PP Lab Name: Shilpi Biswal
PRN: 22070122198

Practical No 10 **Date:** 12 /10/2023

Title: Write a C++ menu driven program to implement unary operator overloading (prefix and postfix increment operator) and binary operator overloading (+ and <).

Description:

In C++, we can change the way operators work for user-defined types like objects and structures. This is known as **operator overloading**. Using operator overloading in C++, you can specify more than one meaning for an operator in one scope. The purpose of operator overloading is to provide a special meaning of an operator for a user-defined data type.

With the help of operator overloading, you can redefine the majority of the C++ operators. You can also use operator overloading to perform different operations using one operator.

To overload a C++ operator, you should define a special function inside the Class as follows:

```
class class_name
{
    ... .. ...
    public
       return_type operator symbol (argument(s))
       {
           ... .. ...
       }
    ... .. ...
};
```

Here is an explanation for the above syntax:

- The return_type is the return type for the function.
- Next, you mention the operator keyword.
- The symbol denotes the operator symbol to be overloaded. For example, +, -, <, ++.
- The argument(s) can be passed to the operator function in the same way as functions.

Hint:

Menu:
Press 1 to apply prefix increment operator on the object of class complex
Press 2 to apply postfix increment operator on the object of class complex
Press 3 to add two objects of class complex (using friend function)
Press 4 to compare two complex numbers using < operator

Overloading unary operator using member function:

Return_type operator op (arg list)
{

}
Operator overloaded functions can be invoked by expressions:
op x;
Or
x op;

Overloading unary operator using friend function:

friend return_type operator op (class_name &obj_name);

Binary operator overloading to add two complex numbers using member function:

```
complex operator +(complex m)
{
complex temp;
temp.real=real+m.real;
temp.imag=imag+m.imag;
return (temp);
}
```

Operator overloaded functions can be invoked by expressions:
c3=c1+c2; //c1,c2,c3 are objects of class complex, this statement is equivalent to
c3=c1.operator+(c2);

Binary operator overloading to add two complex numbers using friend function:

Friend function can also be used in place of member functions for overloading a binary operator, the only difference being that a friend function requires two arguments to be explicitly passed to it, while a member function requires only one.
In the complex number program discussed in the previous section, the statement:

c3=c1+c2;

is equivalent to:

c3= operator+(c1,c2);

Binary operator overloading to compare two complex numbers using member function:

```
bool operator < (complex c)
{…
…
}
```

Operator overloaded functions can be invoked by expression like:
If (c1<c2) // c1 and c2 are the objects

Program Code:

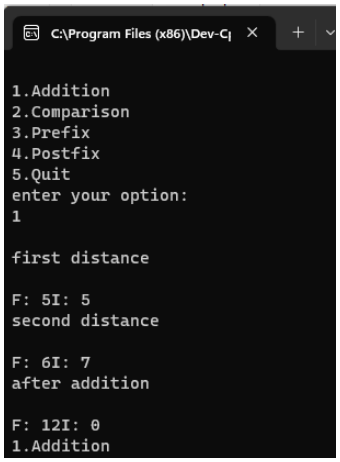
```
#include<iostream>
using namespace std;
class Distance
{
        int feet;
```

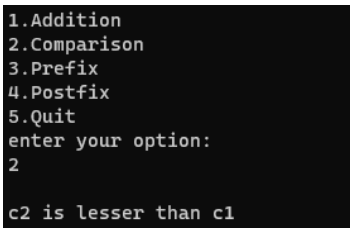
```
        int inch;
public:
        Distance(){
                feet=0;
                inch=0;
        };
        Distance(int a,int b){
                feet=a;
                inch=b;
        }
        Distance operator ++(){

                if(inch>=12){
                        inch=0;
                        ++feet;

                }
                else{
                    ++feet;
                    ++inch;
                }

        }
        Distance operator ++(int){
                if(inch>=12){
                        inch=0;
                        feet++;
                }
                else{
                        feet++;
                        inch++;
                }
        }
        Distance operator +(Distance m){
                Distance temp;
                temp.feet=feet+m.feet;
                temp.inch=inch+m.inch;
                if(temp.inch>=12){
                        temp.feet++;
                        temp.inch=0;
```

```
		}
		return temp;
	}
	bool operator <(Distance m){
		if(m.feet>feet){
			return true;
		}
		if(m.feet==feet && m.inch>inch){
			return true;
		}
	}
	void display(){
		cout<<"\nF: "<<feet<<"I: "<<inch;
	}
};
int main()
{
	Distance c1(5,5);
	Distance c2(6,7);
	Distance c3;
	int opt;
	while(true){
		cout<<"\n1.Addition";
		cout<<"\n2.Comparison";
		cout<<"\n3.Prefix";
		cout<<"\n4.Postfix";
		cout<<"\n5.Quit";
		cout<<"\nenter your option:"<<endl;
		cin>>opt;
		switch(opt){
				case 1:
				cout<<"\nfirst distance "<<endl;
				c1.display();
				cout<<"\nsecond distance "<<endl;
				c2.display();
				cout<<"\nafter addition"<<endl;
				c3=c1+c2;
				c3.display();
				break;
			case 2:
```

```
                    if(c2<c1){
                        cout<<"\nc2 is lesser than c1";
                }
                    else{
                        cout<<"\nc2 is greater than c1";
                    }
                    break;
                case 3:
                    cout<<"\nbefore operation "<<endl;
                    c1.display();
                    ++c1;
                    cout<<"\nafter operation "<<endl;
                    c1.display();
                    break;
                case 4:
                    cout<<"before operation "<<endl;
                    c2.display();
                    c2++;
                    cout<<"\nafter operation "<<endl;
                    c2.display();
                    break;
                case 5:
                    cout<<"\nthank you";
                    return 0;
                default:
                    cout<<"\nenter valid opt:";


        }       }
}
```

Input and Output





PP Lab Name: Shilpi Biswal
PRN: 22070122198

```
1.Addition
2.Comparison
3.Prefix
4.Postfix
5.Quit
enter your option:
3

before operation

F: 5I: 5
after operation

F: 6I: 6
```

```
1.Addition
2.Comparison
3.Prefix
4.Postfix
5.Quit
enter your option:
4
before operation

F: 6I: 7
after operation
```

```
1.Addition
2.Comparison
3.Prefix
4.Postfix
5.Quit
enter your option:
5

thank you
--------------------------------
Process exited with return value 0
Press any key to continue . . .
```

Conclusion: Thus we have implemented the concept of operator overloading in C++

Practice programs: Write a C++ program for Unary logical NOT (!) operator overloading.

Program code

```cpp
#include<iostream>
using namespace std;
class Distance
{
    int feet;
    int inch;
public:
    Distance(){
        feet=0;
        inch=0;
    };
    Distance(int a,int b){
        feet=a;
        inch=b;
    }
    bool operator !=(Distance m){
        if(feet!=m.feet || inch !=m.inch){
            return true;
        }
    }
};
int main(){
    Distance d1(5,6);
    Distance d2(7,6);
    if(d1!=d2){
        cout<<"\nthe two distances are not equal";
    }
    else{
        cout<<"they are equal";
    }
}
```

PP Lab Name: Shilpi Biswal
PRN: 22070122198

Output