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

Учебный Центр Информационных Технологий «Информатика»

Лабораторная работа № 3
по дисциплине «Объектно-ориентированное программирование»

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

Новосибирск, 2019г.

Задание: Для класса предыдущей лабораторной работы реализовать иерархию классов-наследников, перегрузив в них отдельные методы и добавляя члены-данные и методы по заданию и/или усмотрению студента. В иерархию должно входить 2 производных класса.

Добавить в классы обработку исключений при возникновении ошибок: недостатка памяти, выход за пределы диапазона допустимых значений, деление на ноль и т.д.
Изменить демонстрационную программу так, чтобы она демонстрировала создание, копирование объектов родственных типов, вызов методов классов. Добавить в основную программу демонстрацию возникновения и обработки исключений

Класс для работы с комплексными числами. Создать наследников класса:

- класс, хранящий число так же в виде строки и имеющий методы перевода строки в число и назад;
- дополнить класс с датой создания объекта.

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 Complex

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

using namespace std;

class Complex
{
	static int count;

protected:
	double re;
	double im;
	char* number;

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

	// Деструктор
	virtual ~Complex();

	// Методы доступа к полям
	double getRe();
	double getIm();
	char* getNumberComplex();

	// 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 ввода

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

Шаблон класса Data

```
#pragma once
#include "complex.h"

class Date: public Complex {

	int day, mon, year;

public:

 //Вложенный класс обработки исключений
	class ParamExeption {

	public:

		int day, mon, year;

		ParamExeption(int day, int mon, int year) {
			this->day = day;
			this->mon = mon;
			this->year = year;
		}
	};

	// конструкторы
	Date();
	Date(double _re, double _im, int _day, int _mon, int _year) throw (ParamExeption);
	Date(const Date& object);

	// деструктор
	~Date();

	// методы класса
	void getdate();
	void setdate(int hour, int min, int sec);

	// переопределение потоков ввода вывода
	friend ostream& operator<<(ostream& os, const Date& dt);
	friend istream& operator>>(istream& is, Date& dt);
};
```

Поля класса:

- int day – день,
- int mon – месяц,
- int year – год,

Вложенный класс обработки исключений ParamExeption

Поля класса:

- int day – день,
- int mon – месяц,
- int year – год,

Конструктор

- ParamExeption(int day, int mon, int year)

Конструкторы:

- Date(double _re, double _im, int _day, int _mon, int _year) throw (ParamExeption); – конструктор с параметрами
- Date();– конструктор без параметров
- Date(const Date& object);– конструктор копирования

Деструктор:

- ~Date();– деструктор объектов класса без параметров

Методы класса:

- void getdate();– вывод даты в косоль
- void setdate(int hour, int min, int sec);– установка даты

Перегрузка операторов:

- friend ostream& operator<<(ostream& os, const Date& dt);– вывод в поток
- friend istream& operator>>(istream& is, Date& dt);– чтение из потока

// **Конструктор с параметрами**
Date(double _re, double _im, int _day, int _mon, int _year)
Параметры:
double _re **–** действительная часть числа.
double _im **–** мнимая часть числа.
int day – день,
int mon – месяц,
int year – год,

Возвращаемый результат: ничего не возвращает.
Принцип работы: создает объект класса Date.

// **Конструктор без параметров**
Date( )
Параметры:

Возвращаемый результат: ничего не возвращает.
Принцип работы: создает объект класса Date.

// **Конструктор копирования**
Date(const Date& object)
Параметры:
const Date& object **–** объект класса Date.

Возвращаемый результат: ничего не возвращает.
Принцип работы: выполняет копирование объектов.

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

// **Метод получения даты**
void getdate()
Параметры:

Возвращаемый результат: ничего не возвращает.
Принцип работы: выводит дату создания объекта.

// **Метод установки даты**
void setdate(int hour, int min, int sec);
Параметры:
int day – день,
int mon – месяц,
int year – год,

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

// **Оператор вывода в поток**
friend ostream& operator<<(ostream& os, const Date& dt);
Параметры:
const Date& dt – объект типа Date
ostream& os – поток вывода

Возвращаемый результат: поток вывода
Принцип работы: выводит в поток объект типа Date

// **Оператор ввода в поток**
friend istream& operator>>(istream& is, Date& dt);
Параметры:
Date& dt – объект типа Date
ostream& is – поток ввода

Возвращаемый результат: поток ввода
Принцип работы: считывает из потока объект типа Date.

Шаблон класса ComString

```
#pragma once
#include "complex.h"

class ComString : public Complex
{
	char* str_number_math;
	char* str_number_geom;

public:
	class ParamExeption {
	public:
		const char* str;
		double re;
		double im;

		ParamExeption(double _re, double _im, const char* _str) {
			this->re = _re;
			this->im = _im;
			this->str = _str;
		}
	};

	// конструкторы
	ComString();
	ComString(const char* str);
	ComString(double, double);
	ComString(const ComString&);

	// деструктор
	~ComString();

	// Методы доступа к полям
	char* getNumber();
	char* getTrigNumber();

	// Методы
	char* toTrigSring();
	char* convertToString();
	Complex convertToComplex();

	// Переопределение операторов потока
	friend ostream& operator<<(ostream&, const ComString&);
	friend istream& operator>>(istream& is, ComString& object);
};
```

Поля класса:

- char* str_number_math;– строка алгебраическая форма числа,
- char* str_number_geom – строка тригенометрическая форма числа,

Вложенный класс обработки исключений ParamExeption

Поля класса:

- int day – день,
- int mon – месяц,
- int year – год,

Конструктор
ParamExeption(int day, int mon, int year)

// Конструктор с параметрами
ComString(const char* str);
Параметры:
char* str **–** строка содержащая запись комплексного числа.

Возвращаемый результат: ничего не возвращает.
Принцип работы: создает объект класса ComString.

// Конструктор с параметрами
ComString(double re, double im);
Параметры:
double _re **–** действительная часть числа.
double _im **–** мнимая часть числа.

Возвращаемый результат: ничего не возвращает.
Принцип работы: создает объект класса ComString.

// Конструктор без параметров
ComString();
Параметры:

Возвращаемый результат: ничего не возвращает.
Принцип работы: создает объект класса ComString.

// Конструктор копирования
ComString(const ComString& object);
Параметры:
const ComString& object **–** объект класса ComString.

Возвращаемый результат: ничего не возвращает.
Принцип работы: выполняет копирование объектов

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

// Метод получения тригенометрической формы числа
char* toTrigSring();
Параметры:

Возвращаемый результат: строка
Принцип работы: записывает строковое представление числа в поле str_number_geom.

// Метод конвертирования числа в строку
char* convertToString();
Параметры:

Возвращаемый результат: строка
Принцип работы: конвертирует объект типа ComString в строку.

// Метод конвертирования оъекта типа ComString в объект типа Complex
Complex convertToComplex();
Параметры:

Возвращаемый результат: строка
Принцип работы: объект типа Complex.

// Оператор вывода в поток
friend ostream& operator<<(ostream& os, const ComString& object);
Параметры:
const ComString& object – объект типа ComString
ostream& os – поток вывода

Возвращаемый результат: поток вывода
Принцип работы: выводит в поток объект типа ComString

// Оператор ввода в поток
friend istream& operator>>(istream& is, ComString& object);
Параметры:
ComString& object – объект типа Date
ostream& is – поток ввода

Возвращаемый результат: поток ввода
Принцип работы: считывает из потока объект типа ComString.

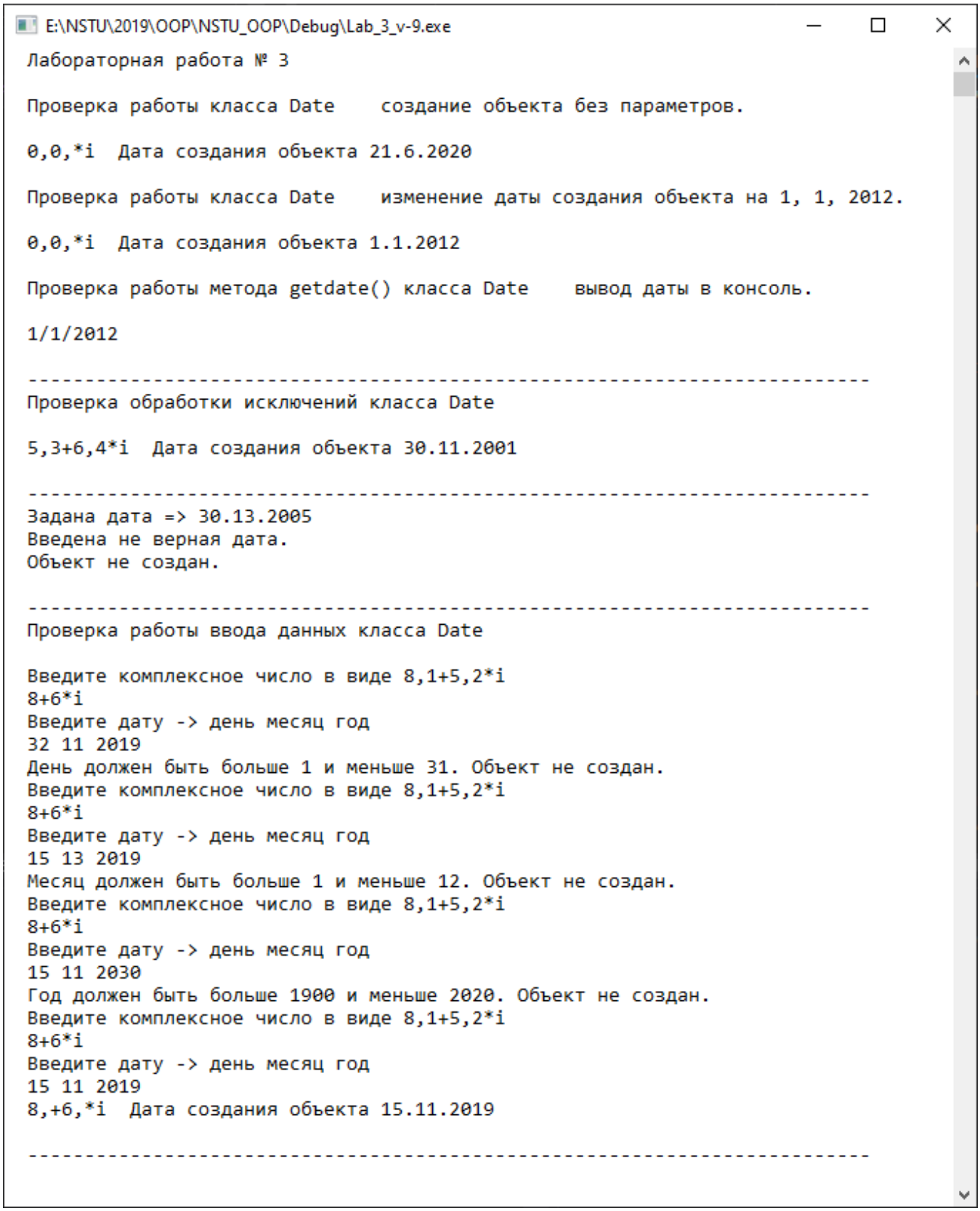
3. Листинг тестовой программы.

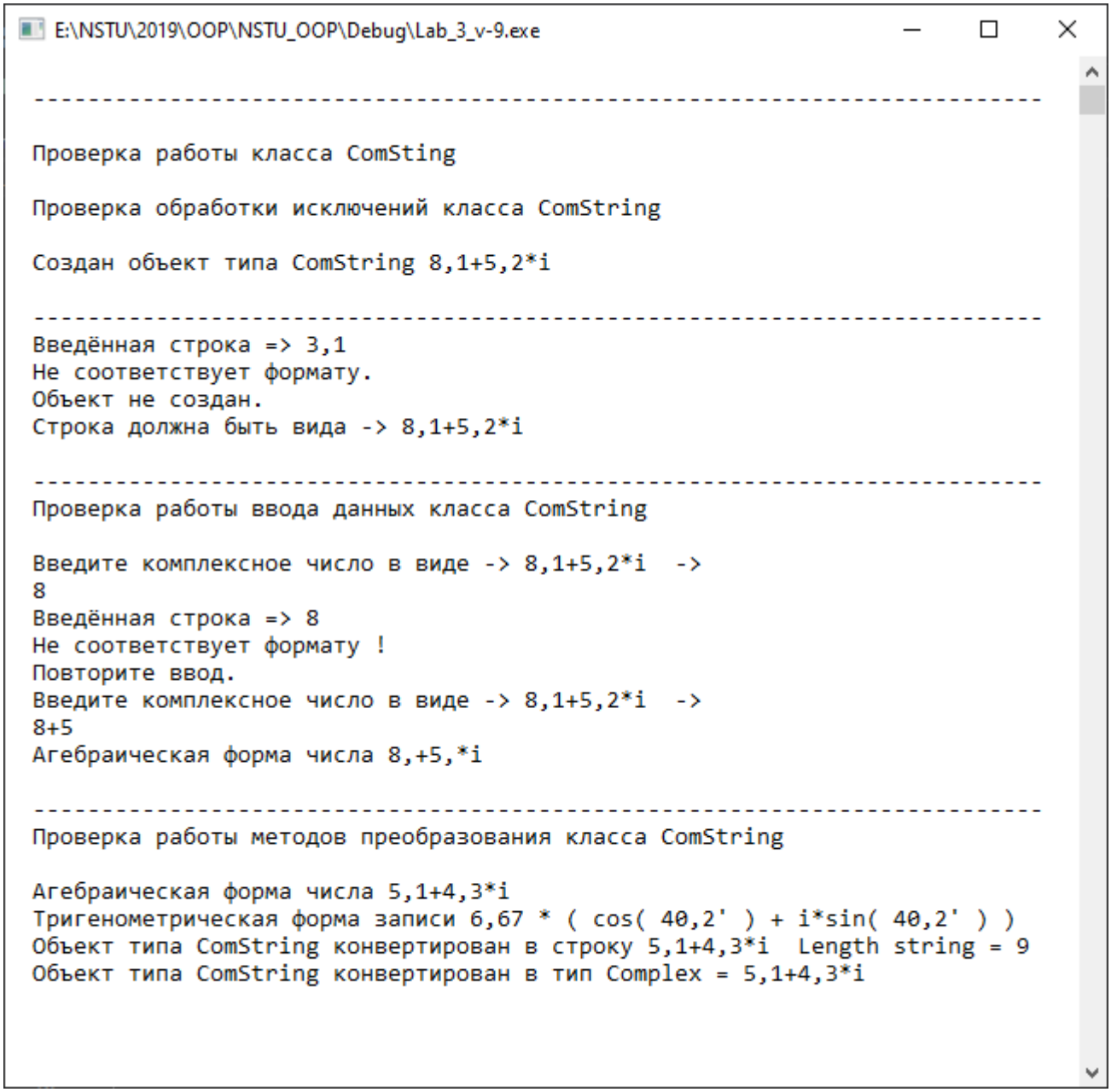
```
#include "complex.h"
#include "com_date.h"
#include "com_string.h"

void print_line();

int main()
{
   setlocale(LC_ALL, "Russian");
   Date obj_1, * obj_2 = nullptr, *obj_3 = nullptr;

   cout << "  Лабораторная работа № 3" << endl;

   cout << "\n  Проверка работы класса Date  "
      << "  создание объекта без параметров.\n" << endl;
   Date test_date;
   cout << "  " << test_date << endl;

   cout << "\n  Проверка работы класса Date  "
      << "  изменение даты создания объекта на 1, 1, 2012.\n" << endl;
   test_date.setdate(1, 1, 2012);
   cout << "  " << test_date << endl;

   cout << "\n  Проверка работы метода getdate() класса Date  "
      << "  вывод даты в консоль.\n" << endl;
   test_date.getdate();

   print_line();
   cout << "  Проверка обработки исключений класса Date  \n" << endl;

   try
   {
      obj_2 = new Date(5.3, 6.4, 30, 11, 2001);
      cout << "  " << *obj_2 << endl;
   }
   catch (Date::ParamExeption e)
   {
      cout << "  Задана дата => " << e.day<<"."<<e.mon<<"."<<e.year << endl;
      cout << "  Введена не верная дата. \n  Объект не создан." << endl;
   }
   print_line();
   try
   {
      obj_3 = new Date(3.3, 6.4, 30, 13, 2005);
      cout << "  " << *obj_3 << endl;
   }
   catch (Date::ParamExeption e)
   {
      cout << "  Задана дата => " << e.day << "." << e.mon << "." << e.year << endl;
      cout << "  Введена не верная дата. \n  Объект не создан." << endl;
   }

   print_line();
   cout << "  Проверка работы ввода данных класса Date  \n" << endl;
```

```
//cout << "  " << obj_1 << endl;
do {
   try {
      cin >> obj_1;
      break;
   }
   catch (const char* str)
   {
      cout << str << endl;
      cin.clear();
      cin.ignore(32767, '\n');
   }
} while (true);
cout << "  " << obj_1<< endl;
print_line();

_getwch();
system("cls");

print_line();
cout << "\n  Проверка работы класса ComSting  \n" << endl;

ComString str_1, *str_2 = nullptr, * str_3 = nullptr;

cout << "  Проверка обработки исключений класса ComString  \n" << endl;
try
{
   str_2 = new ComString("8,1+5,2*i");
   cout <<"  Создан объект типа ComString "<< *str_2 << endl;
}
catch (ComString::ParamExeption e)
{
   cout << "  Введённая строка => " << e.str << endl;
   cout << "  Не соответствует формату. \n  Объект не создан." << endl;
   cout << "  Строка должна быть вида -> ""8,1+5,2*i""" << endl;
}
print_line();
try
{
   str_3 = new ComString("3,1");
   cout << "  Создан объект типа ComString " << *str_3 << endl;
}
catch (ComString::ParamExeption e)
{
   cout << "  Введённая строка => " << e.str << endl;
   cout << "  Не соответствует формату. \n  Объект не создан." << endl;
   cout << "  Строка должна быть вида -> ""8,1+5,2*i""" << endl;
}


print_line();
cout << "  Проверка работы ввода данных класса ComString  \n" << endl;

do {
   try {
      cin >> str_1;
```

```
            break;
        }
        catch (ComString::ParamExeption e)
        {
            cout << "  Введённая строка => " << e.str << endl;
            cout << "  Не соответствует формату ! \n  Повторите ввод." << endl;
            cin.clear();
            cin.ignore(32767, '\n');
        }
    } while (true);
    cout << "  Агебраическая форма числа " << str_1 << endl;


    print_line();
    cout << "  Проверка работы методов преобразования класса ComString  \n" << endl;

    ComString str_4(5.1,4.3);

    cout << "  Агебраическая форма числа " << str_4 << endl;
    cout << "  Тригенометрическая форма записи " << str_4.getTrigNumber() << endl;

    char* test = str_4.convertToString();

    int len = strlen(test);
    cout <<"  Объект типа ComString конвертирован в строку "<< test
         <<"  Length string = "<<len<<endl;

    Complex com_1 = str_4.convertToComplex();

    cout <<"  Объект типа ComString конвертирован в тип Complex = " << com_1 << endl;
    _getwch();
    return 0;
}

void print_line()
{
    cout << "\n  ";
    for (int i = 0; i < 74; i++)
    {
        cout << "-";
    }
    cout << "\n";
}
```

4. Пример работы программы.




E:\NSTU\2019\OOP\NSTU_OOP\Debug\Lab_3_v-9.exe

Проверка работы класса ComSting
Проверка обработки исключений класса ComString
Создан объект типа ComString 8,1+5,2*i

Введённая строка => 3,1
Не соответствует формату.
Объект не создан.
Строка должна быть вида -> 8,1+5,2*i

Проверка работы ввода данных класса ComString
Введите комплексное число в виде -> 8,1+5,2*i ->
8
Введённая строка => 8
Не соответствует формату !
Повторите ввод.
Введите комплексное число в виде -> 8,1+5,2*i ->
8+5
Агебраическая форма числа 8,+5,*i

Проверка работы методов преобразования класса ComString
Агебраическая форма числа 5,1+4,3*i
Тригенометрическая форма записи 6,67 * (cos(40,2') + i*sin(40,2'))
Объект типа ComString конвертирован в строку 5,1+4,3*i Length string = 9
Объект типа ComString конвертирован в тип Complex = 5,1+4,3*i

Приложение. Текст программы

Файл main.cpp

```
#include "complex.h"
#include "com_date.h"
#include "com_string.h"

void print_line();
int main()
{
    setlocale(LC_ALL, "Russian");
    Date obj_1, * obj_2 = nullptr, *obj_3 = nullptr;
    cout << "  Лабораторная работа № 3" << endl;
    cout << "\n  Проверка работы класса Date  "
         << "  создание объекта без параметров.\n" << endl;
    Date test_date;
    cout << "  " << test_date << endl;
    cout << "\n  Проверка работы класса Date  "
         << "  изменение даты создания объекта на 1, 1, 2012.\n" << endl;
    test_date.setdate(1, 1, 2012);
    cout << "  " << test_date << endl;
    cout << "\n  Проверка работы метода getdate() класса Date  "
         << "  вывод даты в консоль.\n" << endl;
    test_date.getdate();
    print_line();
    cout << "  Проверка обработки исключений класса Date  \n" << endl;
    try
    {
        obj_2 = new Date(5.3, 6.4, 30, 11, 2001);
        cout << "  " << *obj_2 << endl;
    }
    catch (Date::ParamExeption e)
    {
        cout << "  Задана дата => " << e.day<<"."<<e.mon<<"."<<e.year << endl;
        cout << "  Введена не верная дата. \n  Объект не создан." << endl;
    }
    print_line();
    try
    {
        obj_3 = new Date(3.3, 6.4, 30, 13, 2005);
        cout << "  " << *obj_3 << endl;
    }
    catch (Date::ParamExeption e)
    {
        cout << "  Задана дата => " << e.day << "." << e.mon << "." << e.year << endl;
        cout << "  Введена не верная дата. \n  Объект не создан." << endl;
    }
    print_line();
    cout << "  Проверка работы ввода данных класса Date  \n" << endl;
    do {
        try {
            cin >> obj_1;
            break;
        }
        catch (const char* str)
        {
            cout << str << endl;
            cin.clear();
            cin.ignore(32767, '\n');
        }
    } while (true);
    cout << "  " << obj_1<< endl;
    print_line();
```

```
    _getwch();
    system("cls");
    print_line();
    cout << "\n  Проверка работы класса ComSting  \n" << endl;
    ComString str_1, *str_2 = nullptr, * str_3 = nullptr;
    cout << "  Проверка обработки исключений класса ComString  \n" << endl;
    try
    {
        str_2 = new ComString("8,1+5,2*i");
        cout <<"  Создан объект типа ComString "<< *str_2 << endl;
    }
    catch (ComString::ParamExeption e)
    {
        cout << "  Введённая строка => " << e.str << endl;
        cout << "  Не соответствует формату. \n  Объект не создан." << endl;
        cout << "  Строка должна быть вида -> ""8,1+5,2*i""" << endl;
    }
    print_line();
    try
    {
        str_3 = new ComString("3,1");
        cout << "  Создан объект типа ComString " << *str_3 << endl;
    }
    catch (ComString::ParamExeption e)
    {
        cout << "  Введённая строка => " << e.str << endl;
        cout << "  Не соответствует формату. \n  Объект не создан." << endl;
        cout << "  Строка должна быть вида -> ""8,1+5,2*i""" << endl;
    }
    print_line();
    cout << "  Проверка работы ввода данных класса ComString  \n" << endl;
    do {
        try {
            cin >> str_1;
            break;
        }
        catch (ComString::ParamExeption e)
        {
            cout << "  Введённая строка => " << e.str << endl;
            cout << "  Не соответствует формату ! \n  Повторите ввод." << endl;
            cin.clear();
            cin.ignore(32767, '\n');
        }
    } while (true);
    cout << "  Агебраическая форма числа " << str_1 << endl;
    print_line();
    cout << "  Проверка работы методов преобразования класса ComString  \n" << endl;
    ComString str_4(5.1,4.3);
    cout << "  Агебраическая форма числа " << str_4 << endl;
    cout << "  Тригенометрическая форма записи " << str_4.getTrigNumber() << endl;
    char* test = str_4.convertToString();
    int len = strlen(test);
    cout <<"  Объект типа ComString конвертирован в строку "<< test
         <<"  Length string = "<<len<<endl;
    Complex com_1 = str_4.convertToComplex();
    cout <<"  Объект типа ComString конвертирован в тип Complex = " << com_1 << endl;
    _getwch();
    return 0;
}
void print_line()
{ cout << "\n  ";
    for (int i = 0; i < 74; i++)
    {
        cout << "-";
    }
    cout << "\n";
}
```

Файл complex.h

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

using namespace std;

class Complex
{
	static int count;

protected:
	double re;
	double im;
	char* number;

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

	// Деструктор
	virtual ~Complex();

	// Методы доступа к полям
	double getRe();
	double getIm();
	char* getNumberComplex();

	// 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;
}
```

Файл com_string.h

```
#pragma once
#include "complex.h"

class ComString : public Complex
{
	char* str_number_math;
	char* str_number_geom;

public:
	class ParamExeption {
	public:
		const char* str;
		double re;
		double im;

		ParamExeption(double _re, double _im, const char* _str) {
			this->re = _re;
			this->im = _im;
			this->str = _str;
		}
	};

	// конструкторы
	ComString();
	ComString(const char* str);
	ComString(double, double);
	ComString(const ComString&);

	// деструктор
	~ComString();

	// Методы доступа к полям
	char* getNumber();
	char* getTrigNumber();

	// Методы
	char* toTrigSring();
	char* convertToString();
	Complex convertToComplex();

	// Переопределение операторов потока
	friend ostream& operator<<(ostream&, const ComString&);
	friend istream& operator>>(istream& is, ComString& object);
};
```

Файл com_string.cpp

```
#include "com_string.h"

// Конструктор без параметров
ComString::ComString() :Complex()
{
	//std::cout << "Конструктор без параметров наследника  " << this << std::endl;
	str_number_math = number;
	str_number_geom = toTrigSring();
}

// Конструктор для строки
ComString::ComString(const char* str):Complex()
{
	//std::cout << "Конструктор с строковым параметром ComString  " << this << std::endl;
	char* ptrEnd;
	double _re = strtod(str, &ptrEnd);
	double _im = strtod(ptrEnd, NULL);
```

```
	if (_re == 0.0 || _im == 0.0) throw ParamExeption(_re, _im, str);
	re = _re;
	im = _im;
	number = toSring(re, im);
	str_number_math = number;
	str_number_geom = toTrigSring();
}

// Конструктор для чисел
ComString::ComString(double _re, double _im) :Complex(_re, _im)
{
	//std::cout << "  Конструктор с числовыми параметрами ComString " << this <<
std::endl;
	str_number_math = number;
	str_number_geom = toTrigSring();
}

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

// Деструктор
ComString::~ComString() { }

// Методы доступа к полям класса
char* ComString::getNumber() { return str_number_math; }
char* ComString::getTrigNumber() { return str_number_geom; }

// Методы класса
char* ComString::toTrigSring()
{
	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;
}

char* ComString::convertToString()
{
	char* _string = new char[50];
	char buf[20];
	_gcvt_s(buf, re, 5);
	strcpy(_string, buf);
	if (im > 0) {
		_string = strcat(_string, "+");
	}
```

```
	_gcvt_s(buf, im, 5);
	_string = strcat(_string, buf);
	_string = strcat(_string, "*i");
	return _string;
}

Complex ComString::convertToComplex()
{
	double _re = re;
	double _im = im;
	Complex rez(_re, _im);
	return rez;
}

//Переопределение операторов потока
ostream& operator<<(ostream& os, const ComString& object)
{
	//os << (Complex)object;
	os << object.str_number_math;
	return os;
}

istream& operator>>(istream& is, ComString& object)
{
	char temp[30];
	cout << "  Введите комплексное число в виде -> ""8,1+5,2*i""  ->  " << endl;
	is >> temp;

	char* ptrEnd;
	double _re = strtod(temp, &ptrEnd);
	double _im = strtod(ptrEnd, NULL);
	if (_re == 0.0 || _im == 0.0)
		throw ComString::ParamExeption(_re, _im, temp);

	object.re = _re;
	object.im = _im;
	object.number = object.toSring(_re, _im);
	object.str_number_math = object.number;

	return is;
}
```

Файл com_date.h

```
#pragma once
#include "complex.h"

class Date: public Complex {

	int day, mon, year;

public:

 //Вложенный класс
	class ParamExeption {

	public:

		int day, mon, year;

		ParamExeption(int day, int mon, int year) {
			this->day = day;
			this->mon = mon;
			this->year = year;
		}
	};
```

```
	// конструкторы
	Date();
	Date(double _re, double _im, int _day, int _mon, int _year) throw (ParamExeption);
	Date(const Date& object);

	// деструктор
	~Date();

	// методы класса
	void getdate();
	void setdate(int hour, int min, int sec);

	// переопределение потоков ввода вывода
	friend ostream& operator<<(ostream& os, const Date& dt);
	friend istream& operator>>(istream& is, Date& dt);
};
```

Файл com_date.cpp

```
#include "com_date.h"

Date::Date() : Complex()
{
	time_t curent_time;
	struct tm* _time;
	curent_time = time(NULL);
	_time = localtime(&curent_time);
	day = _time->tm_mday;
	mon = _time->tm_mon + 1;
	year = _time->tm_year + 1900;
}

Date::Date(double _re, double _im, int _day, int _mon, int _year) : Complex(_re, _im)
{
	if (_day < 1 || _day > 31) throw ParamExeption(_day,_mon,_year);
	day = _day;
	if (_mon < 1 || _mon > 12) throw ParamExeption(_day,_mon,_year);
	mon = _mon;
	if (_year < 1900 || _year > 2020) throw ParamExeption(_day,_mon,_year);
	year = _year;
}

Date::Date(const Date& object) : Complex (object)
{
	day = object.day;
	mon = object.mon;
	year = object.year;
}

Date::~Date() {    }

void Date::getdate() {
	cout<<"  " << day << "/" << mon << "/" << year << endl;
}

void Date::setdate(int day, int mon, int year) {
	this->day = day;
	this->mon = mon;
	this->year = year;
}

ostream& operator<<(ostream& os, const Date& object) {
	os << (Complex)object;
```

```
	os << "  Дата создания объекта "<< object.day << "." << object.mon << "." <<
object.year;
	return os;
}

istream& operator>>(istream& is, Date& object)
{
	is >> (Complex&)object;
	cout << "  Введите дату -> день месяц год " << endl;
	is >> object.day >> object.mon >> object.year;
	if (object.day < 1 || object.day > 31) {

		throw "  День должен быть больше 1 и меньше 31. Объект не создан.";
	}
	else if (object.mon < 1 || object.mon > 12)
	{
		throw "  Месяц должен быть больше 1 и меньше 12. Объект не создан.";
	}
	else if (object.year < 1900 || object.year > 2020)
	{
		throw "  Год должен быть больше 1900 и меньше 2020. Объект не создан.";
	}
	else return is;
}
```