

Experiment No. 11

Operator Overloading

OBJECTIVE:

Things that will be covered in today's lab:

- Operator Overloading

THEORY:

Operator Overloading provides the ability to use the same operator to perform different actions. In C++ the statement `c = a + b` will compile successfully if `a`, `b` and `c` are of "int" and "float" types and if we attempt to compile the statement when `a`, `b` and `c` are the objects of user-defined classes, the compiler will generate error message but with operator overloading, this can happen.

Operator overloading is done with the help of a special function, called operator function, which defines the operation that the overloaded operator will perform relative to the class upon which it will work. An operator function is created using the keyword *operator*.

Operator functions can be either members or nonmembers of a class. Non-member operator functions are almost always friend functions of the class, however. The way operator functions are written differs between member and nonmember functions. The general format of member operator function is:

Function prototype: (with in a class)

```
return_type operator op(arglist)
```

Function definition:

```
return_type class_name :: operator op(arglist)
{
    function body // task defined
}
```

You can also overload an operator for a class by using a non-member function, which is usually a friend of the class. Since a friend function is not a member of the class, it does not have a *this* pointer. Therefore, an overloaded friend operator function is passed the operands explicitly. This means that a friend function that overloads a binary operator has two parameters, and a friend function that overloads a unary operator has one parameter. When overloading a binary operator using a friend function, the left operand is passed in the first parameter and the right operand is passed in the second parameter. Insertion and extraction operators, Operator function must be a nonmember function of the class. Here is the syntax.

Function prototype :(with in a class)

```
Friend ostream& operator<<(ostream&, const class_name&);  
Friend istream& operator>>(istream&, class_name&);
```

Function definition:

```
ostream& operator<<( ostream& out, const class_name& obj )  
{  
    // ...  
    return out;  
}  
istream& operator>>( istream& in, class_name& obj )  
{  
    // ...  
    return in;  
}
```

We can overload the entire C++ operator except following.

1. Class member access operator (.) 5) .*
2. Scope resolution operator (::)
3. Size operator (sizeof)
4. Conditional operator (? :)

Example: What should be the output of this program?

```
class Distance  
{  
private:  
int feet, inches;  
public:  
    Distance(int f, int i) { feet = f; inches = i; }  
void operator=(const Distance &D )  
{  
    feet = D.feet;  
    inches = D.inches;  
}  
void displayDistance() {  
    cout <<"F: "<< feet <<" I:"<< inches << endl;  
}  
};  
void main()  
{  
    Distance D1(11, 10), D2(5, 11);  
    D1 = D2;  
    cout <<"First Distance :";  
    D1.displayDistance();  
}
```

```
}
```

Exercise 1:

Implement class for Complex numbers. Overload the following operators

1. *(Multiplication)
2. +(Addition)
3. =(Assignment)
4. == (Equality Comparator – two equal signs without space)

The Class definition is given below:

```
class comp
{
    double real;
    double imag;
public:
    // default constructor
    // function that set real and imag part of complex no
    // function that print complex number
    // Opertator "+" for addition
    // Opertator "*" for Multiplication

    // Opertator "==" for check both complex are equal or not
    // Opertator "=" for assignment
};
```

Your program should run for the following main().

```
void main()
{
    comp n1,n2,n3;

    n1.setpara(2,3);
    n2.setpara(1,4);

    n3=n1;
    n3.show();

    (n1+n2).show();

    n3=n1*n2;
    n3.show();

    n3.show();

    if(n1==n2)
        cout<<"Both no have same real and imag part "<<endl;
    else
        cout<<"Unequal !!!"<<endl;
}
```

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

class comp {
private:
    double real;
    double imag;

public:
    comp();
    void setpara(double r, double im);
    void print() const;

    comp operator+(const comp&);
    comp operator*(const comp&);
    bool operator==(const comp&);
    comp& operator=(const comp&);
};

comp::comp() : real(0.0), imag(0.0) {}
```

```

void comp::setpara(double r, double im) {
    real = r;
    imag = im;
}

void comp::print() const {
    cout << real << " + " << imag << "i" << endl;
}

// Operator overloading definitions
comp comp::operator+(const comp& a) {
    comp temp;
    temp.real = real + a.real;
    temp.imag = imag + a.imag;
    return temp;
}

comp comp::operator*(const comp& a) {
    comp result;
    result.real = (real * a.real) - (imag * a.imag);
    result.imag = (real * a.imag) + (imag * a.real);
    return result;
}

bool comp::operator==(const comp& ap2) {
    return (real == ap2.real) && (imag == ap2.imag);
}

comp& comp::operator=(const comp& ap2) {
    if (this != &ap2) {
        real = ap2.real;
        imag = ap2.imag;
    }
    return *this;
}

void main() {
    comp n1, n2, n3;

    n1.setpara(2, 3);
    n2.setpara(1, 4);

    n3 = n1;
    n3.print();
}

```

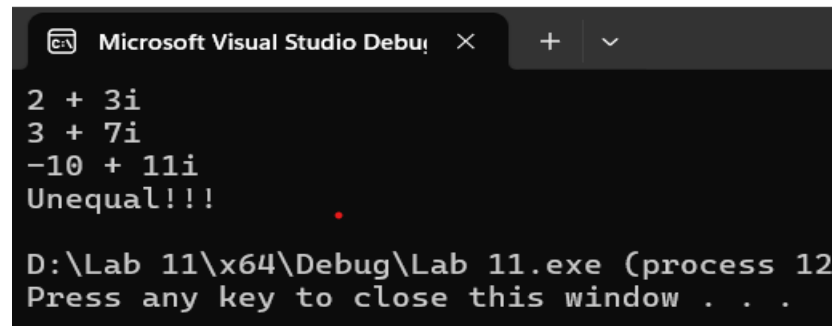
```

    (n1 + n2).print();

    n3 = n1 * n2;
    n3.print();

    if (n1 == n2)
        cout << "Both numbers have the same real and imag part." << endl;
    else
        cout << "Unequal!!!" << endl;
}

```



```

Microsoft Visual Studio Debug Console
+ -
2 + 3i
3 + 7i
-10 + 11i
Unequal!!!

D:\Lab 11\x64\Debug\Lab 11.exe (process 12)
Press any key to close this window . . .

```

Exercise 2:

Write a class implementation for a class named **PhoneNumber** giving the specification below:

Data that is associated with this class are:

- First name of type string.
- Last name of type string.
- Phone number of type string.

Functions:

- Constructor: that initializes any object of type *PhoneNumber*.
- Overloaded function for the insertion operator (<<) to print any object of type *PhoneNumber*.

- Overloaded function for extraction operator (>>) to read in for any object of type *PhoneNumber* all the values of its data members.

Write a driver program that test the class as follow:

- Declare an object of type “*PhoneNumber*”, read in its values, and then print it (using operator<<, operator>>).
- Declare an array of three objects of type “*PhoneNumber*”, read in their values and then print their values (using operator<<, operator>>).

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

class PhoneNumber{
private:
    string firstName;
    string lastName;
    string phoneNum;
public:
    PhoneNumber(){}

    friend ostream& operator<<(ostream&, PhoneNumber&);

    friend istream& operator>>(istream&, PhoneNumber&);
};

ostream& operator<<(ostream& os, PhoneNumber& pn) {
    os << "First Name: " << pn.firstName << endl;
    os << "Last Name: " << pn.lastName << endl;
    os << "Phone Number: " << pn.phoneNum << endl;
    return os;
}

istream& operator>>(istream& in, PhoneNumber& pn) {
    cout << "Enter your First Name: ";
    in >> pn.firstName;
    cout << "Enter Last Name: ";
    in >> pn.lastName;
    cout << "Enter Phone Number: ";
    in >> pn.phoneNum;
    return in;
}
```

```

int main() {
    // single phone number obj
    PhoneNumber phon;
    cout << "Enter single phone Number object" << endl;
    cin >> phon;
    cout << " Single phone Number object" << endl;
    cout << phon;

    // array of phone number obj
    const int size = 3;

    PhoneNumber arr[size];
    cout << "Enter details for " << size << " PhoneNumber objects:" << endl;
    for (int i = 0; i < size; i++) {
        cout << "-> Enter details for PhoneNumber object " << i + 1 << ":" <<
endl;
        cin >> arr[i];
    }
    cout << "Details for " << size << " PhoneNumber objects:" << endl;
    for (int i = 0; i < size; i++) {
        cout << "-> PhoneNumber object " << i + 1 << ":" << endl;
        cout << arr[i] << endl;
    }
}

```

Output :



Microsoft Visual Studio Debug



```
Enter single phone Number object
Enter your First Name: Mudassir
Enter Last Name: Hussain
Enter Phone Number: 03215606827
Single phone Number object
First Name: Mudassir
Last Name: Hussain
Phone Number: 03215606827
Enter details for 3 PhoneNumber objects:
-> Enter details for PhoneNumber object 1:
Enter your First Name: hahdi
Enter Last Name: ch
Enter Phone Number: 0321548622
-> Enter details for PhoneNumber object 2:
Enter your First Name: usman
Enter Last Name: ch
Enter Phone Number: 032451124
-> Enter details for PhoneNumber object 3:
Enter your First Name: usama
Enter Last Name: ch
Enter Phone Number: 0215486212
Details for 3 PhoneNumber objects:
-> PhoneNumber object 1:
First Name: hahdi
Last Name: ch
Phone Number: 0321548622

-> PhoneNumber object 2:
First Name: usman
Last Name: ch
Phone Number: 032451124

-> PhoneNumber object 3:
First Name: usama
Last Name: ch
Phone Number: 0215486212

D:\Lab 11\x64\Debug\Lab 11.exe (process 4268) exited with c
Press any key to close this window . . .|
```