03 - Operators and Streams Here's the things we covered:

- Operator overloading
- Aggregate initialization
- Streams

Homework. Consider the quadratic equation

$$0 = ax^2 + bx + c$$

and its solution

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}. \tag{1}$$

We will only consider this solution for now. Write a function that computes (1) for real values of a, b, c. Test the function on the polynomials

$$0 = x^2 + 2x - 23 \tag{2}$$

and

$$0 = x^2 - 2x + 23. \tag{3}$$

You will notice that the roots of (2) are real and that the roots of (3) are imaginary, which will be a problem for a real-valued implementation of (1). Consider the real-valued implementation of 1 below:

```
#include <iostream>
#include <cmath>

using value_type = double;

value_type quadratic_formula(const value_type& a, const value_type& b, const value_type& c)
{
        return (-1.0*b + sqrt(b*b - 4.0*a*c)) / (2.0*a);
}

int main(int argc, char** argv)
{

        value_type a0 =  1.0;
        value_type b0 =  2.0;
        value_type c0 = -23.0;

        value_type a1 =  1.0;
        value_type b1 = -2.0;
        value_type c1 =  23.0;

        std::cout << quadratic_formula(a0, b0, c0) << std::endl;
        std::cout << quadratic_formula(a1, b1, c1) << std::endl;
        return 0;
}
```

This implementation will output garbage when we use `double` as our underlying type. Therefore, the task is to write a complex number type `my_complex_t` that will produce the expected behavior when we switch the `value_type` type alias.

You can start with the following layout:

```
struct my_complex_t
{
        double x, y;

        my_complex_t(){}

        // Note that this will allow the syntax:
        // my_complex_t x = 1.0;
        my_complex_t (const double& val) : x{val}, y{0.0} {}

        my_complex_t (const double& xx, const double& yy) : x{xx}, y{yy} {}
};

my_complex_t sqrt(const my_complex_t& a)
{
        const auto theta  = atan2(a.y, a.x);
        const auto r_sqrt = sqrt(sqrt(a.x*a.x + a.y*a.y));
        my_complex_t i(0.0, 1.0);
        return r_sqrt*cos(theta/2.0) + i*r_sqrt*sin(theta/2.0);
}
```

Note that you will need to define 9 operators:

1. `+`, `-`, `*`, and `/` for complex-complex binary operations,
2. `+`, `-`, `*`, and `/` for double-complex binary operations, and
3. `<<` for printing to the terminal.

Test your implementation **ONLY** by switching

```
using value_type = double;
```

to

```
using value_type = my_complex_t;
```