МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСОЙ ФЕДЕРАЦИИ МОСКОВСКИЙ АВЦИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЬЕЛЬСКИЙ УНИВЕРСТИТЕТ)

ЛАБОРАТОРНАЯ РАБОТА №1 по курсу

объектно-ориентированное программирование I семестр, 2019/20 уч. год

Студент *Попов Данила Андреевич, группа 08-208Б-18*

Преподаватель *Журавлёв Андрей Андреевич*

Условие

Задание №1: написать класс, который реализует комплексное число в алгебраической форме со следующими функциями:

1. Сложение
2. Вычитание
3. Умножение
4. Деление
5. Сравнение
6. Сопряжённое число

Описание программы

Код программы состоит из 3-х файлов:

1. app/main.cpp: файл, содержащий точку входа приложения
2. include/complex.hpp: файл, содержащий объявление и реализацию inline-функций
3. src/lib/complex.cpp: реализация не-inline методов класса Complex.h

Дневник отладки

Была одна очень забавная проблема в test.py. После считывания тестового запроса и отправки его в программу, весь тест зависал. Как оказалось, проблема была в отсутствующем '\n' в конце запроса, который породил немало головной боли :)

Недочёты

Метод Str() работает не с внешним буффером, а создаёт каждый раз минимум два:

1. Для std::stringstream объекта, который предоставляет удобный интерфейс приведения стандартных типов к строке.
2. Для std::string объекта, который является возвращаемым значением.

Данный метод может использовать достаточно много процессорного времени при приведении объектов типа Complex к строке.

Выводы

В целом, повторил синтаксис написания классов в C++, изучил базовое межпроцессорное взаимодействие через пайпы и отловил один коварный баг, который не так просто обнаружить. Так же вспомнил, что именно делает модификатор inline, который используется по умолчанию при объявлении с реализацией методов в классе.

Исходный код

Complex.hpp

```
#pragma once

#include <cmath>
#include <string>
#include <sstream>
#include <ostream>

class Complex;

class Complex {
public:
    inline Complex() noexcept
        : re{}
        , im{}
    {}

    inline explicit Complex(double real, double imaginary) noexcept
        : re{ real }
        , im{ imaginary }
    {}

    inline Complex(const Complex& other) noexcept
        : re(other.re)
        , im(other.im)
    {}


    inline double& Real() noexcept { return re; }
    inline double& Imag() noexcept { return im; }
    inline const double& Real() const noexcept { return re; }
    inline const double& Imag() const noexcept { return im; }

    inline Complex Add(const Complex& other) const noexcept {
        return Complex{ this->re + other.re, this->im + other.im };
    }

    inline Complex Sub(const Complex& other) const noexcept {
        return this->Add(Complex{ -other.re, -other.im });
    }
```

```
    inline Complex Mul(const Complex& other) const noexcept {
        return Complex{
            this->re * other.re - this->im * other.im,
            this->re * other.im + this->im * other.re
        };
    }

    inline Complex Div(const Complex& other) const noexcept {
        double denominator = other.Mul(other.Conj()).re;
        Complex numerator = this->Mul(other.Conj());
        return Complex{ numerator.re / denominator, numerator.im / denominator };
    }

    inline bool Equ(const Complex& other) const noexcept {
        return (this->re == other.re) && (this->im == other.im);
    }

    inline Complex Conj() const noexcept {
        return Complex{ this->re, -this->im };
    }

    inline double Mod() const noexcept {
        return std::sqrt(this->Mul(this->Conj()).re);
    }

    inline std::string Str() const {
        std::stringstream string;
        Write(string);
        return string.str();
    }

    // IO methods
    void Read(std::istream& stream);
    void Write(std::ostream& stream) const;

private:
    double re, im;
};

int Compare(const Complex& left, const Complex& right);
```

Complex.cpp

```cpp
#include <complex.hpp>

void Complex::Read(std::istream& stream) {
    stream >> re >> im;
}

void Complex::Write(std::ostream& stream) const {
    stream << re << " " << im;
}

int Compare(const Complex& left, const Complex& right) {
    double leftMod = left.Mod();
    double rightMod = right.Mod();

    return (leftMod < rightMod)
        ? -1
        : (leftMod == rightMod
            ? 0
            : 1);
}
```

main.cpp

```cpp
// stdlib headers:
#include <cstdio>
#include <iostream>
#include <string>

// LabLib headers:
#include <complex.hpp>

using namespace std;

void ToUpper(string& str);

int main() {
    string input;
    while (cin) {
        cin >> input;
        ToUpper(input);
        Complex left, right;
        left.Read(cin);
        right.Read(cin);

        if (cin.fail()) {
            break;
        }

        if (input == "ADD") {
            left.Add(right).Write(cout);
        }
        else if (input == "SUB") {
            left.Sub(right).Write(cout);
        }
        else if (input == "MUL") {
            left.Mul(right).Write(cout);
        }
        else if (input == "DIV") {
            left.Div(right).Write(cout);
        }
        else if (input == "EQU") {
            cout << (left.Equ(right) ? "True" : "False");
        }
        else if (input == "CMP") {
```

```
            cout << Compare(left, right);
        }
        cout << endl;
        cout.flush();
    };
}

void ToUpper(string& str) {
    for (auto& c : str) {
        c = static_cast<remove_reference_t<decltype(c)>>(toupper(c));
    }
}
```