

Федеральное агентство связи
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Сибирский государственный университет
телекоммуникаций и информатики»

Лабораторная работа №6

Выполнил: студент 4 курса

ИВТ, гр. ИП-713

Михеев Н.А.

Проверил: ассистент кафедры

ПМиК

Агалаков А.А.

Новосибирск, 2020 г.

Цель

Сформировать практические навыки реализации классов средствами объектно-ориентированного программирования C++.

Задание

1. Разработать и реализовать класс «Ввод и редактирование комплексных чисел» (TEditor), используя класс C++.
2. Протестировать каждую операцию, определенную на типе данных, используя средства модульного тестирования.
3. Если необходимо, предусмотрите возбуждение исключительных ситуаций.

Реализация и описание

В ходе реализации лабораторной работы был реализован класс TComplexEditor с соответствующими методами, далее подробнее про каждый из них:

- static const std::string zeroComplex – статичная строка с нулевым комплексным числом.
- static const std::string impart – статичная строка, содержащая мнимую часть комплексного числа
- TComplexEditor() – базовый конструктор класса, который инициализирует нулевое комплексное число;
- bool isZero() – булевый метод, необходимый для определения, является ли комплексное число нулевым;
- std::string addSign() – метод, добавляет или удаляется знак минус из комплексной строки;
- std::string addNumber(int a) – метод, добавляющий заданное число a к комплексной строке;
- std::string addZero() – метод, добавляющий нуль к комплексной строке;
- std::string addImPart() – метод, добавляющий мнимую часть комплексного числа, в случае добавления мнимой части к строке уже имеющей мнимую часть – выбрасывается исключение;
- std::string removeLastDigit() – метод затирающий крайний символ в комплексной строке;
- std::string clear() – устанавливает пустое комплексное число формата 0+i*0;
- void editComplex(ComplexOperations operation) – метод, который получив операцию, выполняет ее с комплексной строкой;

- `std::string getComplexString()` – метод получения строкового представления комплексной строки;
- `std::string setComplexString(std::string str)` – метод задания комплексной строки, в случае невалидности данной строки выбрасывается исключение;
- `bool isValid(std::string)` – булева вспомогательная функция, необходимая для проверки комплексной строки на валидность;

Заключение

В ходе данной работы согласно спецификациям задания был реализован класс `TComplexEditor`, были протестированы все операции с использованием средств модульного тестирования.

Скриншоты



```
Консоль отладки Microsoft Visual Studio
TComplexEditor:
Init: 0
isZero: 1
Add Number 5: -15
setComplexString 6-i*15: 6-i*15
Add sign: -6-i*15
```

Рис. 1 – пример работы программы

Тестирование	Длительности	Признаки
✓ MPT_Test (241)	27 мс	
✓ MPT_Tests (241)	27 мс	
✓ TComplexEditorTest (19)	3 мс	
✓ TComplexEditor_addImPart	< 1 мс	
✓ TComplexEditor_addImPart2	< 1 мс	
✓ TComplexEditor_addImPart3	2 мс	
✓ TComplexEditor_addNumber	< 1 мс	
✓ TComplexEditor_addNumber2	< 1 мс	
✓ TComplexEditor_addSign	< 1 мс	
✓ TComplexEditor_addSign2	1 мс	
✓ TComplexEditor_addSign3	< 1 мс	
✓ TComplexEditor_construction	< 1 мс	
✓ TComplexEditor_editComplex	< 1 мс	
✓ TComplexEditor_editComplex2	< 1 мс	
✓ TComplexEditor_editComplex3	< 1 мс	
✓ TComplexEditor_editComplex4	< 1 мс	
✓ TComplexEditor_editComplex5	< 1 мс	
✓ TComplexEditor_isZero	< 1 мс	
✓ TComplexEditor_isZero_2	< 1 мс	
✓ TComplexEditor_setComplexString	< 1 мс	
✓ TComplexEditor_setComplexString2	< 1 мс	
✓ TComplexEditor_setComplexString3	< 1 мс	

Рис. 2 – сводка проведённого тестирования программы

Код программы

TComplexEditor.h:

```
#pragma once
#define _USE_MATH_DEFINES
#include <iostream>
#include <math.h>
#include <string>
#include <sstream>
#include <exception>

enum class ComplexOperations
{
    ADD_SIGN,
    ADD_DIGIT,
    ADD_ZERO,
    ADD_IMPART,
    REMOVE_LAST_DIGIT,
    CLEAR
};

enum class ComplexPart
{
```

```

        REAL_PART,
        IM_PART
};

class TComplexEditor
{
private:
    std::string complex;
    ComplexPart currentPart;
public:
    static const std::string zeroComplex;
    static const std::string imPart;
    TComplexEditor();

    bool isZero();
    std::string addSign();
    std::string addNumber(int a);
    std::string addZero();
    std::string addImPart();
    std::string removeLastDigit();
    std::string clear();
    void editComplex(ComplexOperations operation);
    std::string getComplexString();
    std::string setComplexString(std::string str);

    bool isValid(std::string);
};

```

TComplexEditor.cpp

```

#include "TComplexEditor.h"

const std::string TComplexEditor::zeroComplex = "0";
const std::string TComplexEditor::imPart = "+i*";

TComplexEditor::TComplexEditor() : complex(zeroComplex),
currentPart(ComplexPart::REAL_PART) {}

bool TComplexEditor::isZero()
{
    return complex == zeroComplex;
}

std::string TComplexEditor::addSign()
{
    if (currentPart == ComplexPart::REAL_PART)
    {
        if (complex[0] == '-')
            complex.erase(complex.begin());
        else if (complex[0] != '0')
            complex = '-' + complex;
    }
    else
    {
        auto pos = complex.find('i');
        if (complex[pos - 1] == '-')
            complex[pos - 1] = '+';
        else
            complex[pos - 1] = '-';
    }
    return complex;
}

std::string TComplexEditor::addNumber(int a)

```

```

{
    if (currentPart == ComplexPart::IM_PART)
        complex += std::to_string(a);
    else
    {
        auto pos = complex.find('i');
        if (complex[0] == '0')
            complex.replace(0, 1, std::to_string(a));
        else if (pos != -1)
            complex.insert(pos - 1, std::to_string(a));
        else if (complex[0] == '-')
            complex.insert(1, std::to_string(a));
        else
            complex += std::to_string(a);
    }
    return complex;
}

std::string TComplexEditor::addZero()
{
    return addNumber(0);
}

std::string TComplexEditor::addImPart()
{
    if (complex.find(imPart) == std::string::npos)
    {
        complex = complex + imPart;
        currentPart = ComplexPart::IM_PART;
    }
    else
    {
        throw std::logic_error("Complex number already has a image part");
    }
    return complex;
}

std::string TComplexEditor::removeLastDigit()
{
    complex.pop_back();
    if (currentPart == ComplexPart::REAL_PART)
    {
        if (complex == "-" || complex.empty())
            complex = zeroComplex;
    }
    else
    {
        if (complex[complex.length() - 1] == '*')
        {
            complex.erase(complex.end() - 3, complex.end());
            currentPart = ComplexPart::REAL_PART;
        }
        else if (complex.find('i'))
            complex.back();
    }
    return complex;
}

std::string TComplexEditor::clear()
{
    currentPart = ComplexPart::REAL_PART;
    return complex = zeroComplex;
}

void TComplexEditor::editComplex(ComplexOperations operation)

```

```

{
    switch (operation)
    {
    case ComplexOperations::ADD_SIGN:
        addSign();
        break;
    case ComplexOperations::ADD_DIGIT:
        int num;
        std::cout << "Enter number to add: ";
        std::cin >> num;
        addNumber(num);
        break;
    case ComplexOperations::ADD_ZERO:
        addZero();
        break;
    case ComplexOperations::ADD_IMPART:
        addImPart();
        break;
    case ComplexOperations::REMOVE_LAST_DIGIT:
        removeLastDigit();
        break;
    case ComplexOperations::CLEAR:
        clear();
        break;
    default:
        break;
    }
}

std::string TComplexEditor::getComplexString()
{
    return complex;
}

bool TComplexEditor::isValid(std::string str)
{
    bool res = false;
    if (!str.empty() && (str.find("+i*") != std::string::npos || str.find("-i*") !=
std::string::npos))
    {
        auto pos = str.find('i');
        std::string tempRe = str;
        std::string tempIm = str;
        std::string rePart = tempRe.erase(pos - 1, str.length());
        std::string imPart = tempIm.erase(0, pos + 2);

        int digitCount = 0;
        for (auto i : rePart) {
            if (isdigit(i))
                digitCount++;
        }
        for (auto i : imPart) {
            if (isdigit(i))
                digitCount++;
        }

        if (digitCount)
        {
            if (str[0] == '-')
            {
                if (digitCount == str.length() - 4)
                    res = true;
            }
            else
            {

```

```

        if (digitCount == str.length() - 3)
            res = true;
    }
}
return res;
}

std::string TComplexEditor::setComplexString(std::string str)
{
    if (isValid(str))
        complex = str;
    else
        throw std::invalid_argument("Wrong complex number!");
    return std::string();
}

```

TComplexEditor_Test.cpp

```

#include "pch.h"
#include "CppUnitTest.h"

#include "../MPT/TComplexEditor.h"
#include "../MPT/TComplexEditor.cpp"

using namespace Microsoft::VisualStudio::CppUnitTestFramework;

namespace MPT_Tests
{
    TEST_CLASS(TComplexEditorTest)
    {
    public:

        TEST_METHOD(TComplexEditor_construction)
        {
            std::string expected = "0";
            TComplexEditor a;
            Assert::AreEqual(expected, a.getComplexString());
        }

        TEST_METHOD(TComplexEditor_isZero)
        {
            TComplexEditor a;
            Assert::IsTrue(a.isZero());
        }

        TEST_METHOD(TComplexEditor_isZero_2)
        {
            TComplexEditor a;
            a.addNumber(1);
            Assert::IsFalse(a.isZero());
        }

        TEST_METHOD(TComplexEditor_addNumber)
        {
            std::string expected = "1";
            TComplexEditor a;
            a.addNumber(1);
            Assert::AreEqual(expected, a.getComplexString());
        }

        TEST_METHOD(TComplexEditor_addNumber2)
        {
            std::string expected = "-16";

```



```

        TComplexEditor a;
        a.addNumber(-16);
        Assert::AreEqual(expected, a.getComplexString());
    }

TEST_METHOD(TComplexEditor_addSign)
{
    std::string expected = "-16";
    TComplexEditor a;
    a.addNumber(16);
    a.addSign();
    Assert::AreEqual(expected, a.getComplexString());
}

TEST_METHOD(TComplexEditor_addSign2)
{
    std::string expected = "16+i*12";
    TComplexEditor a;
    a.setComplexString("-16+i*12");
    a.addSign();
    Assert::AreEqual(expected, a.getComplexString());
}

TEST_METHOD(TComplexEditor_addSign3)
{
    std::string expected = "-5-i*10";
    TComplexEditor a;
    a.addNumber(-5);
    a.addImPart();
    a.addNumber(10);
    a.addSign();
    Assert::AreEqual(expected, a.getComplexString());
}

TEST_METHOD(TComplexEditor_addImPart)
{
    std::string expected = "15+i*4";
    TComplexEditor a;
    a.addNumber(15);
    a.addImPart();
    a.addNumber(4);
    Assert::AreEqual(expected, a.getComplexString());
}

TEST_METHOD(TComplexEditor_addImPart2)
{
    std::string expected = "-5+i*10";
    TComplexEditor a;
    a.addNumber(-5);
    a.addImPart();
    a.addNumber(10);
    Assert::AreEqual(expected, a.getComplexString());
}

TEST_METHOD(TComplexEditor_addImPart3)
{
    auto func = [] { TComplexEditor a; a.addImPart(); a.addImPart(); };
    Assert::ExpectException<std::logic_error>(func);
}

TEST_METHOD(TComplexEditor_editComplex)
{
    std::string expected = "-8";
    TComplexEditor a;
    a.addNumber(8);

```

```

        a.editComplex(ComplexOperations::ADD_SIGN);
        Assert::AreEqual(expected, a.getComplexString());
    }

TEST_METHOD(TComplexEditor_editComplex2)
{
    std::string expected = "8";
    TComplexEditor a;
    a.addNumber(-8);
    a.editComplex(ComplexOperations::ADD_SIGN);
    Assert::AreEqual(expected, a.getComplexString());
}

TEST_METHOD(TComplexEditor_editComplex3)
{
    std::string expected = "8+i*4";
    TComplexEditor a;
    a.addNumber(8);
    a.editComplex(ComplexOperations::ADD_IMPART);
    a.addNumber(4);
    Assert::AreEqual(expected, a.getComplexString());
}

TEST_METHOD(TComplexEditor_editComplex4)
{
    std::string expected = "8+i*40";
    TComplexEditor a;
    a.addNumber(8);
    a.editComplex(ComplexOperations::ADD_IMPART);
    a.addNumber(4);
    a.editComplex(ComplexOperations::ADD_ZERO);
    Assert::AreEqual(expected, a.getComplexString());
}

TEST_METHOD(TComplexEditor_editComplex5)
{
    std::string expected = "8-i*4";
    TComplexEditor a;
    a.addNumber(8);
    a.editComplex(ComplexOperations::ADD_IMPART);
    a.addNumber(4);
    a.editComplex(ComplexOperations::ADD_SIGN);
    Assert::AreEqual(expected, a.getComplexString());
}

TEST_METHOD(TComplexEditor_setComplexString)
{
    std::string expected = "8+i*4";
    TComplexEditor a;
    a.setComplexString("8+i*4");
    Assert::AreEqual(expected, a.getComplexString());
}

TEST_METHOD(TComplexEditor_setComplexString2)
{
    std::string expected = "-8-i*4";
    TComplexEditor a;
    a.setComplexString("-8-i*4");
    Assert::AreEqual(expected, a.getComplexString());
}

TEST_METHOD(TComplexEditor_setComplexString3)
{
    auto func = [] {TComplexEditor a; a.setComplexString("i*4"); };
    Assert::ExpectException<std::invalid_argument>(func);
}

```

}
};
}