МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСТИТЕТ)

ЛАБОРАТОРНАЯ РАБОТА №2 по курсу

объектно-ориентированное программирование I семестр, 2021/22 уч. год

Студент *Борисов Ян Артурович, группа М80-208Б-20*
Преподаватель *Дорохов Евгений Павлович*

Цель:

- Изучение основ работы с классами в C++;
- Перегрузка операций и создание литералов

Требования к программе

Вариант задания: 2, Комплексное число в тригонометрической форме

Разработать программу на языке C++ согласно варианту задания. Программа на C++ должна собираться с помощью системы сборки CMake. Программа должна получать данные из стандартного ввода и выводить данные в стандартный вывод.

Реализовать над объектами реализовать в виде перегрузки операторов.

Реализовать пользовательский литерал для работы с константами объектов созданного класса.

Описание программы

Исходный код лежит в main.cpp файле.

Дневник отладки

Во время выполнения лабораторной работы программа не нуждалась в отладке, все ошибки компиляции были исправлены с первой попытки. После их исправления программа работала так, как было задумано изначально.

Недочёты

Недочётов не было обнаружено.

Выводы

В данной лабораторной познакомились с пользовательскими литералами в C++. Самостоятельно реализовали пользовательский литерал для комплексного числа в тригонометрической форме и перегрузили операцию сравнения для сравнения чисел по действительной части.

Исходный код

main.cpp

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

class Complex{
public:
    Complex();
    Complex(double r, double j);
    Complex add(Complex num2);
    Complex sub(Complex num2);
    Complex mul(Complex num2);
    Complex div(Complex num2);
    bool equ(Complex num2);
    bool equ_by_real(Complex num2);
    Complex conj();
    Complex alg_form(Complex num);
    Complex alg_form_to_geom(Complex num);
    friend std::istream& operator>> (std::istream &in, Complex &num);
    friend std::ostream& operator<< (std::ostream &out, const Complex
&num);
    bool operator==(const Complex& rh);

private:
    double r,j;
};

Complex::Complex(double r, double j) {
    this->r = r;
    this->j = j;
}

Complex Complex::add(Complex num2) {
    Complex alg_form1 = alg_form(*this);
    Complex alg_form2 = alg_form(num2);
```

```
    Complex result(alg_form1.r + alg_form2.r, alg_form1.j +
alg_form2.j);
    return alg_form_to_geom(result);
}

Complex Complex::sub(Complex num2) {
    if(this->r == num2.r && this->j == num2.j) return Complex(0, 0);
    Complex alg_form1 = alg_form(*this);
    Complex alg_form2 = alg_form(num2);
    Complex result(alg_form1.r - alg_form2.r, alg_form1.j -
alg_form2.j);
    return alg_form_to_geom(result);
}

Complex Complex::mul(Complex num2) {
    return Complex(this->r * num2.r, this->j + num2.j);
}

Complex Complex::div(Complex num2) {
    if(num2.r == 0 && num2.j == 0){
        std::cout << "На 0 делить нельзя!" << std::endl;
        return *this;
    }
    return Complex(this->r / num2.r, this->j - num2.j);
}

bool Complex::equ(Complex num2) {
    return this->r == num2.r && this->j == num2.j;
}

Complex Complex::conj() {
    return Complex(this->r, -this->j);
}

bool Complex::equ_by_real(Complex num2) {
    const double e = 1e-5;
    Complex alg_form1 = alg_form(*this);
    Complex alg_form2 = alg_form(num2);
    return (std::abs(alg_form1.r - alg_form2.r) < e);
}

Complex Complex::alg_form(Complex num) {
    return Complex(num.r * cos(num.j), num.r * sin(num.j));
}

Complex Complex::alg_form_to_geom(Complex num) {
    double a = num.r, b = num.j;
    double z = sqrt(a * a + b * b);
    double argZ = (-3.14 + atan(b / a)) * (180 / 3.14);
    return Complex(z, argZ);
}
```

```
Complex::Complex() {
    this->r = 0;
    this->j = 0;
}

std::istream &operator>>(std::istream &in, Complex &num) {
    in >> num.r >> num.j;
    return in;
}

std::ostream &operator<<(std::ostream &out, const Complex &num) {
    out << "Complex num in trigonometric form: " << num.r << "*(cos" <<
num.j << " + i * sin" << num.j << ")" << std::endl;
    return out;
}

bool Complex::operator==(const Complex &rh) {
    return this->equ_by_real(rh);
}

Complex operator "" _complex(const char* str, size_t size) {
    int cnt = 0;
    std::string s = "";
    while (str[cnt] != ' '){
        s += str[cnt++];
    }
    double r = 0, j = 0;
    for (int i = 0; i < s.size(); ++i) {
        r *= 10;
        r += s[i] - '0';
    }
    s = "";
    while (str[cnt++] != '\0') {
        s += str[cnt];
    }
    for (int i = 0; i < s.size() - 1; ++i) {
        j *= 10;
        j += s[i] - '0';
    }
    Complex res(r, j);
    return res;
}


int main() {
    Complex test_num1, test_num2;
    std::cout << "2 25"_complex << std::endl;
    std::cin >> test_num1;
    std::cin >> test_num2;
    std::cout << test_num1 << test_num2 << std::endl;
    std::cout << "Сложение\n" << test_num1.add(test_num2) << std::endl;
    std::cout << "Вычитание\n" << test_num1.sub(test_num2) << std::endl;
```

```
    std::cout << "Умножение\n" << test_num1.mul(test_num2) << std::endl;
    std::cout << "Деление\n" << test_num1.div(test_num2) << std::endl;
    std::cout << "Проверка на равенство\n" << test_num1.equ(test_num2)
<< std::endl;
    if(test_num1 == test_num2){
        std::cout << "Числа равны по действительной части" << std::endl;
    }
    else std::cout << "Числа не равны по действительной части";
    std::cout << "Сопряженное число\n" << test_num1.conj() << std::endl;
    std::cout << "Проверка на равенство по действительной части\n" <<
test_num1.equ_by_real(test_num2) << std::endl;
    return 0;
}
```