

Министерство цифрового развития, связи и массовых коммуникаций
Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования «Сибирский государственный университет
телекоммуникаций и информатики» (СибГУТИ)

Кафедра прикладной математики и кибернетики

Лабораторная работа
«Редактор комплексных чисел»

Выполнил:
Студент группы ИП-017
Костин А.В.
Работу проверил:
ассистент кафедры ПМиК
Агалаков А.А.

Новосибирск 2021 г.

Содержание

1. Задание	3
2. Исходный код программы.....	4
2.1. Код программы	4
2.2. Код тестов.....	11
3. Результаты модульных тестов	15
4. Вывод	15

1. 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. На Унифицированном языке моделирования UML (Unified Modeling Language) наш класс можно обозначить следующим образом:

РедакторКомплексныхЧисел

- строка: String
- комплексноеЧислоЕстьНоль: Boolean
- добавитьЗнак: String
- добавитьЦифру (a: Integer): String
- добавитьНоль: String
- забойСимвола: String
- очистить: String
- конструктор
- читатьСтрокаВформатеСтроки: String (метод свойства)
- писатьСтрокаВформатеСтроки (a: String) (метод свойства)
- редактировать (a: Integer): String

Обязанность:

Ввод, хранение и редактирование строкового представления комплексных чисел

4. Класс должен отвечать за посимвольный ввод, хранение и редактирование строкового представления комплексных чисел. Значение комплексного нуля - '0, i* 0, '. Класс должен обеспечивать:

- добавление цифры;
- добавление и изменение знака действительной и мнимой частей;
- добавление разделителя целой и дробной частей действительной и мнимой частей комплексного числа;
- добавление разделителя мнимой и действительной частей комплексного числа
- забой символа, стоящего справа (BackSpace);
- установку нулевого значения комплексного числа (Clear);
- чтение строкового представления комплексного числа;
- запись строкового представления комплексного числа.

2. Исходный код программы

2.1. Код программы

TEditor.p

```
#pragma once
#include <string>

enum complexPart {
    REAL,
    IMAG
};

class TEditor
{
    std::string real;
    std::string imag;
    std::string complex;
    complexPart editingPart;
    bool hasRealDel = false;
    bool hasImagDel = false;

public:
    TEditor();
    bool isZero();
    void flipSign();
    void addDigit(int digit);
    void addZero();
    void deleteDigit();
    void clear();
    void processCommand(int command);
    std::string toString();
    void addDelimiter();
    void setComplexNumber(std::string num);
    void setEditingPart(complexPart part);
};
```

TEditor.cpp

```
#include "TEditor.h"
#include <iostream>
#include <vector>
#include <regex>
```

```

TEditor::TEditor()
{
    clear();
    editingPart = REAL;
}

bool TEditor::isZero()
{
    return real == "0" && imag == "0";
}

void TEditor::flipSign()
{
    switch (editingPart)
    {
    case REAL:
        if (real[0] == '-') {
            real.erase(0, 1);
        }
        else {
            real.insert(0, 1, '-');
        }
        break;
    case IMAG:
        if (imag[0] == '-') {
            imag.erase(0, 1);
        }
        else {
            imag.insert(0, 1, '-');
        }
        break;
    default:
        break;
    }
}

void TEditor::addDigit(int digit)
{
    if (digit > 0 && digit < 10) {
        char d = digit + '0';
        switch (editingPart)
        {
        case REAL:

```

```

        real += d;
        break;
    case IMAG:
        imag += d;
        break;
    default:
        break;
    }
}

void TEditor::addZero()
{
    switch (editingPart)
    {
    case REAL:
        real += '0';
        break;
    case IMAG:
        imag += '0';
        break;
    default:
        break;
    }
}

void TEditor::deleteDigit()
{
    switch (editingPart)
    {
    case REAL:
        if (real.length() > 0) {
            if (real[real.length() - 1] == '.') {
                hasRealDel = false;
            }
            real.pop_back();
        }
        break;
    case IMAG:
        if (imag.length() > 0) {
            if (imag[imag.length() - 1] == '.') {
                hasImagDel = false;
            }
            imag.pop_back();
        }
    }
}

```

```

        }
        break;
default:
    break;
    }
}

```

```

void TEditor::clear()
{
    real = "0";
    imag = "0";
    complex = "0 + 0i";
}

```

```

void TEditor::processCommand(int command)
{
    switch (command)
    {
    case 1:
        // add digit
        int digit;
        std::cout << "Enter digit: ";
        std::cin >> digit;
        if (digit == 0) {
            addZero();
        }
        else {
            addDigit(digit);
        }
        break;
    case 2:
        flipSign();
        break;
    case 3:
        addDelimiter();
        break;
    case 4:
        deleteDigit();
        break;
    case 5:
        clear();
        break;
    case 6:
        std::cout << "Complex nuber = " << toString() << std::endl;
    }
}

```

```

        system("pause");
        break;
case 7:
{
    std::string num;
    std::cout << "Enter complex number: ";
    std::cin >> num;
    setComplexNumber(num);
    break;
}
case 8:
    int input;
    std::cout << "Select editing part: 1 - real, 2 - imaginary: ";
    std::cin >> input;
    switch (input)
    {
        case 1:
            setEditingPart(REAL);
            break;
        case 2:
            setEditingPart(IMAG);
            break;
        default:
            std::cout << "Invalid input" << std::endl;
            return;
    }
default:
    std::cout << "Invalid command: " << command << std::endl;
    break;
}
}

std::string TEditor::toString()
{
    complex = real;
    complex += " ";
    if (imag[0] != '-') {
        complex += "+ ";
    }
    complex += imag;
    complex += "i";
    return complex;
}

```



```

void TEditor::addDelimiter()
{
    switch (editingPart)
    {
        case REAL:
            if (!hasRealDel) {
                real += ".";
                hasRealDel = true;
            }
            break;
        case IMAG:
            if (!hasImagDel) {
                imag += ".";
                hasImagDel = true;
            }
            break;
        default:
            break;
    }
}

std::vector<std::string> split(std::string str, std::string delim) {
    size_t pos_start = 0, pos_end, delim_len = delim.length();
    std::string token;
    std::vector<std::string> res;

    while ((pos_end = str.find(delim, pos_start)) != std::string::npos) {
        token = str.substr(pos_start, pos_end - pos_start);
        pos_start = pos_end + delim_len;
        res.push_back(token);
    }

    res.push_back(str.substr(pos_start));
    return res;
}

void TEditor::setComplexNumber(std::string num)
{
    std::string no_spaces = "";
    for (char c : num) {
        if (c != ' ') {
            no_spaces += c;
        }
    }
}

```

```

        std::regex r(R"(\-?\d+(\.\d+)?(\+|\-)\d+(\.\d+)?i)"); // real.realFrac (+ or -)
imag.imagFrac i (fractional parts are optional)
        if (!std::regex_match(no_spaces.data(), r)) {
            std::cout << "Invalid format" << std::endl;
            return;
        }
        editingPart = REAL;
        real = "";
        imag = "";
        int i = 0;
        if (no_spaces[i] == '-') {
            real += '-';
            i++;
        }
        for (; i < no_spaces.length(); i++) {
            if (no_spaces[i] == '+' || no_spaces[i] == '-') {
                editingPart = IMAG;
                if (no_spaces[i] == '-') {
                    flipSign();
                }
                continue;
            }
            if (no_spaces[i] == '.') {
                addDelimiter();
                continue;
            }
            int digit = no_spaces[i] - '0';
            if (digit == 0) {
                addZero();
            }
            else {
                addDigit(digit);
            }
        }
        editingPart = REAL;
    }

void TEditor::setEditingPart(complexPart part)
{
    editingPart = part;
}

```

2.2. Код тестов

UnitTests.cs

```
#include "pch.h"
#include "CppUnitTest.h"
#include "../ModernCodingComplex/TEditor.h"

using namespace Microsoft::VisualStudio::CppUnitTestFramework;

namespace UnitTest
{
    TEST_CLASS(UnitTest)
    {
    public:

        TEST_METHOD(TestClear)
        {
            TEditor editor;
            editor.clear();
            std::string result = editor.toString();
            std::string expected = "0 + 0i";
            Assert::AreEqual(result, expected);
        }
        TEST_METHOD(TestFlipSign1)
        {
            TEditor editor;
            editor.clear();
            editor.flipSign();
            std::string result = editor.toString();
            std::string expected = "-0 + 0i";
            Assert::AreEqual(result, expected);
        }
        TEST_METHOD(TestFlipSign2)
        {
            TEditor editor;
            editor.clear();
            editor.setEditingPart(IMAG);
            editor.flipSign();
            std::string result = editor.toString();
            std::string expected = "0 -0i";
            Assert::AreEqual(result, expected);
        }
        TEST_METHOD(TestAddDigit1)
        {
```

```

    TEditor editor;
    editor.clear();
    editor.addDigit(156);
    std::string result = editor.toString();
    std::string expected = "0 + 0i";
    Assert::AreEqual(result, expected);
}
TEST_METHOD(TestAddDigit2)
{
    TEditor editor;
    editor.clear();
    editor.addDigit(1);
    std::string result = editor.toString();
    std::string expected = "01 + 0i";
    Assert::AreEqual(result, expected);
}
TEST_METHOD(TestAddDigit3)
{
    TEditor editor;
    editor.clear();
    editor.setEditingPart(IMAG);
    editor.addDigit(1);
    std::string result = editor.toString();
    std::string expected = "0 + 01i";
    Assert::AreEqual(result, expected);
}
TEST_METHOD(TestAddZero1)
{
    TEditor editor;
    editor.clear();
    editor.addZero();
    std::string result = editor.toString();
    std::string expected = "00 + 0i";
    Assert::AreEqual(result, expected);
}
TEST_METHOD(TestAddZero2)
{
    TEditor editor;
    editor.clear();
    editor.setEditingPart(IMAG);
    editor.addZero();
    std::string result = editor.toString();
    std::string expected = "0 + 00i";
    Assert::AreEqual(result, expected);
}

```

```

}
TEST_METHOD(TestDeleteDigit1)
{
    TEditor editor;
    editor.clear();
    editor.setComplexNumber("15 + 3i");
    editor.deleteDigit();
    std::string result = editor.toString();
    std::string expected = "1 + 3i";
    Assert::AreEqual(expected, result);
}
TEST_METHOD(TestDeleteDigit2)
{
    TEditor editor;
    editor.clear();
    editor.setComplexNumber("1 + 31i");
    editor.setEditingPart(IMAG);
    editor.deleteDigit();
    std::string result = editor.toString();
    std::string expected = "1 + 3i";
    Assert::AreEqual(result, expected);
}
TEST_METHOD(TestAddDelimiter1)
{
    TEditor editor;
    editor.clear();
    editor.setComplexNumber("1 + 3i");
    editor.addDelimiter();
    std::string result = editor.toString();
    std::string expected = "1. + 3i";
    Assert::AreEqual(result, expected);
}
TEST_METHOD(TestAddDelimiter2)
{
    TEditor editor;
    editor.clear();
    editor.setComplexNumber("1.0 + 3i");
    editor.addDelimiter();
    std::string result = editor.toString();
    std::string expected = "1.0 + 3i";
    Assert::AreEqual(result, expected);
}
TEST_METHOD(TestAddDelimiter3)
{

```

```

        TEditor editor;
        editor.clear();
        editor.setComplexNumber("1 + 3i");
        editor.setEditingPart(IMAG);
        editor.addDelimiter();
        std::string result = editor.toString();
        std::string expected = "1 + 3.i";
        Assert::AreEqual(result, expected);
    }
    TEST_METHOD(TestAddDelimiter4)
    {
        TEditor editor;
        editor.clear();
        editor.setComplexNumber("1 + 3.0i");
        editor.setEditingPart(IMAG);
        editor.addDelimiter();
        std::string result = editor.toString();
        std::string expected = "1 + 3.0i";
        Assert::AreEqual(result, expected);
    }
};
}

```

3. Результаты модульных тестов

Тестирование	Длительность
▲ ✓ UnitTest (14)	5 мс
▲ ✓ UnitTest (14)	5 мс
▲ ✓ UnitTest (14)	5 мс
✓ TestAddDelimiter1	5 мс
✓ TestAddDelimiter2	< 1 мс
✓ TestAddDelimiter3	< 1 мс
✓ TestAddDelimiter4	< 1 мс
✓ TestAddDigit1	< 1 мс
✓ TestAddDigit2	< 1 мс
✓ TestAddDigit3	< 1 мс
✓ TestAddZero1	< 1 мс
✓ TestAddZero2	< 1 мс
✓ TestClear	< 1 мс
✓ TestDeleteDigit1	< 1 мс
✓ TestDeleteDigit2	< 1 мс
✓ TestFlipSign1	< 1 мс
✓ TestFlipSign2	< 1 мс

4. Вывод

По итогам данной лабораторной работе были сформированы практические навыки реализации абстрактных типов данных в соответствии с заданной спецификацией с помощью классов C++ и их модульного тестирования.