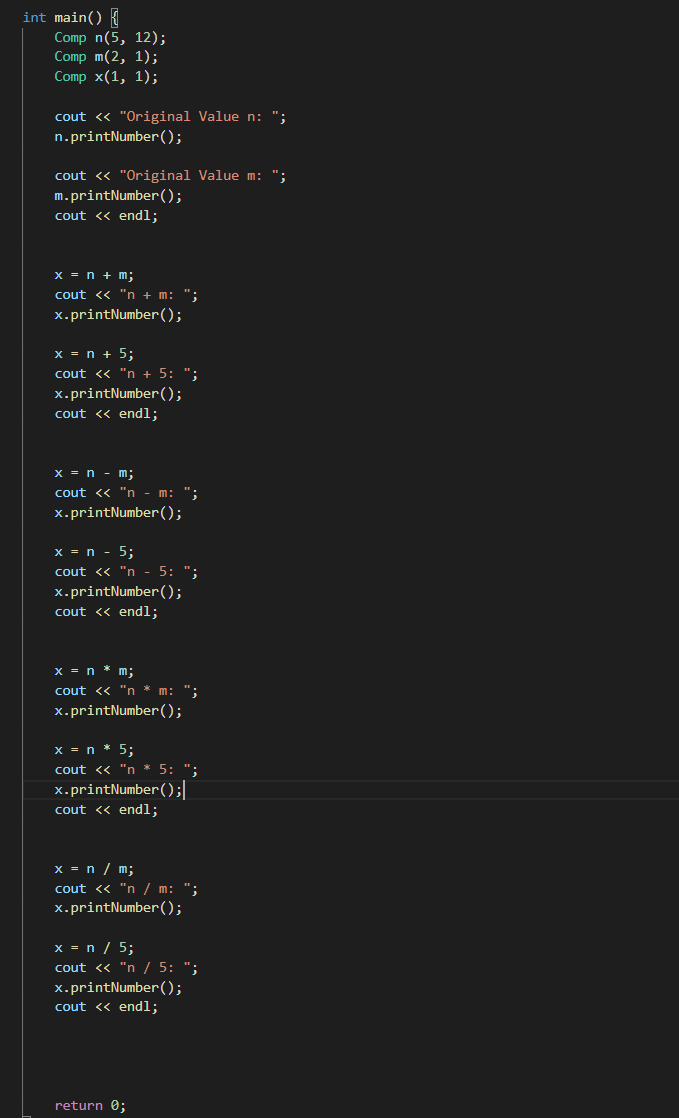
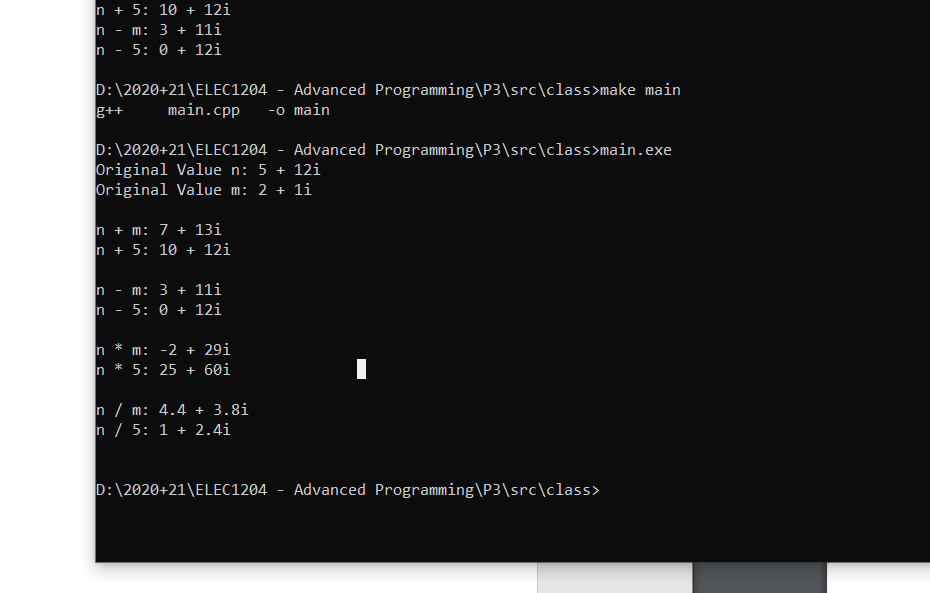
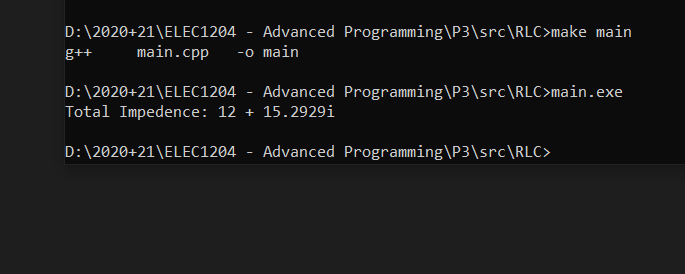
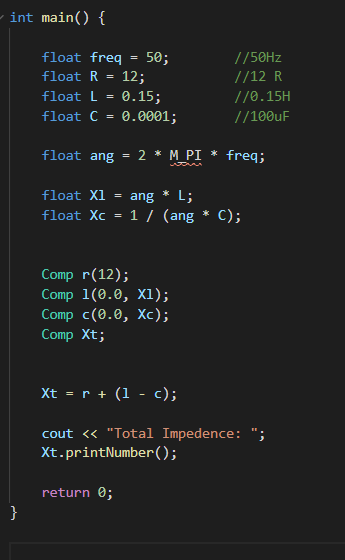
**P3- Imaginary Numbers**

Final

3.1 Implementing Complex Class



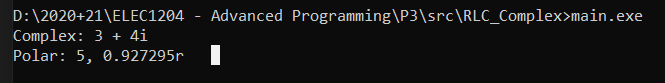
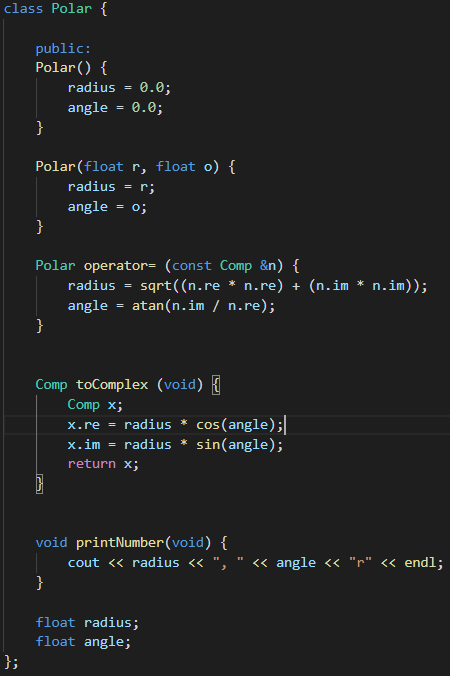
3.2 Converting RLC to impedance



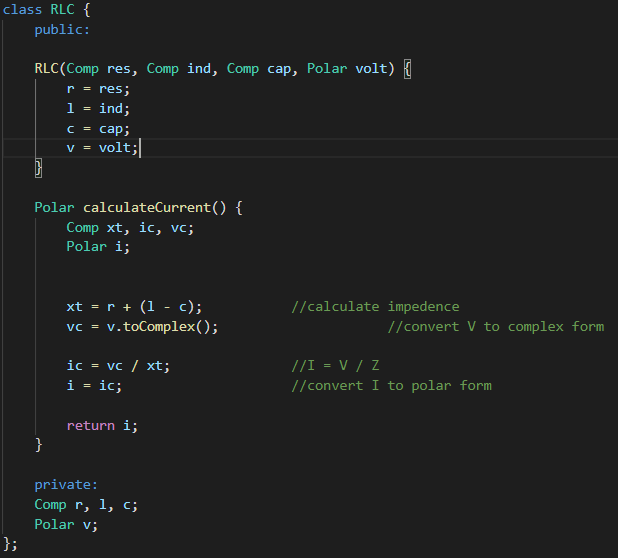
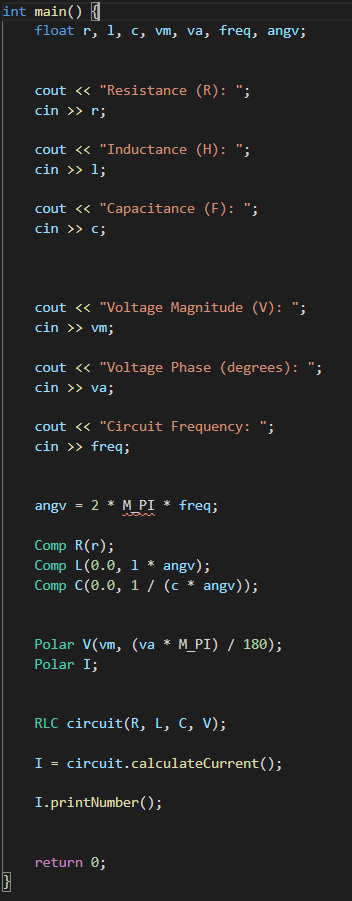
The answers match so I can confirm the code works.

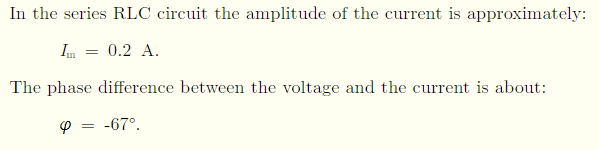
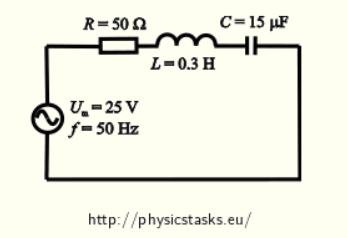
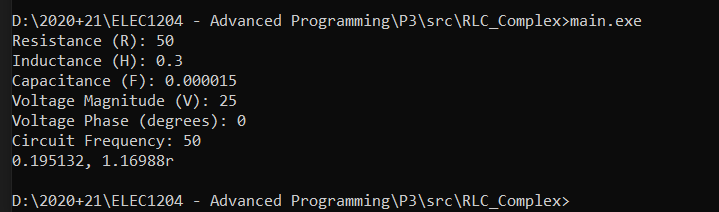
3.3 RLC Circuit Simulation

To better implement this I created a new class besides Comp which is Polar. I then allowed seamless conversion between Comp and Polar using overloaded operators.



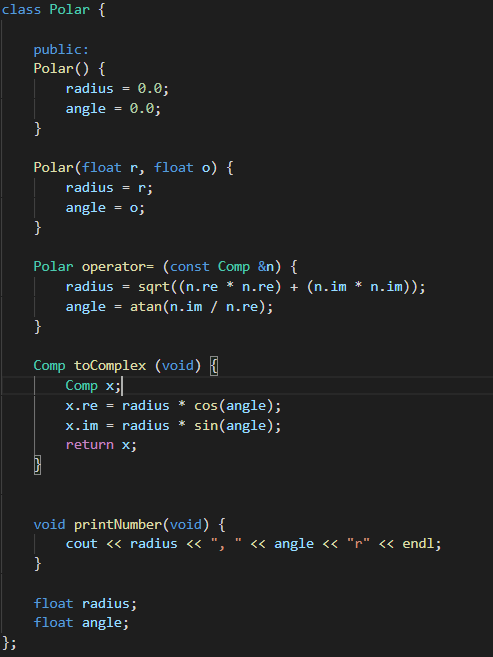
This makes it easier to deal with voltage and current which is in polar form and impedance which is in imaginary number form.

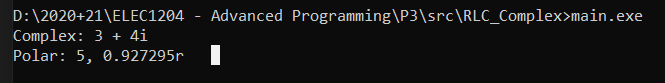




4 Optional Additional Work

I had already included the ability to do this in my Polar class. This class can take in either and angle and a radius or can be set equal to a complex number (Comp class). This is displayed working below.





Complete Class Code

#include <iostream>

//#include "PolarNumbers.cpp"

#include <cmath>

using namespace std;

class Comp;

class Polar;

class Comp {

    public:

    Comp() {

        re = 0.0;

        im = 0.0;

    }

    Comp(float real) {

        re = real;

        im = 0.0;

    }

    Comp(float real, float imaginary) {

        re = real;

        im = imaginary;

    }

    Comp operator= (const Comp &n) {

        //a + bi = c + di

        re = n.re;

        im = n.im;

    }

    Comp operator= (const float n) {

        //a + bi = n + 0i

        re = n;

        im = 0.0;

    }

    Comp operator+ (const Comp &n) {

        //(a + bi) + (c + di) = (a + c) + (b + d)i

        Comp x;

        x.re = this->re + n.re;

        x.im = this->im + n.im;

        return x;

    }

    Comp operator+ (const float n) {

        //(a + bi) + n = (a + n) + (b)i

        Comp x;

        x.re = this->re + n;

        x.im = this->im;

        return x;

    }

    Comp operator- (const Comp &n) {

        //(a + bi) - (c + di) = (a - c) + (b - d)i

        Comp x;

        x.re = this->re - n.re;

        x.im = this->im - n.im;

        return x;

    }

    Comp operator- (const float n) {

        //(a + bi) - n = (a - n) + (b - n)i

        Comp x;

        x.re = this->re - n;

        x.im = this->im;

        return x;

    }

    Comp operator\* (const Comp &n) {

        //(a + bi) \* (c + di) = (ac - bd) + (ad + bc)i

        Comp x;

        x.re = (this->re \* n.re) - (this->im \* n.im);

        x.im = (this->re \* n.im) + (this->im \* n.re);

        return x;

    }

    Comp operator\* (const float n) {

        //(a + bi) \* n = (an) + (bn)i

        Comp x;

        x.re = this->re \* n;

        x.im = this->im \* n;

        return x;

    }

    Comp operator/ (const Comp &n) {

        //(a + bi) / (c + di) = (ac + bd) / (c^2 + d^2) + (bc - ad)i / (c^2 + d^2)

        float denom = (n.re \* n.re) + (n.im \* n.im);        //common denominator makes reading easier

        Comp x;

        x.re = ((this->re \* n.re) + (this->im \* n.im)) / denom;

        x.im = ((this->im \* n.re) - (this->re \* n.im)) / denom;

        return x;

    }

    Comp operator/ (const float n) {

        //(a + bi) / n = (a / n) + (b / n)i

        Comp x;

        x.re = this->re / n;

        x.im = this->im / n;

        return x;

    }

    void printNumber(void) {

        cout << re;

        cout << ((im < 0) ? "- " : " + ");

        cout << im << "i" << endl;

    }

    float re;

    float im;

};

class Polar {

    public:

    Polar() {

        radius = 0.0;

        angle = 0.0;

    }

    Polar(float r, float o) {

        radius = r;

        angle = o;

    }

    Polar operator= (const Comp &n) {

        radius = sqrt((n.re \* n.re) + (n.im \* n.im));

        angle = atan(n.im / n.re);

    }

    Comp toComplex (void) {

        Comp x;

        x.re = radius \* cos(angle);

        x.im = radius \* sin(angle);

        return x;

    }

    void printNumber(void) {

        cout << radius << ", " << angle << "r" << endl;

    }

    float radius;

    float angle;

};

class RLC {

    public:

    RLC(Comp res, Comp ind, Comp cap, Polar volt) {

        r = res;

        l = ind;

        c = cap;

        v = volt;

    }

    Polar calculateCurrent() {

        Comp xt, ic, vc;

        Polar i;

        xt = r + (l - c);           //calculate impedence

        vc = v.toComplex();                     //convert V to complex form

        ic = vc / xt;               //I = V / Z

        i = ic;                     //convert I to polar form

        return i;

    }

    private:

    Comp r, l, c;

    Polar v;

};