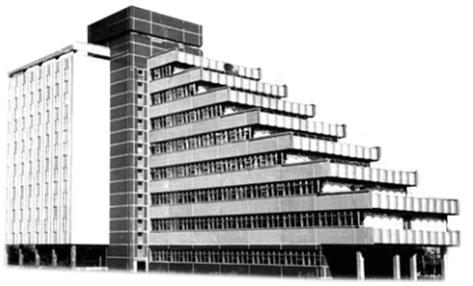
МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННО БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ
ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Учебный Центр Информационных Технологий «Информатика»



Лабораторная работа № 2
по дисциплине «Объектно-ориентированное программирование»

Выполнил слушатель: Пешков Е.В.
Вариант: 9
Дата сдачи:
Преподаватель: Васюткина И.А.

Новосибирск, 2019г.

Задание: Для класса из лаб. работы 1 реализовать набор операций для работы с объектами класса по заданию. Изменить демонстрационную программу, продемострировав все перегруженные операции. Операции перегрузить методами класса и дружественными функциями. Обязательно перегрузить операции ввода/вывода со стандартными устройствами (клавиатура, монитор).

Для класса работы с комплексными числами перегрузить следующие операции:

- умножение/деление объектов;
- операции сравнения комплексных чисел;
- операцию вызова функции – преобразование строки (например, “ 5+i*8”) в комплексное число;
- операцию вызова функции – как получение тригонометрической формы;
- операцию присваивания.

1. Постановка задачи.

В соответствии с заданием необходимо построить класс для работы с комплексными числами. Класс должен включать следующее:

Поля класса:

- double re – действительная часть числа,
- double im – мнимая часть числа,
- char* number – строковое представление числа,
- static int count – статическое поле класса для хранения количества созданных объектов класса

Конструкторы:

- Complex(double _re, double _im) – конструктор с параметрами
- Complex() – конструктор без параметров
- Complex(Complex& object) – конструктор копирования

Деструктор:

- ~ Complex () – деструктор объектов класса без параметров

Методы класса:

- void addition(const Complex&, const Complex&) – сложение комплексных чисел
- void subtraction(const Complex&, const Complex&) – вычитание комплексных чисел
- void multiplication(const Complex&, const Complex&) – умножение комплексных чисел
- void division(const Complex&, const Complex&) – деление комплексных чисел
- void print() - вывод числа в консоль в алгебраической форме записи
- char* toSring(double, double) – преобразование комплексного числа в строку для записи в поле number
- static void Numobject() – вывод количества созданных объектов в программе

Перегрузка операторов:

- Complex& operator= (const Complex&) – оператор присваивания
- Complex operator+ (const Complex&) – оператор сложения
- Complex operator- (const Complex&) - оператор вычитания
- Complex operator* (const Complex&) – оператор умножения
- Complex operator/ (const Complex&) – оператор деления
- void operator() (const char*) – оператор вызова функции с параметрами
- char* operator() () – оператор вызова функции без параметров

- friend bool operator== (const Complex&, const Complex&) – оператор проверки на равенство
- friend ostream& operator<<(ostream&, const Complex&) – оператор вывода в поток
- friend istream& operator>>(istream&, Complex&) – оператор ввода в поток

2. Определение пользовательского класса.

Шаблон класса

```
#pragma warning(disable : 4996)
#include <iostream>
#include <string>
#include <locale>
#include <cstdlib>
#include <cmath>

using namespace std;

class Complex
{
	double re;
	double im;
	char* number;
	static int count;

public:
	// Конструкторы
	Complex(double _re, double _im);
	Complex();
	Complex(const Complex& object);

	// Деструктор
	~Complex();

	// Методы доступа к полям
	double getRe();
	double getIm();

	// Методы
	void addition(const Complex&);
	void subtraction(const Complex&);
	void multiplication(const Complex&);
	void division(const Complex&);
	void print();
	char* toSring(double, double);
	static void Numobject();

	// Переопределение операторов методы класса
	Complex& operator= (const Complex&);
	Complex operator+ (const Complex&);
	Complex operator- (const Complex&);
	Complex operator* (const Complex&);
	Complex operator/ (const Complex&);
	void operator() (const char*);
	char* operator() ();
```

```
        // Переопределение операторов дружеств. ф-ми.
        friend bool operator== (const Complex&, const Complex&);
        friend ostream& operator<<(ostream&, const Complex&);
        friend istream& operator>>(istream&, Complex&);
};
```

// Конструктор с параметрами
Complex::Complex(double _re = 0, double _im = 0)
Параметры:
double _re – действительная часть числа.
double _im – мнимая часть числа.
Возвращаемый результат: ничего не возвращает.
Принцип работы: создает объект класса Complex.

// Конструктор без параметров
Complex::Complex()
Параметры:
Возвращаемый результат: ничего не возвращает.
Принцип работы: создает объект класса Complex.

// Конструктор копирования
Complex::Complex(const Complex& object)
Параметры:
object – объект класса Complex
Возвращаемый результат: ничего не возвращает.
Принцип работы: выполняет копирование одного объекта класса Complex в другой.

// Деструктор
Complex:: ~Complex( )
Параметры:
Возвращаемый результат: ничего не возвращает.
Принцип работы: удаляет объект класса Complex.

// Метод сложения комплексных чисел
void Complex::addition (const Complex& a2)
Параметры:

const Complex& a2– ссылка на второе число.

Возвращаемый результат: ничего не возвращает.
Принцип работы: выполняет сложение комплексных чисел.

// Метод вычитания комплексных чисел
void Complex:: subtraction (const Complex& a2)
Параметры:

const Complex& a2– ссылка на второе число.

Возвращаемый результат: ничего не возвращает.
Принцип работы: выполняет вычитание второго числа из первого комплексного числа.

// **Метод перемножения комплексных чисел**
void Complex:: multiplication (const Complex& a2)
Параметры:

const Complex& a2– ссылка на второе число.

Возвращаемый результат: ничего не возвращает.
Принцип работы: выполняет перемножение первого числа на второе число.

// **Метод деления комплексных чисел**
void Complex:: division (const Complex& a2)
Параметры:

const Complex& a2– ссылка на второе число.

Возвращаемый результат: ничего не возвращает.
Принцип работы: выполняет деление первого числа на второе число.

// **Метод вывода числа в консоль в алгебраической форме**
void Complex:: print ( )
Параметры:

Возвращаемый результат: ничего не возвращает.
Принцип работы: выводит в консоль комплексное число в алгебраической форме записи.

// **Метод преобразования комплексного числа в строку**
char* Complex::toSring(double _re, double _im)
Параметры:
double _re – действительная часть числа.
double _im – мнимая часть числа.

Возвращаемый результат: возвращает указатель на строку.
Принцип работы: преобразование комплексного числа в строку для записи в поле number.

// **Метод подсчёта количества созданных объектов**
void Complex::Numobject()
Параметры:

Возвращаемый результат: ничего не возвращает.
Принцип работы: выводит в консоль количество созданных объектов класса Complex.

// Оператор присваивания
Complex& operator= (const Complex& a2);
Параметры:
const Complex& a2 – ссылка на комплексное число.

Возвращаемый результат: комплексное число.
Принцип работы: выполняет присваивание комплекных чисел.

// Оператор сложения
Complex& operator + (const Complex& a2);
Параметры:
const Complex& a2 – ссылка на комплексное число.

Возвращаемый результат: комплексное число.
Принцип работы: выполняет сложение комплекных чисел.

// Оператор вычитания
Complex& operator - (const Complex& a2);
Параметры:
const Complex& a2 – ссылка на комплексное число.

Возвращаемый результат: комплексное число.
Принцип работы: выполняет вычитание комплекных чисел.

// Оператор умножения
Complex& operator * (const Complex& a2);
Параметры:
const Complex& a2 – ссылка на комплексное число.

Возвращаемый результат: комплексное число.
Принцип работы: выполняет перемножение комплекных чисел.

// Оператор деления
Complex& operator / (const Complex& a2);
Параметры:
const Complex& a2 – ссылка на комплексное число.

Возвращаемый результат: комплексное число.
Принцип работы: выполняет деление комплекных чисел.

// Оператор вызова функции
void operator() (const char*);
Параметры:
const char* - входная строка
Возвращаемый результат: ничего не возвращает
Принцип работы: преобразование строки (например, “ 5+i*8”) в комплексное число;

// Оператор вызова функции
char* operator() ();
Параметры:

Возвращаемый результат: строку содержащую тригенометрическую форму представления числа
Принцип работы: преобразование комплексного числа в строку содержащую тригенометрическую форму

// **Оператор проверки на равенство**
friend bool operator== (const Complex& a1, const Complex& a2);
Параметры:
const Complex& a1 – комплексное число
const Complex& a2 – комплексное число

Возвращаемый результат: логическое значение
Принцип работы: выполняет проверку на равенство 2-х чисел

// **Оператор вывода в поток**
friend ostream& operator<<(ostream& os, const Complex& a1);
Параметры:
const Complex& a1 – комплексное число
ostream& os – поток вывода

Возвращаемый результат: поток вывода
Принцип работы: выводит в поток комплексное число

// **Оператор ввода в поток**
friend istream& operator>>(istream& is, Complex&);
Параметры:
const Complex& a1 – комплексное число
ostream& is – поток bdjlf

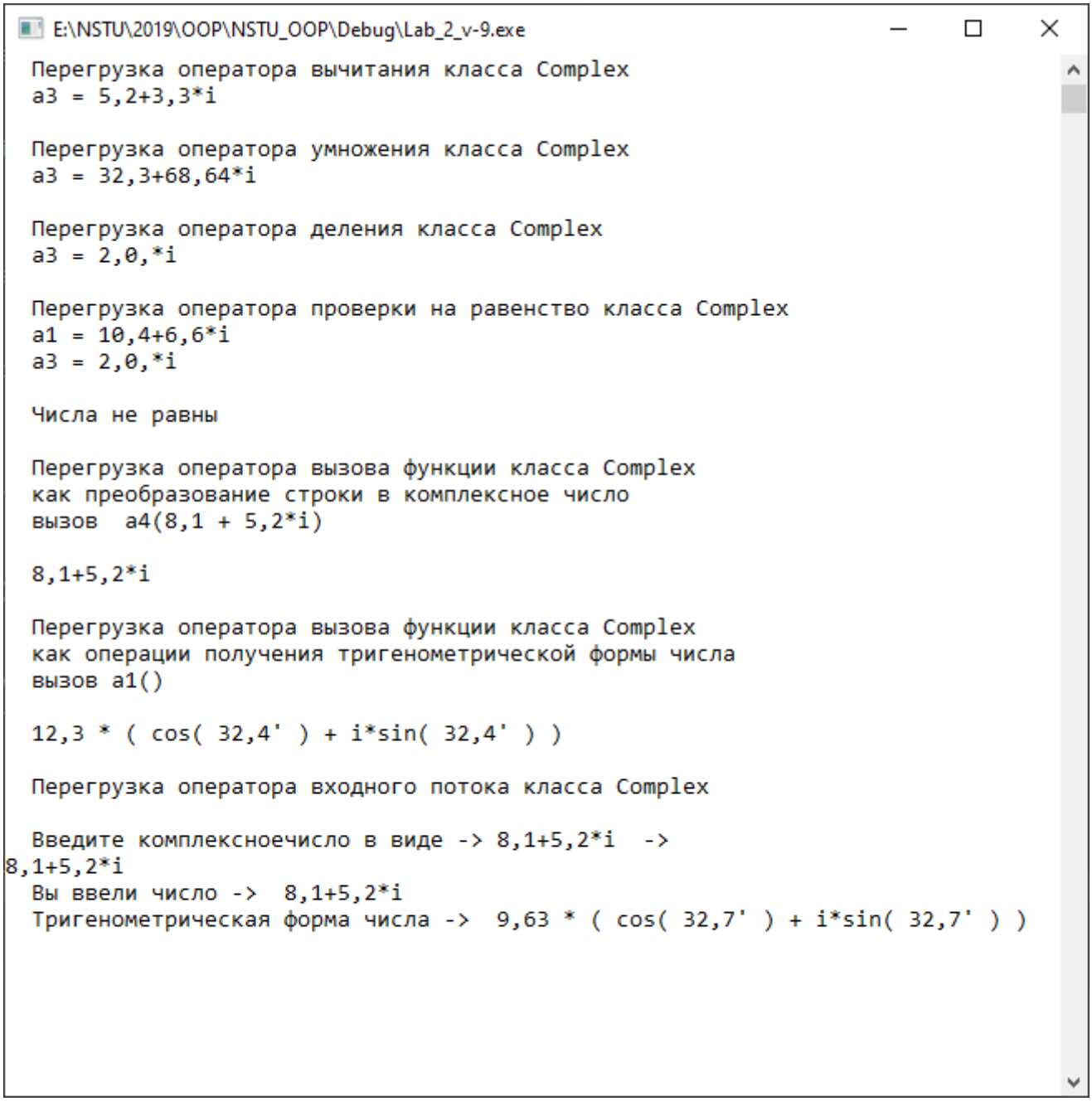
Возвращаемый результат: поток ввода
Принцип работы: считывает из потока комплексное число.

3. Листинг тестовой программы.

```
#include "Complex.h"

int main()
{
   setlocale(LC_ALL, "Russian");

   cout << "  Демонстрация перегрузок операторов для класса Complex  "<< endl;
   cout << endl;
   Complex a1(10.4, 6.6);
   Complex a2(5.2, 3.3);
   Complex a3, a4;
   cout <<"   a1 = "<< a1 << "\n   a2 = "<< a2 << "\n   a3 = " << a3 << "\n   a4 = " << a4 <<
"\n"<<endl;

   cout << "  Перегрузка оператора присваивания класса Complex  " << endl;
   a3 = a1;
   cout <<"  a3 = "<< a3  << endl;
   cout << endl;
   cout << "  Перегрузка оператора сложения класса Complex  " << endl;
   a3 = a1 + a2;
   cout << "  a3 = " << a3 << endl;
   cout << endl;
   cout << "  Перегрузка оператора вычитания класса Complex  " << endl;
   a3 = a1 - a2;
   cout << "  a3 = " << a3 << endl;
   cout << endl;
   cout << "  Перегрузка оператора умножения класса Complex  " << endl;
   a3 = a1 * a2;
   cout << "  a3 = " << a3 << endl;
   cout << endl;
   cout << "  Перегрузка оператора деления класса Complex  " << endl;
   a3 = a1 / a2;
   cout << "  a3 = " << a3 << endl;
   cout << endl;
   cout << "  Перегрузка оператора проверки на равенство класса Complex  " << endl;
   cout << "  a1 = " << a1 << "\n  a3 = "<< a3 << "\n" << endl;
   if (a1 == a3) {
      std::cout << "  Числа равны \n";
   }
   else {
      std::cout << "  Числа не равны \n";
   }
   cout << endl;
   cout << "  Перегрузка оператора вызова функции класса Complex  " << endl;
   cout << "  как преобразование строки в комплексное число  " << endl;
   cout << "  вызов  a4(""8,1 + 5,2*i"") \n" << endl;
   a4("8,1+5,2*i");
   cout <<"  "<< a4 <<"\n"<< endl;
   cout << "  Перегрузка оператора вызова функции класса Complex  " << endl;
   cout << "   как операции получения тригенометрической формы числа \n  вызов a1()\n" <<
endl;

   cout <<"  "<< a1()<< endl;
   cout << "\n  Перегрузка оператора входного потока класса Complex  \n" << endl;
   Complex a5;
```

```
    cout << "  Введите комплексноечисло в виде -> ""8,1+5,2*i""  ->  " << endl;
    cin >> a5;
    cout <<"  Вы ввели число ->  "<< a5 << endl;
    cout << "  Тригенометрическая форма числа ->  " << a5() << endl;
    _getwch();
    return 0;
}
```

4. Пример работы программы.



Приложение. Текст программы

Файл main.c

```
#include "Complex.h"

int main()
{
    setlocale(LC_ALL, "Russian");

    cout << "  Демонстрация перегрузок операторов для класса Complex  "<< endl;
    cout << endl;
    Complex a1(10.4, 6.6);
    Complex a2(5.2, 3.3);
    Complex a3, a4;
    cout <<"  a1 = "<< a1 << "\n  a2 = "<< a2 << "\n  a3 = " << a3 << "\n  a4 = " << a4 <<
"\n"<<endl;

    cout << "  Перегрузка оператора присваивания класса Complex  " << endl;
    a3 = a1;
    cout <<"  a3 = "<< a3  << endl;
    cout << endl;
    cout << "  Перегрузка оператора сложения класса Complex  " << endl;
    a3 = a1 + a2;
    cout << "  a3 = " << a3 << endl;
    cout << endl;
    cout << "  Перегрузка оператора вычитания класса Complex  " << endl;
    a3 = a1 - a2;
    cout << "  a3 = " << a3 << endl;
    cout << endl;
    cout << "  Перегрузка оператора умножения класса Complex  " << endl;
    a3 = a1 * a2;
    cout << "  a3 = " << a3 << endl;
    cout << endl;
    cout << "  Перегрузка оператора деления класса Complex  " << endl;
    a3 = a1 / a2;
    cout << "  a3 = " << a3 << endl;
    cout << endl;
    cout << "  Перегрузка оператора проверки на равенство класса Complex  " << endl;
    cout << "  a1 = " << a1 << "\n  a3 = "<< a3 << "\n" << endl;
    if (a1 == a3) {
        std::cout << "  Числа равны \n";
    }
    else {
        std::cout << "  Числа не равны \n";
    }
    cout << endl;
    cout << "  Перегрузка оператора вызова функции класса Complex  " << endl;
    cout << "  как преобразование строки в комплексное число  " << endl;
    cout << "  вызов  a4(""8,1 + 5,2*i"") \n" << endl;
    a4("8,1+5,2*i");
    cout <<"  "<< a4 <<"\n"<< endl;
    cout << "  Перегрузка оператора вызова функции класса Complex  " << endl;
    cout << "  как операции получения тригенометрической формы числа \n  вызов a1()\n" << endl;
    cout <<"  "<< a1()<< endl;
    cout << "\n  Перегрузка оператора входного потока класса Complex  \n" << endl;
    Complex a5;
    cout << "  Введите комплексноечисло в виде -> ""8,1+5,2*i""  ->  " << endl;
    cin >> a5;
    cout <<"  Вы ввели число ->  "<< a5 << endl;
    cout << "  Тригенометрическая форма числа ->  " << a5() << endl;
    _getwch();
    return 0;
}
```

Файл Complex.h

```
#pragma warning(disable : 4996)
#include <iostream>
#include <string>
#include <locale>
#include <cstdlib>
#include <cmath>

using namespace std;

class Complex
{
	double re;
	double im;
	char* number;
	static int count;

public:
	// Конструкторы
	Complex(double _re, double _im);
	Complex();
	Complex(const Complex& object);

	// Деструктор
	~Complex();

	// Методы доступа к полям
	double getRe();
	double getIm();

	// Методы
	void addition(const Complex&);
	void subtraction(const Complex&);
	void multiplication(const Complex&);
	void division(const Complex&);
	void print();
	char* toSring(double, double);
	static void Numobject();

	// Переопределение операторов методы класса
	Complex& operator= (const Complex&);
	Complex operator+ (const Complex&);
	Complex operator- (const Complex&);
	Complex operator* (const Complex&);
	Complex operator/ (const Complex&);
	void operator() (const char*);
	char* operator() ();

	// Переопределение операторов дружеств. ф-ми.
	friend bool operator== (const Complex&, const Complex&);
	friend ostream& operator<<(ostream&, const Complex&);
	friend istream& operator>>(istream&, Complex&);
};
```

Файл Complex.cpp

```
#include "Complex.h"

// Конструктор с параметрами
Complex::Complex(double _re = 0, double _im = 0) {
	//std::cout << "Конструктор с параметрами  " << this << std::endl;
	re = _re;
	im = _im;
	char* _number = toSring(re, im);
	number = new char[strlen(_number) + 1];
	strcpy(number, _number);
	delete[] _number;
	count++;
}

// Конструктор без параметров
Complex::Complex() {
	//std::cout << "Конструктор без параметров  " << this << std::endl;
	re = im = 0.0;
	char* _number = toSring(re, im);
	number = new char[strlen(_number) + 1];
	strcpy(number, _number);
	delete[] _number;
	count++;
}

//Конструктор копирования
Complex::Complex(const Complex& object)
{
	//std::cout << "Конструктор копирования  " << this << std::endl;
	re = object.re;
	im = object.im;
	number = _strdup(object.number);
	count++;
}

// Деструктор
Complex::~Complex()
{
	//std::cout << "Вызов деструктора  " << this << std::endl;
	if (number)
		delete[] number;
	number = nullptr;
	count--;
}

// Методы доступа к полям
double Complex::getRe() { return re; }
double Complex::getIm() { return im; }

#pragma region // Методы
// Методы
void Complex::addition(const Complex& a2)
{
	re = re + a2.re;
	im = im + a2.im;
	number = toSring(re, im);
}

void Complex::subtraction(const Complex& a2)
{
	re = re - a2.re;
	im = im - a2.im;
	number = toSring(re, im);
}

void Complex::multiplication(const Complex& a2)
{
	re = (re * a2.re - im * a2.im);
	im = (re * a2.im + a2.re * im);
	this->number = toSring(re, im);
}
```

```
void Complex::division(const Complex& a2)
{
	re = (re * a2.re + im * a2.im) / ((a2.re * a2.re) + (a2.im * a2.im));
	this->im = (a2.re * im - re * a2.im) / ((a2.re * a2.re) + (a2.im * a2.im));
	this->number = toSring(re, im);
}

void Complex::print()
{
	std::cout << "( " << number << " )\n\n";
}

char* Complex::toSring(double _re, double _im)
{
	char* _number = new char[50];
	char buf[20];
	_gcvt_s(buf, _re, 5);
	strcpy(_number, buf);
	if (_im > 0) {
		_number = strcat(_number, "+");
	}
	_gcvt_s(buf, _im, 5);
	_number = strcat(_number, buf);
	_number = strcat(_number, "*i");
	return _number;
}

int Complex::count = 0;

void Complex::Numobject()
{
	std::cout << count << std::endl;
}

#pragma endregion

#pragma region Переопределение операторов методы класса
// Переопределение операторов методы класса
Complex& Complex::operator=(const Complex& object)
{
	if (number != NULL) delete[]number;
	re = object.re;
	im = object.im;
	number = _strdup(object.number);
	return *this;
}

Complex Complex::operator+(const Complex& a2)
{
	Complex rez(0.0, 0.0);
	rez.re = re + a2.re;
	rez.im = im + a2.im;
	rez.number = toSring(rez.re, rez.im);
	return rez;
}
Complex Complex::operator-(const Complex& a2)
{
	Complex rez(0.0, 0.0);
	rez.re = re - a2.re;
	rez.im = im - a2.im;
	rez.number = toSring(rez.re, rez.im);
	return rez;
}
Complex Complex::operator*(const Complex& a2)
{
	Complex rez(0.0, 0.0);
	rez.re = (re * a2.re - im * a2.im);
	rez.im = (re * a2.im + a2.re * im);
	rez.number = toSring(rez.re, rez.im);
	return rez;

}
Complex Complex::operator/(const Complex& a2)
```

```
{
	Complex rez(0.0, 0.0);
	rez.re = (re * a2.re + im * a2.im) / ((a2.re * a2.re) + (a2.im * a2.im));
	rez.im = (a2.re * im - re * a2.im) / ((a2.re * a2.re) + (a2.im * a2.im));
	rez.number = toSring(rez.re, rez.im);
	return rez;
}
void Complex::operator() (const char* _string)
{
	char* ptrEnd;
	re = strtod(_string, &ptrEnd);
	im = strtod(ptrEnd, NULL);
	number = toSring(re, im);
}
char* Complex::operator()()
{
	char* _trig_number = new char[50];

	char buf[20];
	const char* _cos = " * ( cos( ";
	const char* _sin = "' ) + i*sin( ";
	const char* _brecet = "' ) )";
	double Pi = 3.14;
	double _abs = sqrt(pow(this->re, 2) + pow(this->im, 2));
	double _arg = (atan(this->im / this->re) * (180 / Pi));

	_gcvt_s(buf, _abs, 3);
	strcpy(_trig_number, buf);
	_trig_number = strcat(_trig_number, _cos);
	_gcvt_s(buf, _arg, 3);
	_trig_number = strcat(_trig_number, buf);
	_trig_number = strcat(_trig_number, _sin);
	_trig_number = strcat(_trig_number, buf);
	_trig_number = strcat(_trig_number, _brecet);
	return _trig_number;
}
#pragma endregion

#pragma region Переопределение операторов дружеств. ф-ми.
// Переопределение операторов дружеств. ф-ми.
bool operator== (const Complex& a1, const Complex& a2)
{
	if (a1.re == a2.re && a1.im == a2.im) {
		return true;
	}
	else return false;
}
ostream& operator<<(ostream& os, const Complex& a1)
{
	os << a1.number;
	return os;
}
istream& operator>>(istream& is, Complex& a1)
{
	char temp[1024];
	is >> temp;
	if (a1.number != 0) {
		delete[]a1.number;
	}
	char* ptrEnd;
	a1.re = strtod(temp, &ptrEnd);
	a1.im = strtod(ptrEnd, NULL);
	a1.number = a1.toSring(a1.re,a1.im);
	return is;
}
```