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 *Фаттяхетдинов Сильвестр Динарович, группа М80-208Б-20*

Преподаватель *Дорохов Евгений Павлович*

**Цель:**

● Изучение основ работы с классами в С++;

● Перегрузка операций и создание литералов

## Требования к программе

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 3 файлах:

1. main.cpp - исполняемый код.

2. complex.h - специальный файл .h, содержащий прототипы используемых мною функций.

3. complex.cpp - реализация функций для моего задания.

4. CMakeLists.txt - специальный дополнительный файл типа CMakeLists.

**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**

Лабораторная работа №2 - это, по сути, та же самая лабораторная №1, только предусматривающая возможность перегрузки операторов. Лабораторная была выполнена успешно, в ее процессе были еще раз осознаны основные принципы ООП и перегрузки операторов.

**Исходный код**

complex.h

#include <cmath>

#include <string>

#include <iostream>

class complex

{

public:

    complex(double a1, double b1);

    complex(const complex &c);

    complex();

    complex operator+(const complex &c1);

    complex operator-(const complex &c1);

    complex operator\*(complex cc);

    complex operator/(complex cc);

    bool operator==(complex c);

    bool equm(complex c); //сравнение по модулю

    complex conj();       //сопряжённое

    complex &operator=(const complex &c);

    friend std::ostream &operator<<(std::ostream &out, const complex &c);

    friend std::istream &operator>>(std::istream &in, complex &c);

    void print()

    {

        std::cout << "(" << a << ", " << b << "i)";

    };

private:

    double a;

    double b;

};

std::string operator"" \_getimpart(const char \*str, size\_t size);

std::string operator"" \_getrealpart(const char \*str, size\_t size);

complex.cpp   
  
#include "complex.h"

complex::complex(double a1, double b1)

{

    a = a1;

    b = b1;

}

complex::complex(const complex &c)

{

    a = c.a;

    b = c.b;

}

complex::complex()

{

    a = 0;

    b = 0;

};

complex complex::operator+(const complex &c1)

{

    complex ans(a + c1.a, b + c1.b);

    return ans;

};

complex complex::operator-(const complex &c1)

{

    complex ans(a - c1.a, b - c1.b);

    return ans;

};

complex complex::operator\*(complex cc)

{

    double c = cc.a;

    double d = cc.b;

    double a1 = a \* c - b \* d;

    double b1 = a \* d + b \* c;

    complex ans(a1, b1);

    return ans;

}

complex complex::operator/(complex cc)

{

    double c = cc.a;

    double d = cc.b;

    double a1 = (a \* c + b \* d) / (c \* c + d \* d);

    double b1 = (b \* c - a \* d) / (c \* c + d \* d);

    complex ans(a1, b1);

    return ans;

}

bool complex::operator==(complex c)

{

    if (a == c.a && b == c.b)

        return true;

    return false;

}

bool complex::equm(complex c)

{

    if (abs(a) == abs(c.a) && abs(b) == abs(c.b))

        return true;

    return false;

}

complex complex::conj()

{

    complex ans(a, -b);

    return ans;

}

complex &complex::operator=(const complex &c)

{

    a = c.a;

    b = c.b;

    return \*this;

}

std::ostream &operator<<(std::ostream &out, const complex &c)

{

    out << "(" << c.a << ", " << c.b << ") ";

    return out;

};

std::istream &operator>>(std::istream &in, complex &c)

{

    in >> c.a;

    in >> c.b;

    return in;

}

std::string operator"" \_getimpart(const char \*str, size\_t size)

{

    int k = 0;

    std::string im\_part;

    while (str[k] != 'i')

    {

        im\_part.push\_back(str[k]);

        ++k;

    }

    return im\_part;

}

std::string operator"" \_getrealpart(const char \*str, size\_t size)

{

    int k = 0;

    std::string real\_part;

    while (str[k] != 'i')

    {

        ++k;

    }

    ++k;

    while (str[k] != '\0')

    {

        real\_part.push\_back(str[k]);

        ++k;

    }

    return real\_part;

}

main.cpp  
  
#include <iostream>

#include "complex.h"

int main()

{

     complex a, b;

     char option = 'y';

     while (option == 'y')

     {

          std::cout << "Enter 2 complex numbers: \n";

          std::cin >> a >> b;

          std::cout << a << " + " << b << " = " << a + b << "\n";

          std::cout << a << " - " << b << " = " << a - b << "\n";

          std::cout << a << " \* " << b << " = " << a \* b << "\n";

          std::cout << a << " / " << b << " = " << a / b << "\n";

          if (a == b)

               std::cout << a << " = " << b << "\n";

          else

               std::cout << a << " != " << b << "\n";

          std::cout << "Continue?\ny/n: \n";

          std::cin >> option;

          while (option != 'y' && option != 'n')

          {

               std::cout << "Continue?\ny/n: \n";

               std::cin >> option;

          }

     }

     std::cout << "Literals test: \n";

     std::cout << "Your complex number is: "

               << "2i+3"\_getimpart

               << "\*i"

               << "2i+3"\_getrealpart << std::endl;

     std::cout << "Your complex number is: "

               << "100i+10"\_getimpart

               << "\*i"

               << "100i+10"\_getrealpart << std::endl;

     std::cout << "Your complex number is: "

               << "50i+13"\_getimpart

               << "\*i"

               << "50i+13"\_getrealpart << std::endl;

     std::cout << "Shutting down...\n";

     return 0;

}