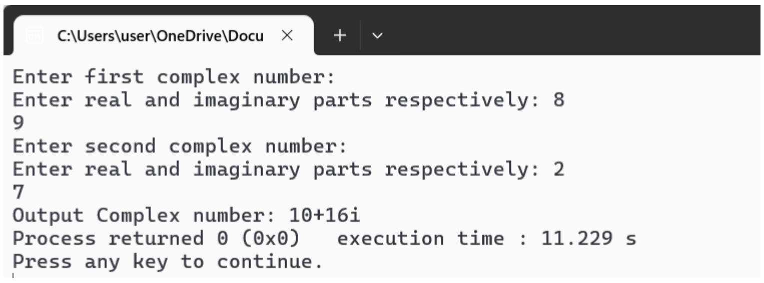
A C++ program to find the sum of two complex numbers using binary operator overloading.

Input:

```
#include <iostream>
using namespace std;
class Complex {
private:
    float real;
    float 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 << "Output Complex number: " << real << imag << "i";
    else
           cout << "Output 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();
    return 0;
}
```

Output :



```
Enter first complex number:
Enter real and imaginary parts respectively: 8
9
Enter second complex number:
Enter real and imaginary parts respectively: 2
7
Output Complex number: 10+16i
Process returned 0 (0x0)   execution time : 11.229 s
Press any key to continue.
```

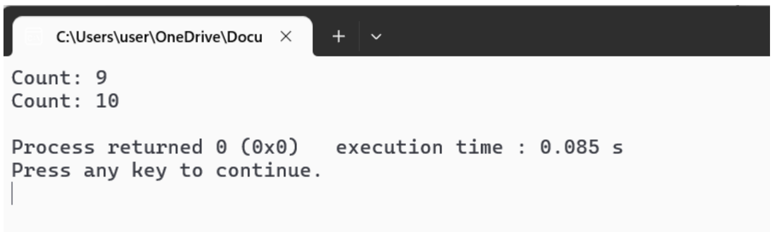
Example-2 :

++ Operator (Unary Operator) Overloading.

Input:

```
#include <iostream>
using namespace std;
class  Count{
private:
    int value;
public:
    Count()
    {
        value = 8;
    }
    void operator ++ (){
        ++value;
    }
    void operator ++ (int){
        value++;
    }
    void display(){
        cout<< "Count: "<<value<<endl;
    }
};
int main()
{
    Count c1;
    c1++;
    c1.display();
    ++c1;
    c1.display();
    return 0;
}
```

Output :



C:\Users\user\OneDrive\Docu

```
Count: 9
Count: 10

Process returned 0 (0x0)   execution time : 0.085 s
Press any key to continue.
```

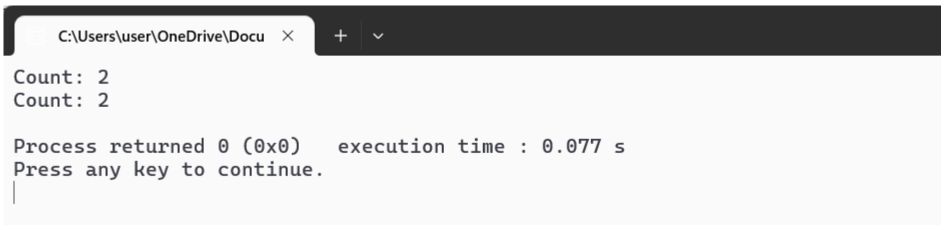
Example-3 :

Return Value from Operator Function (++ Operator).

Input:

```
#include <iostream>
using namespace std;
class  Count{
private:
    int value;
public:
    Count()
    {
        value = 8;
    }
    void operator ++ (){
        ++value;
    }
    void operator ++ (int){
        value++;
    }
    void display(){
        cout<< "Count: "<<value<<endl;
    }
};
int main()
{
    Count c1;
    c1++;
    c1.display();
    ++c1;
    c1.display();
    return 0;
}
```

Output :



```
Count: 2
Count: 2

Process returned 0 (0x0)   execution time : 0.077 s
Press any key to continue.
```

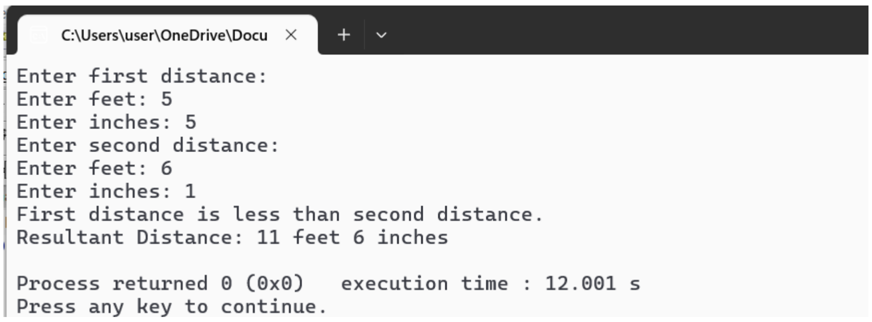
Practice Exercise: 1

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.

Input:

```
#include <iostream>
using namespace std;
class Distance {
public:
    int feet, inch;
    Distance() {
        feet = 0;
        inch = 0;
    }
    Distance(int f, int i) {
        feet = f;
        inch = i;
    }
    void input() {
        cout << "Enter feet: ";
        cin >> feet;
        cout << "Enter inches: ";
        cin >> inch;
        if (inch >= 12) {
            feet += inch / 12;
            inch %= 12;
        }
    }
    Distance operator+(Distance& d2) {
        Distance temp;
        temp.feet = feet + d2.feet;
        temp.inch = inch + d2.inch;
        if (temp.inch >= 12) {
            temp.feet += temp.inch / 12;
            temp.inch %= 12;
        }
        return temp;
    }
    bool operator<(const Distance& d) const {
        int totalInches1 = feet * 12 + inch;
        int totalInches2 = d.feet * 12 + d.inch;
        return totalInches1 < totalInches2;
    }
    void print() const {
        cout << feet << " feet " << inch << " inches" << endl;
    }
};
int main() {
    Distance d1, d2;
    cout << "Enter first distance:" << endl;
    d1.input();
    cout << "Enter second distance:" << endl;
    d2.input();
    if (d1 < d2)
        cout << "First distance is less than second distance." << endl;
    else
        cout << "First distance is not less than second distance." << endl;
        Distance temp = d1 + d2;
    cout << "Resultant Distance: ";
    temp.print();
    return 0;
}
```

Output :



```
Enter first distance:
Enter feet: 5
Enter inches: 5
Enter second distance:
Enter feet: 6
Enter inches: 1
First distance is less than second distance.
Resultant Distance: 11 feet 6 inches

Process returned 0 (0x0)   execution time : 12.001 s
Press any key to continue.
```

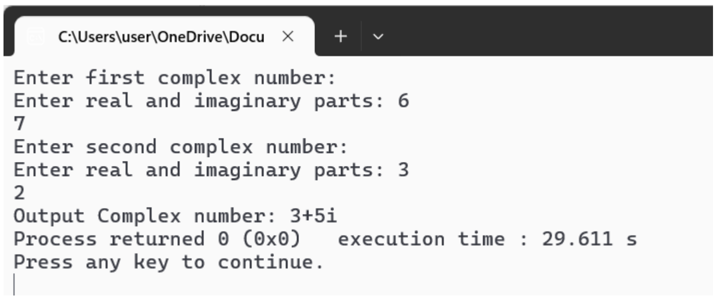
Practice Exercise: 2

Write a C++ program to Overloaded operator to subtract two complex number.

Input:

```
#include<iostream>
using namespace std;
class Complex {
private:
    float real;
    float imag;
public:
    Complex()
    {
          real=0;
          imag=0;
    }
    void input() {
        cout<<"Enter real and imaginary parts: ";
        cin>>real>>imag;
    }
    Complex operator-(Complex c2) {
        Complex temp;
        temp.real = real - c2.real;
        temp.imag = imag - c2.imag;
        return temp;
    }
    void output() {
      if (imag < 0)
       cout<<"Output Complex number: "<< real << imag << "i";
      else
       cout<<"Output 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();
    return 0;
}
```

Output :



```
Enter first complex number:
Enter real and imaginary parts: 6
7
Enter second complex number:
Enter real and imaginary parts: 3
2
Output Complex number: 3+5i
Process returned 0 (0x0)   execution time : 29.611 s
Press any key to continue.
```

Discussion:

In this experiment above, operator overloading was introduced. Here, operator overloading was implemented using friend and member function. It is observed that operator overloading is a compile-time polymorphism in which the operator is overloaded to provide the special meaning to the user-defined data type. It is seen that unary operators are overloaded through a member function, took no explicit arguments. Friend function couldn't be used to overload certain operators but the member function used to overload those operators. While solving the problems, I faced a little difficulty when overloading through a friend function. But soon the concept was understood and so fixing the errors, the problems were solved as well.