Page No.
Date

Experiment No. 1.

Name - Sumit Gulab Bhamare
SEComp A08
Sub - OOPL

Aim:- Design a class 'complex' with data members for real & imaginary part. Provide default & parameterized constructors. Write a program to perform arithmetic operations of two complex numbers using operator overloading.
Addition & Subtraction using friend function.
Multiplication & division using member functions.

Objectives: To understand concept of operator overloading.
To understand the concept of friend function
To understand the concept of member function.

Software used - Linux Operating System, GCC.

Theory - You can redefine or overload most of the built in operators available in C++. Thus a programmer can use operators with user defined types as well
Overloaded operators are functions with special names the keyword operator followed by the symbol for the operator being defined. Like any other function, an overloaded operator has a return type & a parameter list.
Box operator op(operator object);

Page No.
Date

1) Unary operator : The unary operators operate a single operand & following are the examples of unary operators : The increment (++) & decrement (--) operators . The unary minus(-) operator. The logical not(!) operator.
The unary operators operate on the object for which they were called & normally, this operator appears on the ~~right~~ left side of the object , as in !obj , -obj and ++obj but sometime they can be used as postfix as well like obj++ or obj-- . following example explains how minus (-) operator can be overloaded for prefix as well as postfix usage,

```
# include <iostream>
using namespace std;
class Distance
{
   private:
   int feet ; // 0 to infinite
   int inches ; // 0 to 12
   public:
   // required constructors
   Distance () {
   feet=0;
   inches=0;
   }
   Distance (int f, int i){
   feet = f;
   inches = i;
   }
   // method to display distance.
   Void display Distance()
   {
   cout<< "F:" << feet << "I:" << inches << endl;
   }
```

Page No.
Date

```
// overloaded minus(-) operator
Distance operator-()
{
 feet = -feet;
 inches= -inches;
returnDistance(feet, inches);
}
};
int main()
{
Distance D1(11,10), D2(-5,11);
-D1; // apply negation
D1.displayDistance(); //display D1
-D2; //apply negation
D2.display Distance(); //display D2
return 0;
}
```

2) Binary Operators: Overloading with a single ~~processor~~ parameter is called binary operator overloading. Similar to unary operators, binary operators can also be overloaded. Binary operators require two operands, and they are overloaded by using member functions & friend functions.

Example

```
using namespace std;
class temp
{
complex operator + (complex c2)
    {
        complex c3;
        c3.x = x+c2.x;
        c3.y=y + c2.y;
```

Page No.
Date

```
        return c3;
      }
  };
```

Algorithm :

1) Start
2) Create class complex, with data members x & y & ~~member~~ member functions accept(), display()
3) Initialize 3 objects c1, c2, c3
4) Define default constructor to initialize variables to 0+0i
5) Define operator overloaded functions to add, subtract, multiply & divide two complex numbers
6) Call the operator overloaded functions
7) Use c3 = c1 + c2; to invoke the overloaded +operator.
8) Use c3 = c1 - c2; to invoke the overloaded - operator.
9) Use c3 = c1/c2; to invoke the overloaded / operator
10) Use c3 = c1 * c2; to invoke the overloaded * operator
11) After ~~pre~~ performing the required operations call display()
12) Stop.

Input:
3+2i (ordinary number & imaginary number.
2+3i

Output:
5+5i;

Conclusion: Thus we studied concept of friend function & member function.

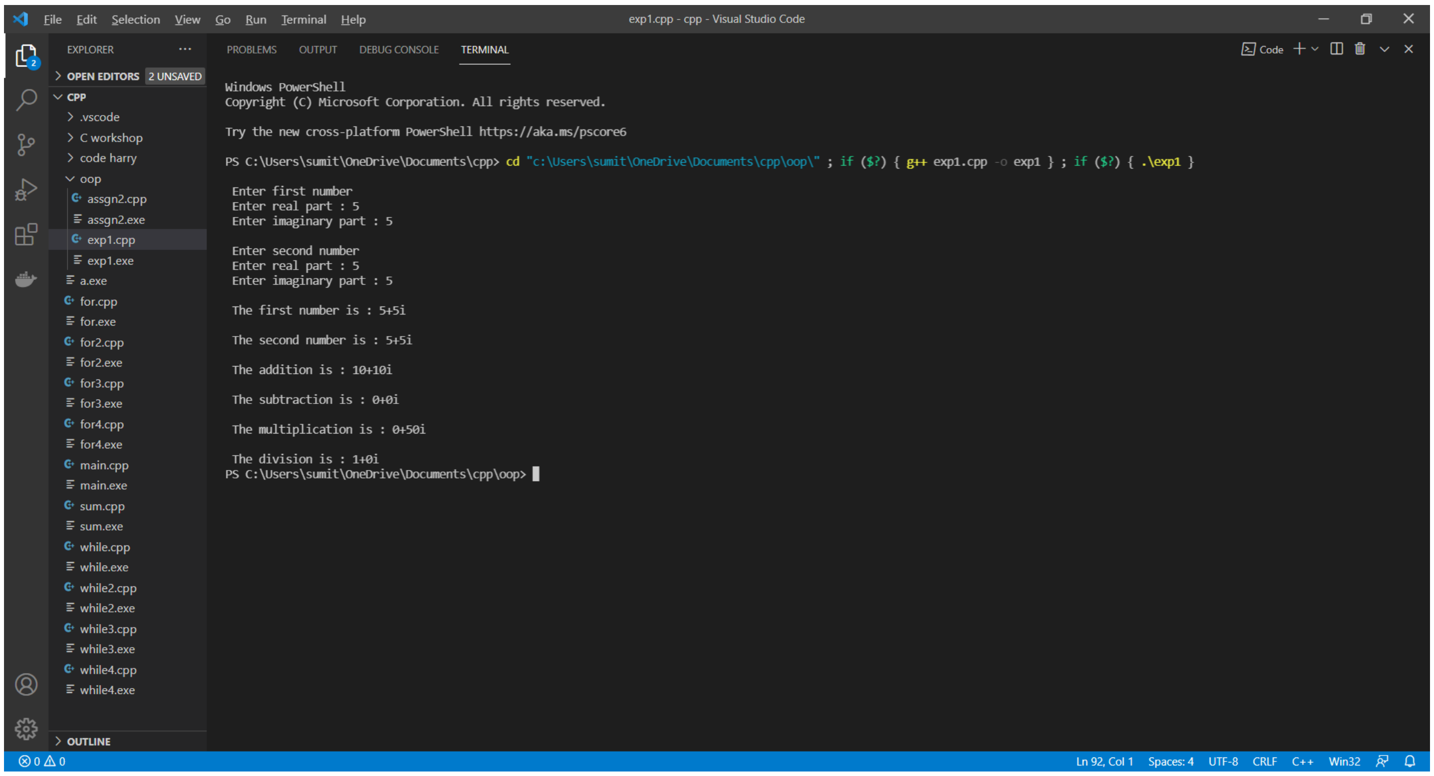
Program :

```
#include<iostream>
#include<stdio.h>
#include<conio.h>
using namespace std;
class complex
{
  float x;
  float y;
  public:
  complex operator+(complex);
  complex operator-(complex);
  complex operator*(complex);
  complex operator/(complex);
  complex();
  complex(float,float);
  void display();
  void getdata();
};
complex::complex()
{
  x=0;
  y=0;
}
complex::complex(float a, float b)
{
  x=a;
  y=b;
}
complex complex::operator+(complex c)
{
  complex temp;
  temp.x=x+c.x;
  temp.y=y+c.y;
  return(temp);
}
complex complex::operator-(complex c)
{
```

```cpp
  complex temp1;
  temp1.x=x-c.x;
  temp1.y=y-c.y;
  return(temp1);
}
complex complex::operator*(complex c)
{
  complex temp2;
  temp2.x=(x*c.x)-(y*c.y);
  temp2.y=(y*c.x)+(x*c.y);
  return(temp2);
}
complex complex::operator/(complex c)
{
  complex temp3;
  temp3.x=((x*c.x)+(y*c.y))/((c.x*c.x)+(c.y*c.y));
  temp3.y=((y*c.x)-(x*c.y))/((c.x*c.x)+(c.y*c.y));
  return(temp3);
}
void complex::getdata()
{
  cout<<" Enter real part : ";
  cin>>x;
  cout<<" Enter imaginary part : ";
  cin>>y;
}
void complex::display()
{
  cout<<x<<"+"<<y<<"i\n";
}
int main()
{
  complex c1, c2, c3, c4, c5, c6;
  cout<<"\n Enter first number"<<endl;
  c1.getdata();
  cout<<"\n Enter second number"<<endl;
  c2.getdata();
  c3=c1+c2;
  c4=c1-c2;
  c5=c1*c2;
```

```cpp
    c6=c1/c2;
    cout<<"\n The first number is : ";
    c1.display();
    cout<<"\n The second number is : ";
    c2.display();
    cout<<"\n The addition is : ";
    c3.display();
    cout<<"\n The subtraction is : ";
    c4.display();
    cout<<"\n The multiplication is : ";
    c5.display();
    cout<<"\n The division is : ";
    c6.display();
}
```

Output :



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\sumit\OneDrive\Documents\cpp> cd "c:\Users\sumit\OneDrive\Documents\cpp\oop\" ; if ($?) { g++ exp1.cpp -o exp1 } ; if ($?) { .\exp1 }

 Enter first number
 Enter real part : 5
 Enter imaginary part : 5

 Enter second number
 Enter real part : 5
 Enter imaginary part : 5

 The first number is : 5+5i

 The second number is : 5+5i

 The addition is : 10+10i

 The subtraction is : 0+0i

 The multiplication is : 0+50i

 The division is : 1+0i
PS C:\Users\sumit\OneDrive\Documents\cpp\oop>
```