

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-113
Шпилев Д. И.
Работу проверил:
старший преподаватель кафедры ПМиК
Агалаков А.А.

Новосибирск 2024 г.

Содержание

1. Задание.....	3
2. Исходный код программы	4
2.1. Код программы	4
2.2 Код тестов.....	9
3. Результаты модульных тестов	11
4. Вывод	11

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

UAEditor.h

```
#pragma once
#include <string>
#include "UANumber.h"

class AEditor
{
public:
    enum Command {
        cZero,
        cOne,
        cTwo,
        cThree,
        cFour,
        cFive,
        cSize,
        cSeven,
        cEight,
        cNine,
        cA,
        cB,
        cC,
        cD,
        cE,
        cF,
        cSign,
        cSeparatorFR,
        cSeparatorC,
        cBS,
        cCE
    };

    virtual ~AEditor() = 0;

    std::string number() const noexcept { return m_number; }
    virtual void setNumber(const std::string& number) = 0;

    virtual bool isNull() const noexcept = 0;
    virtual std::string toggleMinus() noexcept = 0;
    virtual std::string addNumber(int number) = 0;
    virtual std::string addSeparator() = 0;
    virtual std::string addZero() = 0;
    virtual std::string BackSpace() = 0;
    virtual std::string CE() = 0;
    virtual std::string Edit(Command command) = 0;

protected:
    std::string m_number;
    std::string m_separator;
};

class CEditor : public AEditor {
public:
    enum EditMode
    {
        Actual,
        Imaginary
    };

    CEditor();
```

```

    CEditor(double actual, double imaginary);
    CEditor(const std::string& number);
    virtual ~CEditor() = default;

    void setNumber(const std::string& number) override;
    bool isNull() const noexcept override;
    std::string toggleMinus() noexcept override;
    std::string addNumber(int number) override;
    std::string addSeparator() override;
    std::string addZero() override;
    std::string BackSpace() override;
    std::string CE() override;
    std::string Edit(Command command) override;

    std::string addNumberSeparator();
    EditMode changeEditMode();

private:
    EditMode editMode;
};

```

UAEEditor.cpp

```

#include "UAEEditor.h"

AEditor::~AEditor() = default;

CEditor::CEditor()
{
    m_number = TComplex().numberString();
    m_separator = " + i * ";
    editMode = EditMode::Actual;
}

CEditor::CEditor(double actual, double imaginary)
{
    m_number = TComplex(actual, imaginary).numberString();
    m_separator = imaginary >= 0 ? " + i * " : " - i * ";
    editMode = EditMode::Actual;
}

CEditor::CEditor(const std::string& number)
{
    TComplex num(number);
    m_number = num.numberString();
    m_separator = num.getImaginary() >= 0 ? " + i * " : " - i * ";
    editMode = EditMode::Actual;
}

void CEditor::setNumber(const std::string& number)
{
    m_number = TComplex(number).numberString();
}

bool CEditor::isNull() const noexcept
{
    return TComplex(m_number).isNull();
}

std::string CEditor::toggleMinus() noexcept
{
    if (editMode == EditMode::Actual)
    {
        if (m_number[0] == '-')

```

```

        {
            m_number = m_number.substr(1);
        }
        else
        {
            m_number = "-" + m_number;
        }
    }
    else
    {
        size_t position = m_number.find(m_separator);
        if (m_separator[1] == '+') m_separator[1] = '-';
        else m_separator[1] = '+';
        m_number = m_number.substr(0, position) + m_separator +
m_number.substr(position + m_separator.length());
    }
    return m_number;
}

std::string CEditor::addNumber(int number)
{
    size_t position = m_number.find(m_separator);
    std::string left = m_number.substr(0, position);
    std::string right = m_number.substr(position + m_separator.length());
    if (editMode == EditMode::Actual)
    {
        if (left == "0")
            m_number = std::to_string(number) + m_separator + right;
        else
            m_number = left + std::to_string(number) + m_separator + right;
    }
    else
    {
        if (right == "0")
            m_number.pop_back();
        m_number += std::to_string(number);
    }
    return m_number;
}

std::string CEditor::addSeparator()
{
    size_t position = m_number.find(m_separator);
    if (position == std::string::npos) {
        m_number += m_separator + "0";
    }
    return m_number;
}

std::string CEditor::addNumberSeparator()
{
    size_t position = m_number.find(m_separator);
    std::string left = m_number.substr(0, position);
    std::string right = m_number.substr(position + m_separator.length());
    if (editMode == EditMode::Actual)
    {
        if (left.find(".") == std::string::npos)
            m_number = left + "." + m_separator + right;
    }
    else
    {
        if (right.find(".") == std::string::npos)
            m_number += ".";
    }
    return m_number;
}

```

```

}

CEditor::EditMode CEditor::changeEditMode()
{
    if (editMode == EditMode::Actual)
        editMode = EditMode::Imaginary;
    else
        editMode = EditMode::Actual;
    return editMode;
}

std::string CEditor::addZero()
{
    size_t position = m_number.find(m_separator);
    std::string left = m_number.substr(0, position);
    std::string right = m_number.substr(position + m_separator.length());
    if (editMode == EditMode::Actual)
    {
        if (left.find(".") != std::string::npos || left.length() == 0)
            m_number = left + "0" + m_separator + right;
    }
    else
    {
        if (right.find(".") != std::string::npos || right.length() == 0)
            m_number += "0";
    }
    return m_number;
}

std::string CEditor::BackSpace()
{
    size_t position = m_number.find(m_separator);
    std::string left = m_number.substr(0, position);
    std::string right = m_number.substr(position + m_separator.length());
    if (editMode == EditMode::Actual)
    {
        if (left != "0")
        {
            left.pop_back();
            if (left.length() == 0 || left == "-")
                left = "0";
        }
    }
    else
    {
        if (right != "0")
        {
            right.pop_back();
            if (right.length() == 0 || right == "-")
                right = "0";
        }
    }
    m_number = left + m_separator + right;
    return m_number;
}

std::string CEditor::CE()
{
    *this = CEditor();
    return m_number;
}

std::string CEditor::Edit(Command command)
{
    switch (command)

```

```

{
case AEditor::cZero:
    addZero();
    break;
case AEditor::cOne:
    addNumber(1);
    break;
case AEditor::cTwo:
    addNumber(2);
    break;
case AEditor::cThree:
    addNumber(3);
    break;
case AEditor::cFour:
    addNumber(4);
    break;
case AEditor::cFive:
    addNumber(5);
    break;
case AEditor::cSize:
    addNumber(6);
    break;
case AEditor::cSeven:
    addNumber(7);
    break;
case AEditor::cEight:
    addNumber(8);
    break;
case AEditor::cNine:
    addNumber(9);
    break;
case AEditor::cSign:
    toggleMinus();
    break;
case AEditor::cSeparatorFR:
    addNumberSeparator();
    break;
case AEditor::cSeparatorC:
    addSeparator();
    break;
case AEditor::cBS:
    BackSpace();
    break;
case AEditor::cCE:
    CE();
    break;
default:
    break;
}
return m_number;
}

```


2.2 Код тестов

```
TEST_CLASS(CEditorTests)
{
public:

    TEST_METHOD(TestConstructorDefault)
    {
        CEditor editor;
        Assert::AreEqual(std::string("0 + i * 0"), editor.number());
    }

    TEST_METHOD(TestConstructorWithDoublesPositiveImaginary)
    {
        CEditor editor(5.0, 3.0);
        Assert::AreEqual(std::string("5 + i * 3"), editor.number());
    }

    TEST_METHOD(TestConstructorWithDoublesNegativeImaginary)
    {
        CEditor editor(5.0, -3.0);
        Assert::AreEqual(std::string("5 - i * 3"), editor.number());
    }

    TEST_METHOD(TestConstructorWithStringPositiveImaginary)
    {
        CEditor editor("5 + i * 3");
        Assert::AreEqual(std::string("5 + i * 3"), editor.number());
    }

    TEST_METHOD(TestConstructorWithStringNegativeImaginary)
    {
        CEditor editor("5 - i * 3");
        Assert::AreEqual(std::string("5 - i * 3"), editor.number());
    }

    TEST_METHOD(TestSetNumber)
    {
        CEditor editor;
        editor.setNumber("3 + i * 4");
        Assert::AreEqual(std::string("3 + i * 4"), editor.number());
    }

    TEST_METHOD(TestToggleMinusPositive)
    {
        CEditor editor(5.0, 3.0);
        Assert::AreEqual(std::string("-5 + i * 3"), editor.toggleMinus());
    }

    TEST_METHOD(TestToggleMinusNegative)
    {
        CEditor editor(-5.0, 3.0);
        Assert::AreEqual(std::string("5 + i * 3"), editor.toggleMinus());
    }

    TEST_METHOD(TestAddNumber)
    {
        CEditor editor(1.0, 2.0);
        Assert::AreEqual(std::string("16 + i * 2"), editor.addNumber(6));
    }

    TEST_METHOD(TestAddZero)
    {
        CEditor editor;
        Assert::AreEqual(std::string("0 + i * 0"), editor.addZero());
    }
}
```

```

}

TEST_METHOD(TestBackSpace)
{
    CEditor editor("12 + i * 34");
    Assert::AreEqual(std::string("1 + i * 34"), editor.BackSpace());
}

TEST_METHOD(TestCE)
{
    CEditor editor("12 + i * 34");
    Assert::AreEqual(std::string("0 + i * 0"), editor.CE());
}

TEST_METHOD(TestChangeEditMode)
{
    CEditor editor;
    Assert::AreEqual(CEditor::EditMode::Imaginary, editor.changeEditMode());
    Assert::AreEqual(CEditor::EditMode::Actual, editor.changeEditMode());
}

TEST_METHOD(TestBackSpaceRight)
{
    CEditor editor("12 + i * 3");
    editor.changeEditMode();
    Assert::AreEqual(std::string("12 + i * 0"), editor.BackSpace());
}

TEST_METHOD(TestAddSeparator_SeparatorAlreadyPresent)
{
    CEditor editor("5 + i * 3");
    Assert::AreEqual(std::string("5 + i * 3"), editor.addSeparator());
}

TEST_METHOD(TestAddNumberSeparator_ActualWithoutDot)
{
    CEditor editor("5 + i * 3");
    Assert::AreEqual(std::string("5. + i * 3"), editor.addNumberSeparator());
}

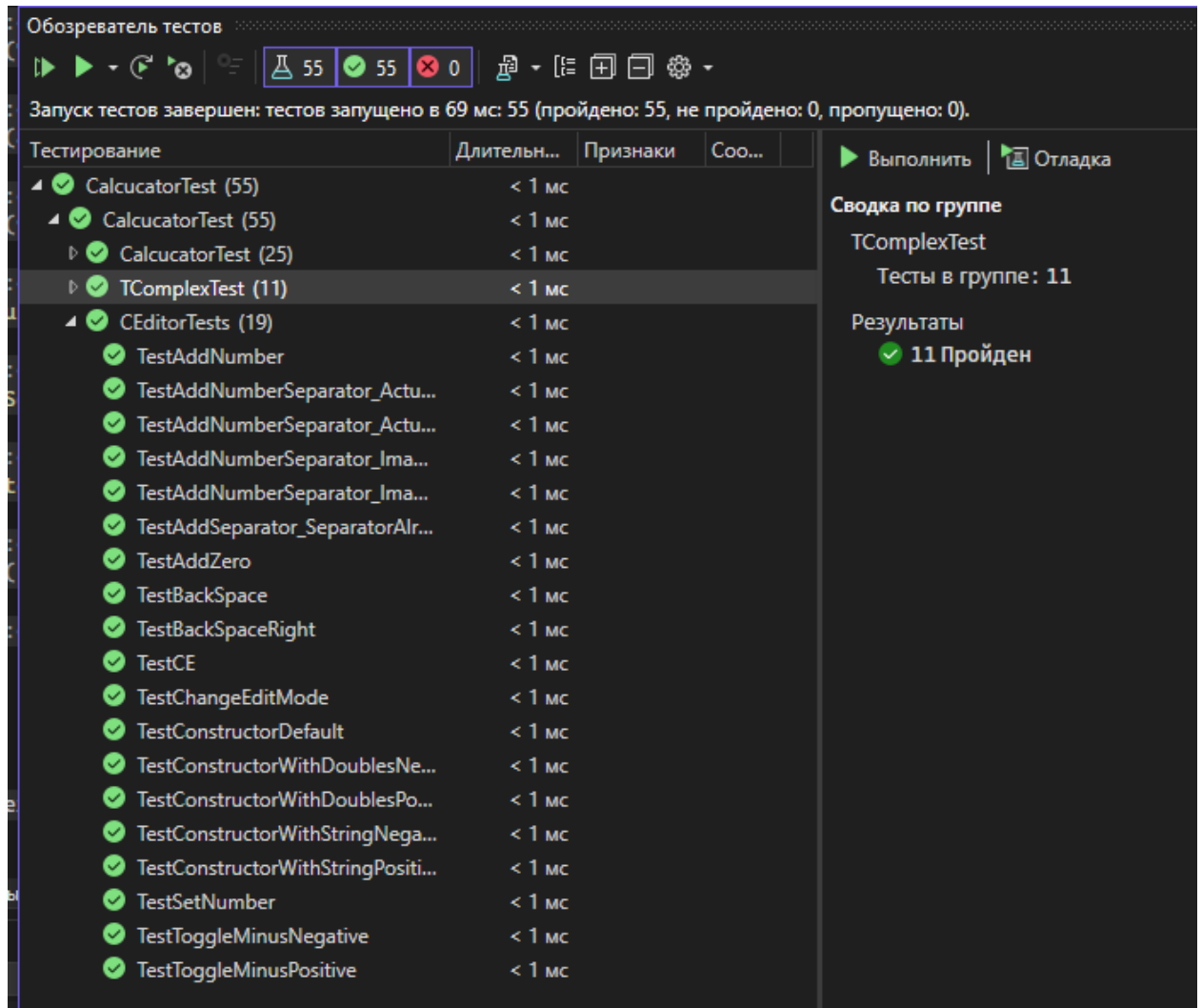
TEST_METHOD(TestAddNumberSeparator_ActualWithDot)
{
    CEditor editor("5. + i * 3");
    Assert::AreEqual(std::string("5. + i * 3"), editor.addNumberSeparator());
}

TEST_METHOD(TestAddNumberSeparator_ImaginaryWithoutDot)
{
    CEditor editor("5 + i * 3");
    editor.changeEditMode();
    Assert::AreEqual(std::string("5 + i * 3."), editor.addNumberSeparator());
}

TEST_METHOD(TestAddNumberSeparator_ImaginaryWithDot)
{
    CEditor editor("5 + i * 3.");
    editor.changeEditMode();
    Assert::AreEqual(std::string("5 + i * 3."), editor.addNumberSeparator());
}
};

```

3. Результаты модульных тестов



Обозреватель тестов

Запуск тестов завершен: тестов запущено в 69 мс: 55 (пройдено: 55, не пройдено: 0, пропущено: 0).

Тестирование	Длительн...	Признаки	Сoo...
▲ ✓ CalcucatorTest (55)	< 1 мс		
▲ ✓ CalcucatorTest (55)	< 1 мс		
▶ ✓ CalcucatorTest (25)	< 1 мс		
▶ ✓ TComplexTest (11)	< 1 мс		
▲ ✓ CEditorTests (19)	< 1 мс		
✓ TestAddNumber	< 1 мс		
✓ TestAddNumberSeparator_Actu...	< 1 мс		
✓ TestAddNumberSeparator_Actu...	< 1 мс		
✓ TestAddNumberSeparator_Ima...	< 1 мс		
✓ TestAddNumberSeparator_Ima...	< 1 мс		
✓ TestAddSeparator_SeparatorAlr...	< 1 мс		
✓ TestAddZero	< 1 мс		
✓ TestBackSpace	< 1 мс		
✓ TestBackSpaceRight	< 1 мс		
✓ TestCE	< 1 мс		
✓ TestChangeEditMode	< 1 мс		
✓ TestConstructorDefault	< 1 мс		
✓ TestConstructorWithDoublesNe...	< 1 мс		
✓ TestConstructorWithDoublesPo...	< 1 мс		
✓ TestConstructorWithStringNega...	< 1 мс		
✓ TestConstructorWithStringPositi...	< 1 мс		
✓ TestSetNumber	< 1 мс		
✓ TestToggleMinusNegative	< 1 мс		
✓ TestToggleMinusPositive	< 1 мс		

Выполнить | Отладка

Сводка по группе

TComplexTest

Тесты в группе: 11

Результаты

✓ 11 Пройден

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.