**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.

#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;

    }

};

// 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();

    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

PS D:\Fullstack-Java-FirstBit-Solutions\Basic-C-and-CPP\CPP\Assignments\Assignment03\output>