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

ЛАБОРАТОРНАЯ РАБОТА №01

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

Студент Абдуль-Хади Филипп, группа М8О-207Б-21
Преподаватель Дорохов Евгений Павлович

Условие

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.

Необходимо настроить сборку лабораторной работы с помощью CMake. Собранная программа должна называться **oop_exercise_01** (в случае использования Windows **oop_exercise_01.exe)**

Вариант №2

Комплексное число в тригонометрической форме представляются парой действительных чисел (r, j), где r – радиус (модуль), j – угол. Реализовать класс Complex для работы с комплексными числами. Обязательно должны быть присутствовать операции

- сложения add, $(r_1, j_1) + (r_2, j_2)$;
- вычитания sub, $(r_1, j_1) – (r_2, j_2)$;
- умножения mul, $(r_1, j_1) \cdot (r_2, j_2)$;
- деления div, $(r_1, j_1) / (r_2, j_2)$;
- сравнение equ, $(r_1, j_1) = (r_2, j_2)$, если $(r_1 = r_2)$ и $(j_1 = j_2)$;
- сопряженное число conj, conj(r, j) = (r, –j).

Реализовать операции сравнения по действительной части.

Описание программы

Исходный код лежит в 11 файлах:

1. main.cpp: основная программа
2. Complex.h: описание класса комплексного числа
3. Complex.cpp: реализация класса комплексного числа
4. CMakeLists.txt: файл сборки cmake

Гитхаб:
https://github.com/64bitwormNR/maiOOP

Дневник отладки

Проблема	**Исправление**
Неудобно собирать без cmake	Создал cmake файл

Недочёты

Отсутствуют

Выводы

Стоит отметить удобство системы сборки cmake, которая автоматически генерирует makefile для linux. Также могу добавить что классы являются очень полезной возможностью языка т.к. теперь всю реализацию касаемо определенного типа объекта можно «засунуть» в классы, а не тащить за собой в виде кучи функций с непонятными аргументами.

Тесты:

Набор 1:
Ввод: a 0.5 1 0.25 0.5
Вывод: (0.729312,0.834909)
Пояснение: Сложение комплексных чисел (0.5,1) и (0.25,0.5)

Набор 2:
Ввод: m 0.3 0.8 0.7 0.5
Вывод: (0.21,1.3)
Пояснение: Умножение комплексных чисел (0.3,0.8) и (0.7,0.5)

Результаты тестов были проверены вручную, ответы верные

Исходный код ниже:

// main.cpp

```
#include <iostream>
#include "Complex.h"


using namespace std;


Complex readFromConsole()
{
        double r;
        double j;
        cin>>r;
        cin>>j;
        return Complex(r,j);
}


int main(int argc,char *argv[])
{
        cout<<"[Variant 2]"<<endl;
        cout<<
        "Supported commands:\n"
        "a r1 j1 r2 j2 (add two complex numbers)\n"
        "s r1 j1 r2 j2 (sub two complex numbers)\n"
        "m r1 j1 r2 j2 (mul two complex numbers)\n"
        "d r1 j1 r2 j2 (div two complex numbers)\n"
        "e r1 j1 r2 j2 (check equality of two complex numbers)\n"
        "c r1 j1 (get conjugacy complex number)\n"
        "r r1 j1 r2 j2 (comparison by real part)\n"
        "q (quit program)\n";
        char run=1;
        while(run)
        {
                cout<<"> ";
                char cmd;
                cin>>cmd;
                switch(cmd)
                {
                        case 'a':
                        {
                                Complex res =
readFromConsole().Add(readFromConsole());
                                cout<<'('<<res.r<<','<<res.j<<')'<<endl;
                                break;
                        }
                        case 's':
                        {
                                Complex res =
readFromConsole().Sub(readFromConsole());
                                cout<<'('<<res.r<<','<<res.j<<')'<<endl;
                                break;
                        }
                        case 'm':
                        {
```

```
                                Complex res =
readFromConsole().Mul(readFromConsole());
                                cout<<'('<<res.r<<','<<res.j<<')'<<endl;
                                break;
                        }
                        case 'd':
                        {
                                Complex res =
readFromConsole().Div(readFromConsole());
                                cout<<'('<<res.r<<','<<res.j<<')'<<endl;
                                break;
                        }
                        case 'e':
                        {
                                char res =
readFromConsole().Equ(readFromConsole());
                                cout<<(res ? "equals":"not equals")<<endl;
                                break;
                        }
                        case 'c':
                        {
                                Complex res = readFromConsole().Conj();
                                cout<<'('<<res.r<<','<<res.j<<')'<<endl;
                                break;
                        }
                        case 'r':
                        {
                                char res =
readFromConsole().EquR(readFromConsole());
                                cout<<(res ? "equals":"not equals")<<endl;
                                break;
                        }
                        case 'q':
                        {
                                run=0;
                                break;
                        }
                        default:
                        {
                                cout<<"wrong command"<<endl;
                        }
                }
        }
        return 0;
}
```

```
// Complex.h

#ifndef COMPLEX_H
#define COMPLEX_H

#include <iostream>

class Complex
{
        public:
                double r;
                double j;
                Complex(double _r, double _j);
                Complex Add(Complex arg);
                Complex Sub(Complex arg);
                Complex Mul(Complex arg);
                Complex Div(Complex arg);
                char Equ(Complex arg);
                Complex Conj();
                char EquR(Complex arg);
        private:
                const double PI = 3.14159265358979323846;
                Complex a2t(Complex arg);
                Complex t2a(Complex arg);
                double sign(double arg);
};

#endif
```

```cpp
// Complex.cpp

#include "Complex.h"
#include <cmath>

Complex::Complex(double _r, double _j)
{
	r = _r;
	j = _j;
}

Complex Complex::Add(Complex arg)
{
	Complex a1 = t2a(arg);
	Complex a2 = t2a(*this);
	return a2t(Complex(a1.r+a2.r,a1.j+a2.j));
}

Complex Complex::Sub(Complex arg)
{
	Complex a1 = t2a(arg);
	Complex a2 = t2a(*this);
	return a2t(Complex(a1.r-a2.r,a1.j-a2.j));
}

Complex Complex::Mul(Complex arg)
{
	Complex a1 = t2a(arg);
	Complex a2 = t2a(*this);
	return a2t(Complex(a1.r*a2.r-a1.j*a2.j,a1.r*a2.j+a1.j*a2.r));
}

Complex Complex::Div(Complex arg)
{
	Complex a1 = t2a(arg);
	Complex a2 = t2a(*this);
	double sqr = a2.r*a2.r+a2.j*a2.j;
	return
a2t(Complex((a1.r*a2.r+a1.j*a2.j)/sqr,(a1.j*a2.r-a1.r*a2.j)/sqr));
}

char Complex::Equ(Complex arg)
{
	Complex a1 = t2a(arg);
	Complex a2 = t2a(*this);
	return a1.r==a2.r && a1.j==a2.j;
}

Complex Complex::Conj()
{
	return Complex(r,-j);
}

char Complex::EquR(Complex arg)
{
	Complex a1 = t2a(arg);
	Complex a2 = t2a(*this);
	return a1.r == a2.r;
}
```

```
Complex Complex::t2a(Complex arg)
{
        return Complex(arg.r*cos(arg.j),arg.r*sin(arg.j));
}

Complex Complex::a2t(Complex arg)
{
        double z = sqrt(arg.r*arg.r+arg.j*arg.j);
        double za = 0;
        double a = arg.r;
        double b = arg.j;
        if(a>0)
        {
                za = atan(b/a)*sign(b);
        }
        else if(a<0)
        {
                za = (PI-atan(b/a))*sign(b);
        }
        else
        {
                za = (PI/2)*sign(b);
        }
        return Complex(z,za);
}

double Complex::sign(double arg)
{
        if(arg>0)
                return 1;
        if(arg<0)
                return -1;
        return 0;
}
```

```
# CmakeLists.txt

cmake_minimum_required(VERSION 3.5)

project (oop_exercise_01)

set(SOURCES
   Complex.cpp
   main.cpp
)

add_executable(oop_exercise_01 ${SOURCES})

target_include_directories(oop_exercise_01
   PRIVATE
      ${PROJECT_SOURCE_DIR}
)
```