Assignment 03

```
// 1. Write a operator overloading code to overload all the
arithmetic operators to add 2 complex no, 1 complex no and int
value and one non member function to add int and complex no.
//Q 3,4 & 5 Also covered in this.

#include <iostream>
#include <string.h>
using namespace std;

struct Complex
{
private:
    int real;
    int imaginary;

public:
    Complex()
    {
        this->real = 0;
        this->imaginary = 0;
    }

    Complex(int real, int imaginary)
    {
        this->real = real;
        this->imaginary = imaginary;
    }

    // Setters
    void setReal(int r)
    {
        this->real = r;
    }

    void setImaginary(int i)
    {
        this->imaginary = i;
    }

    // Getters
    int getReal()
    {
        return this->real;
    }

    int getImaginary()
    {
        return this->imaginary;
    }
```

```
    // Display
    void display()
    {
        cout << this->real << "+" << this->imaginary << "i  ";
    }

    // Addition
    Complex operator+(Complex c)
    {
        Complex temp;
        temp.real = this->real + c.getReal();
        temp.imaginary = this->imaginary + c.getImaginary();
        return temp;
    }

    Complex operator+(int a)
    {
        Complex temp;
        temp.real = this->real + a;
        temp.imaginary = this->imaginary + a;
        return temp;
    }

    // substraction
    Complex operator-(Complex c)
    {
        Complex temp;
        temp.real = this->real - c.getReal();
        temp.imaginary = this->imaginary - c.getImaginary();
        return temp;
    }

    Complex operator-(int a)
    {
        Complex temp;
        temp.real = this->real - a;
        temp.imaginary = this->imaginary - a;
        return temp;
    }

    // operator*tiplication
    Complex operator*(Complex c)
    {
        cout << "\nOperator *\n";
        Complex temp;
        temp.real = this->real * c.getReal();
        temp.imaginary = this->imaginary * c.getImaginary();
        return temp;
    }

    Complex operator*(int a)
    {
        cout << "\nOperator *****\n";
```

```
        Complex temp;
        temp.real = this->real * a;
        temp.imaginary = this->imaginary * a;
        return temp;
    }
    // Division
    Complex operator/(Complex c)
    {
        Complex temp;
        temp.real = this->real / c.getReal();
        temp.imaginary = this->imaginary / c.getImaginary();
        return temp;
    }

    Complex operator/(int a)
    {
        Complex temp;
        temp.real = this->real / a;
        temp.imaginary = this->imaginary / a;
        return temp;
    }
    // Mod
    Complex operator%(Complex c)
    {
        cout << "\nOperator Mod\n";
        Complex temp;
        temp.real = this->real % c.getReal();
        temp.imaginary = this->imaginary % c.getImaginary();
        return temp;
    }

    Complex operator%(int a)
    {
        cout << "\nOperator Mod.......\n";
        Complex temp;
        temp.real = this->real % a;
        temp.imaginary = this->imaginary % a;
        return temp;
    }

    // Relational
    int operator>(Complex c)
    {

        if (this->real > c.getReal())
            return 1;
        else
            return 0;
    }

    int operator<(Complex c)
    {
```

```
        if (this->real < c.getReal())
            return 1;
        else
            return 0;
    }

    // Unary Inc post
    Complex operator++(int a)
    {
        Complex temp;
        int x = this->real++;
        int y = this->imaginary++;
        temp.setReal(x);
        temp.setImaginary(y);
        return temp;
    }

    // Unary Inc pre
    Complex operator++()
    {
        Complex temp;
        int x = ++this->real;
        int y = ++this->imaginary;
        temp.setReal(x);
        temp.setImaginary(y);
        return temp;
    }

    // Unary Dec
    Complex operator--(int a)
    {
        Complex temp;
        int x = this->real--;
        int y = this->imaginary--;
        temp.setReal(x);
        temp.setImaginary(y);
        return temp;
    }

    // Unary Inc pre
    Complex operator--()
    {
        Complex temp;
        int x = --this->real;
        int y = --this->imaginary;
        temp.setReal(x);
        temp.setImaginary(y);
        return temp;
    }
    // Logical
    //  Logical AND (&&)
    int operator&&(Complex c)
    {
```

```
        return (this->real && c.getReal()) && (this->imaginary && c.getImaginary());
    }

    // Logical OR (||)
    int operator||(Complex c)
    {
        return (this->real || c.getReal()) || (this->imaginary || c.getImaginary());
    }

    // Logical NOT (!)
    int operator!()
    {
        return !this->real && !this->imaginary;
    }
};

// Global Add
Complex operator+(int a, Complex c)
{
    printf("\nGlobal Add Fun");
    Complex temp;
    temp.setReal(a + c.getReal());
    temp.setImaginary(a + c.getImaginary());
    return temp;
}
// Global Sub
Complex operator-(int a, Complex c)
{
    printf("\nGlobal Substract Fun");
    Complex temp;
    temp.setReal(a - c.getReal());
    temp.setImaginary(a - c.getImaginary());
    return temp;
}
// Global operator*
Complex operator*(int a, Complex c)
{
    printf("\nGlobal operator* Fun");
    Complex temp;
    temp.setReal(a * c.getReal());
    temp.setImaginary(a * c.getImaginary());
    return temp;
}
// Global Divide
Complex operator/(int a, Complex c)
{
    printf("\nGlobal Div Fun");
    Complex temp;
    temp.setReal(a / c.getReal());
    temp.setImaginary(a / c.getImaginary());
    return temp;
}
```

```
int main()
{
    Complex c1(10, 20), c2(30, 40);

    Complex c3;
    // ADD
    cout << "\n\nAddition of  : ";
    c1.display();
    cout << " + ";
    c2.display();
    cout << " is  = ";
    c3 = c1 + c2;
    c3.display();

    cout << "\nAddition of  : ";
    c1.display();
    cout << " + ";
    cout << "10 is  = ";
    c3 = c1 + 10;
    c3.display();
    // Sub
    cout << "\n\nSubstraction of  : ";
    c1.display();
    cout << " - ";
    c2.display();
    cout << " is  = ";
    c3 = c2 - c1;
    c3.display();

    cout << "\nSubstraction of  : ";
    c3.display();
    cout << " - ";
    cout << "10 is  = ";
    c3 = c3 - 10;
    c3.display();
    // Div
    cout << "\n\nDivision of  : ";
    c1.display();
    cout << " / ";
    c2.display();
    cout << " is  = ";
    c3 = c2 / c1;
    c3.display();

    cout << "\nDivision of  : ";
    c2.display();
    cout << " / ";
    cout << "10 is  = ";
    c3 = c2 / 10;
    c3.display();

    // Mul
    cout << "\n\nMultiplication of  : ";
```

```
    c1.display();
    cout << " * ";
    c2.display();
    cout << " is  = ";
    c3 = c2 * c1;
    c3.display();

    cout << "\nMultiplication of  : ";
    c3.display();
    cout << " * ";
    cout << "10 is  = ";
    c3 = c3 * 10;
    c3.display();
    // Mod
    cout << "\n\nMod of  : ";
    c1.display();
    cout << " % ";
    c2.display();
    cout << " is  = ";
    c3 = c2 % c1;
    c3.display();

    cout << "\nMod of  : ";
    c2.display();
    cout << " % ";
    cout << "7 is  = ";
    c3 = c2 % 10;
    c3.display();

    cout << "\nMod of  : ";
    c2.display();
    cout << " % ";
    cout << "7 is  = ";
    c3 = c2 % 10;
    c3.display();
    // Compare
    cout << "\n\nComparision of: ";
    c1.display();
    cout << " > ";
    c2.display();
    cout << " is  = ";

    if (c2 > c1)
    {
        c2.display();
        cout << "Is greater..\n";
    }
    else
    {
        c1.display();
        cout << "Is greater..\n";
    }
    //
```

```
    cout << "\n\nComparision of: ";
    c1.display();
    cout << " < ";
    c2.display();
    cout << " is  = ";

    if (c2 < c1)
    {
        c2.display();
        cout << "Is Less..\n";
    }
    else
    {
        c1.display();
        cout << "Is Less..\n";
    }

    // Inc
    cout << "\n\nPre Increment of: ";
    c1.display();
    c3 = ++c1; // c1.operator++(int);
    cout << " is  : ";
    c3.display();

    cout << "\nPost Increment of: ";
    c1.display();
    c3 = c1++; // c1.operator++(int);
    cout << " is  : ";
    c3.display();

    // DEC
    cout << "\n\nPre Decrement of: ";
    c1.display();
    c3 = --c1; // c1.operator++(int);
    cout << " is  : ";
    c3.display();

    cout << "\nPost Decrement of: ";
    c1.display();
    c3 = c1--; // c1.operator++(int);
    cout << " is  : ";
    c3.display();

    // Logical AND
    cout << "\n\nLogical AND of: ";
    c1.display();
    cout << " && ";
    c2.display();
    cout << " is  = ";
    if (c1 && c2)
        cout << "True\n";
    else
        cout << "False\n";
```

```
    // Logical OR
    cout << "\nLogical OR of: ";
    c1.display();
    cout << " || ";
    c2.display();
    cout << " is  = ";
    if (c1 || c2)
        cout << "True\n";
    else
        cout << "False\n";

    // Logical NOT
    cout << "\nLogical NOT of: ";
    c1.display();
    cout << " is  = ";
    if (!c1)
        cout << "True\n";
    else
        cout << "False\n";

    return 1;
}
```

Output:

```
PS D:\Fullstack-Java-FirstBit-Solutions\Basic-C-and-CPP\CPP\Assignments\Assignment03\output> &
.\'q1ComplexCalculaor.exe'
Addition of  : 10+20i  + 30+40i  is  = 40+60i
Addition of  : 10+20i  + 10 is  = 20+30i
Substraction of  : 10+20i  - 30+40i  is  = 20+20i
Substraction of  : 20+20i  - 10 is  = 10+10i
Division of  : 10+20i  / 30+40i  is  = 3+2i
Division of  : 30+40i  / 10 is  = 3+4i

Multiplication of  : 10+20i  * 30+40i  is  =
Operator *
300+800i
Multiplication of  : 300+800i  * 10 is  =
Operator *****
3000+8000i

Mod of  : 10+20i  % 30+40i  is  =
Operator Mod
0+0i
Mod of  : 30+40i  % 7 is  =
Operator Mod.....
0+0i
Mod of  : 30+40i  % 7 is  =
Operator Mod.....
0+0i

Comparision of: 10+20i  > 30+40i  is  = 30+40i  Is greater..
```

Comparision of: 10+20i < 30+40i is = 10+20i Is Less..

Pre Increment of: 10+20i is : 11+21i
Post Increment of: 11+21i is : 11+21i

Pre Decrement of: 12+22i is : 11+21i
Post Decrement of: 11+21i is : 11+21i

Logical AND of: 10+20i && 30+40i is = True

Logical OR of: 10+20i || 30+40i is = True

Logical NOT of: 10+20i is = False
PS D:\Fullstack-Java-FirstBit-Solutions\Basic-C-and-CPP\CPP\Assignments\Assignment03\output>

```cpp
// 2. Write a operator overloading code to overload all the arithmetic operators to add 2 distances, 1 distance and int value and one non member function to add int and distance.
//Q 3,4 & 5 Also covered in this.

#include <iostream>
using namespace std;

typedef struct Distance
{
    int feet;
    int inch;
    // Constructor
    Distance()
    {
        this->feet = 0;
        this->inch = 0;
    }

    Distance(int feet, int inch)
    {
        this->feet = feet;
        this->inch = inch;
    }

public:
    // Setters
    void setFeet(int feet) { this->feet = feet; }
    void setInch(int inch) { this->inch = inch; }

    // Getters
    int getFeet() { return this->feet; }
    int getInch() { return this->inch; }
```

```
// Display
void display()
{
    cout << "\nDistance: " << this->feet << " feet " << this->inch << " inches";
}
// Arithmatic
Distance operator+(Distance distance)
{
    Distance temp;
    temp.feet = this->feet + distance.getFeet();
    temp.inch = this->inch + distance.getInch();
    return temp;
}
Distance operator-(Distance distance)
{
    Distance temp;
    temp.feet = this->feet - distance.getFeet();
    temp.inch = this->inch - distance.getInch();
    return temp;
}
Distance operator/(Distance distance)
{
    Distance temp;
    temp.feet = this->feet / distance.getFeet();
    temp.inch = this->inch / distance.getInch();
    return temp;
}
Distance operator*(Distance distance)
{
    Distance temp;
    temp.feet = this->feet * distance.getFeet();
    temp.inch = this->inch * distance.getInch();
    return temp;
}

Distance operator+(int distance)
{
    Distance temp;
    temp.feet = this->feet + distance;
    temp.inch = this->inch + distance;
    return temp;
}
Distance operator-(int distance)
{
    Distance temp;
    temp.feet = this->feet - distance;
    temp.inch = this->inch - distance;
    return temp;
}
Distance operator/(int distance)
{
    Distance temp;
```

```
        temp.feet = this->feet / distance;
        temp.inch = this->inch / distance;
        return temp;
    }
    Distance operator*(int distance)
    {
        Distance temp;
        temp.feet = this->feet * distance;
        temp.inch = this->inch * distance;
        return temp;
    }

    // logical operator
    int operator&&(Distance distance)
    {
        return (this->feet && distance.feet) && (this->inch && distance.inch);
    }

    int operator||(Distance distance)
    {
        return (this->feet || distance.feet) || (this->inch || distance.inch);
    }

    int operator!()
    {
        return !(this->feet || this->inch);
    }

    // relational operator
    int operator==(Distance distance)
    {
        return (this->feet == distance.feet) && (this->inch == distance.inch);
    }

    int operator!=(Distance distance)
    {
        return (this->feet != distance.feet) || (this->inch != distance.inch);
    }

    int operator>(Distance distance)
    {
        if (this->feet > distance.feet)
            return true;
        else if (this->feet == distance.feet)
            return this->inch > distance.inch;
        return false;
    }

    int operator<(Distance distance)
    {
        if (this->feet < distance.feet)
            return true;
        else if (this->feet == distance.feet)
```

```
            return this->inch < distance.inch;
        return false;
    }

    int operator>=(Distance distance)
    {
        return !(*this < distance);
    }

    int operator<=(Distance distance)
    {
        return !(*this > distance);
    }

    // unary operator

    // Pre-increment
    Distance operator++()
    {
        ++this->feet;
        ++this->inch;
        return *this;
    }

    // Post-increment
    Distance operator++(int)
    {
        Distance temp = *this;
        this->feet++;
        this->inch++;
        return temp;
    }

    // Pre-decrement
    Distance operator--()
    {
        --this->feet;
        --this->inch;
        return *this;
    }

    // Post-decrement
    Distance operator--(int)
    {
        Distance temp = *this;
        this->feet--;
        this->inch--;
        return temp;
    }

} Distance;

Distance operator+(int distance, Distance distance1)
```

```
{
    Distance temp;
    temp.setFeet(distance + distance1.getFeet());
    temp.setInch(distance + distance1.getInch());
    return temp;
}

Distance operator-(int distance, Distance distance1)
{
    Distance temp;
    temp.setInch(distance - distance1.getInch());
    temp.setFeet(distance - distance1.getFeet());
    return temp;
}

Distance operator*(int distance, Distance distance1)
{
    Distance temp;
    temp.setFeet(distance * distance1.getFeet());
    temp.setInch(distance * distance1.getInch());
    return temp;
}

Distance operator/(int distance, Distance distance1)
{
    Distance temp;
    temp.setFeet(distance / distance1.getFeet());
    temp.setInch(distance / distance1.getInch());
    return temp;
}

int main()
{

    Distance distance1(10, 5);
    Distance distance2(5, 5);
    Distance distance3;

    cout << "\nDistance 1";
    distance1.display();
    cout << "\nDistance 2";
    distance2.display();

    // Arithmetic Operators
    cout << "\n\nAddition of both Distances:";
    distance3 = distance1 + distance2;
    distance3.display();

    cout << "\n\nSubtraction of both Distances:";
    distance3 = distance1 - distance2;
    distance3.display();

    cout << "\n\nMultiplication of both Distances:";
```

```
    distance3 = distance1 * distance2;
    distance3.display();

    cout << "\n\nDivision of both Distances:";
    distance3 = distance1 / distance2;
    distance3.display();

    cout << "\n\nAddition of 10 and Distance:";
    distance2.display();
    cout << " = ";
    distance3 = 10 + distance2;
    distance3.display();

    cout << "\n\nSubstraction of 10 and Distance:";
    distance2.display();
    cout << " = ";
    distance3 = 10 - distance2;
    distance3.display();

    cout << "\n\nMultiplication of 10 and Distance:";
    distance2.display();
    cout << " = ";
    distance3 = 10 * distance2;
    distance3.display();

    cout << "\n\nDivision of 10 and Distance:";
    distance2.display();
    cout << " = ";
    distance3 = 10 / distance2;
    distance3.display();

    // Logical Operators
    cout << "\n\nLogical AND (&&) of both Distances: " << (distance1 && distance2);
    cout << "\nLogical OR (||) of both Distances: " << (distance1 || distance2);
    cout << "\nLogical NOT (!) of Distance 1: " << (!distance1);

    // Relational Operators
    cout << "\n\nDistance 1 == Distance 2: " << (distance1 == distance2);
    cout << "\nDistance 1 != Distance 2: " << (distance1 != distance2);
    cout << "\nDistance 1 > Distance 2: " << (distance1 > distance2);
    cout << "\nDistance 1 < Distance 2: " << (distance1 < distance2);
    cout << "\nDistance 1 >= Distance 2: " << (distance1 >= distance2);
    cout << "\nDistance 1 <= Distance 2: " << (distance1 <= distance2);

    // Unary Operators
    cout << "\nPre-Increment (++Distance 1):";
    distance3 = ++distance1;
    distance3.display();

    cout << "\nPost-Increment (Distance 1++):";
    distance3 = distance1++;
    distance3.display();
```

```
    cout << "\nPre-Decrement (--Distance 1):";
    distance3 = --distance1;
    distance3.display();

    cout << "\nPost-Decrement (Distance 1--):";
    distance3 = distance1--;
    distance3.display();

    return 1;
}
```

Output: PS D:\Fullstack-Java-FirstBit-Solutions\Basic-C-and-CPP\CPP\Assignments\Assignment03\output> & .\'q2DistanceCalculator.exe'

Distance 1 Distance: 10 feet 5 inches
Distance 2 Distance: 5 feet 5 inches

Addition of both Distances: Distance: 15 feet 10 inches

Subtraction of both Distances: Distance: 5 feet 0 inches

Multiplication of both Distances: Distance: 50 feet 25 inches

Division of both Distances: Distance: 2 feet 1 inches

Addition of 10 and Distance: Distance: 5 feet 5 inches =
Distance: 15 feet 15 inches

Substraction of 10 and Distance: Distance: 5 feet 5 inches =
Distance: 5 feet 5 inches

Multiplication of 10 and Distance: Distance: 5 feet 5 inches =
Distance: 50 feet 50 inches

Division of 10 and Distance: Distance: 5 feet 5 inches =
Distance: 2 feet 2 inches

Logical AND (&&) of both Distances: 1
Logical OR (||) of both Distances: 1
Logical NOT (!) of Distance 1: 0

Distance 1 == Distance 2: 0
Distance 1 != Distance 2: 1
Distance 1 > Distance 2: 1
Distance 1 < Distance 2: 0
Distance 1 >= Distance 2: 1
Distance 1 <= Distance 2: 0
Pre-Increment (++Distance 1): Distance: 11 feet 6 inches
Post-Increment (Distance 1++): Distance: 11 feet 6 inches
Pre-Decrement (--Distance 1): Distance: 11 feet 6 inches
Post-Decrement (Distance 1--): Distance: 11 feet 6 inches
PS D:\Fullstack-Java-FirstBit-Solutions\Basic-C-and-CPP\CPP\Assignments\Assignment03\output>