

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

## **Лабораторная работа №5**

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 TEditor «Ввод и редактирование простых дробей», 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

Абстрактный класс TFracEditor был реализован посредством создания шаблона класса. Шаблон класса имеет единственное поле со строковым представлением дроби. Так же были реализованы процедуры для взаимодействия с этой строкой.


- TFracEditor() – базовый конструктор класса, формирует нулевую дробь «0/1»;
- void simplify() – вспомогательная функция, необходимая для сокращения дроби в случае необходимости;
- bool isZero() – булевый метод, необходимый для проверки на то, является ли дробь нулевой;
- std::string addSign() – метод, необходимый для добавления или убирания знака минус у дроби;
- std::string addNumber(int num) – метод, необходимый для добавления числа в конец дроби;
- std::string addZero() – метод, добавляющий нуль к концу дроби;
- std::string removeLastDigit() – метод, удаляющий последний стоящий справа символ у дроби;
- std::string clear() – метод очистки строки дроби;
- std::string addDivider() – метод по добавлению символа дроби «/» к дробной строке;
- void editFraction(Operations operation) – метод, который принимает в себя необходимую для выполнения операцию среди выше перечисленных;
- void setFraction(std::string frac) – метод который устанавливает значение дроби, есть проверка на корректность;

- `std::string getFraction()` – метод получения строкового представления дроби;
- `bool isValid(std::string)` – вспомогательный метод, служащий для определения корректности введенной дроби;

## Заключение

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

## Скриншоты



```
TFracEditor:  
init: 0/1  
Enter number: 80  
Add Number: 80  
Add divider: 80/  
Add sign: -80/  
Enter number: 40  
Add Number: -2/1  
Remove digit: -2/
```

Рис. 1 – пример работы программы



```

        ADD_DIGIT,
        ADD_ZERO,
        REMOVE_LAST_DIGIT,
        CLEAR,
        ADD_DIVIDER
    };

class TFracEditor
{
private:
    std::string frac;
public:
    static const char divider;
    static const std::string zeroFrac;
    TFracEditor();
    void simplify();

    bool isZero();
    std::string addSign();
    std::string addNumber(int num);
    std::string addZero();
    std::string removeLastDigit();
    std::string clear();
    std::string addDivider();
    void editFraction(Operations operation);
    void setFraction(std::string frac);
    std::string getFraction();

    bool isValid(std::string);
};

```

## TFracEditor.cpp

```

#include "TFracEditor.h"

const char TFracEditor::divider = '/';
const std::string TFracEditor::zeroFrac = "0/1";

TFracEditor::TFracEditor() : frac(zeroFrac) {}

void TFracEditor::simplify()
{
    unsigned int average = frac.find('/');
    if (average != -1) {
        char buf1[30], buf2[30];
        size_t n1 = frac.copy(buf1, average, 0);
        size_t n2 = frac.copy(buf2, frac.length() - average - 1, average + 1);
        buf1[n1] = '\0';
        buf2[n2] = '\0';
        int num = atoi(buf1);
        int den = atoi(buf2);

        int g = std::gcd(num, den);
        num /= g;
        den /= g;
        if (den < 0) {
            num *= -1;
            den *= -1;
        }
        frac = std::to_string(num) + "/" + std::to_string(den);
    }
}

bool TFracEditor::isZero()

```

```

{
    return frac == zeroFrac;
}

std::string TFracEditor::addSign()
{
    if (frac[0] == '-')
        frac.erase(frac.begin());
    else if (frac != zeroFrac)
        frac = "-" + frac;
    return frac;
}

std::string TFracEditor::addNumber(int num)
{
    if (frac == zeroFrac)
        frac.erase();
    frac += std::to_string(num);
    simplify();
    return frac;
}

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

std::string TFracEditor::removeLastDigit()
{
    frac.pop_back();
    if (frac == "-" || frac.empty())
        frac = zeroFrac;
    return frac;
}

std::string TFracEditor::clear()
{
    return frac = zeroFrac;
}

std::string TFracEditor::addDivider()
{
    if (frac.find(divider) == std::string::npos)
        frac += divider;
    return frac;
}

void TFracEditor::editFraction(Operations operation)
{
    switch (operation)
    {
        case Operations::ADD_SIGN:
            addSign();
            break;
        case Operations::ADD_DIGIT:
            int num;
            std::cout << "Enter number: ";
            std::cin >> num;
            addNumber(num);
            break;
        case Operations::ADD_ZERO:
            addZero();
            break;
        case Operations::REMOVE_LAST_DIGIT:
            removeLastDigit();
    }
}

```

```

        break;
    case Operations::CLEAR:
        clear();
        break;
    case Operations::ADD_DIVIDER:
        addDivider();
        break;
    default:
        break;
}
}

bool TFracEditor::isValid(std::string fraction)
{
    bool isValid = false;
    if (!fraction.empty() && (fraction.find('/') != std::string::npos))
    {
        unsigned int position = fraction.find('/');
        std::string firstPart = fraction;
        std::string secondPart = fraction;
        std::string numerator = firstPart.erase(position, fraction.length());
        std::string denominator = secondPart.erase(0, position);

        int digitsCounter = 0;
        for (char i : numerator)
        {
            if (isdigit(i))
                digitsCounter++;
        }

        for (char i : denominator)
        {
            if (isdigit(i))
                digitsCounter++;
        }

        if (digitsCounter != 0)
        {
            if (fraction[0] == '-')
            {
                if (digitsCounter == fraction.length() - 2)
                    isValid = true;
            }
            else
            {
                if (digitsCounter == fraction.length() - 1)
                    isValid = true;
            }
        }
    }
    return isValid;
}

void TFracEditor::setFraction(std::string fraction)
{
    if (isValid(fraction))
    {
        frac = fraction;
        simplify();
    }
    else
    {
        std::cout << "Wrong format" << std::endl;
        frac = zeroFrac;
    }
}

```

```

}

std::string TFracEditor::getFraction()
{
    return frac;
}

```

## TFracEditor\_Test.cpp

```

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

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

using namespace Microsoft::VisualStudio::CppUnitTestFramework;

namespace MPT_Tests
{
    TEST_CLASS(TFracEditorTest)
    {
    public:
        TEST_METHOD(FracEditor_Constructor)
        {
            std::string expected = "0/1";
            TFracEditor editor;
            Assert::AreEqual(expected, editor.getFraction());
        }

        TEST_METHOD(FracEditor_AddNumber)
        {
            std::string expected = "8";
            TFracEditor editor;
            editor.addNumber(8);
            Assert::AreEqual(expected, editor.getFraction());
        }

        TEST_METHOD(FracEditor_AddNumber_2)
        {
            std::string expected = "-8";
            TFracEditor editor;
            editor.addNumber(-8);
            Assert::AreEqual(expected, editor.getFraction());
        }

        TEST_METHOD(FracEditor_isZero)
        {
            TFracEditor editor;
            Assert::IsTrue(editor.isZero());
        }

        TEST_METHOD(FracEditor_isZero_2)
        {
            TFracEditor editor;
            editor.addNumber(8);
            Assert::IsFalse(editor.isZero());
        }

        TEST_METHOD(FracEditor_addSign)
        {
            std::string expected = "-8";
            TFracEditor editor;
            editor.addNumber(8);
            editor.addSign();
        }
    }
}

```



```

        Assert::AreEqual(expected, editor.getFraction());
    }

TEST_METHOD(FracEditor_addSign_2)
{
    std::string expected = "8";
    TFracEditor editor;
    editor.addNumber(-8);
    editor.addSign();
    Assert::AreEqual(expected, editor.getFraction());
}

TEST_METHOD(FracEditor_addSign_3)
{
    std::string expected = "-1/2";
    TFracEditor editor;
    editor.addNumber(8);
    editor.addDivider();
    editor.addNumber(16);
    editor.addSign();
    Assert::AreEqual(expected, editor.getFraction());
}

TEST_METHOD(FracEditor_addZero)
{
    std::string expected = "0";
    TFracEditor editor;
    editor.addZero();
    Assert::AreEqual(expected, editor.getFraction());
}

TEST_METHOD(FracEditor_addZero_2)
{
    std::string expected = "80";
    TFracEditor editor;
    editor.addNumber(8);
    editor.addZero();
    Assert::AreEqual(expected, editor.getFraction());
}

TEST_METHOD(FracEditor_removeLastDigit)
{
    std::string expected = "8";
    TFracEditor editor;
    editor.addNumber(80);
    editor.removeLastDigit();
    Assert::AreEqual(expected, editor.getFraction());
}

TEST_METHOD(FracEditor_removeLastDigit_2)
{
    std::string expected = "-8";
    TFracEditor editor;
    editor.addNumber(-80);
    editor.removeLastDigit();
    Assert::AreEqual(expected, editor.getFraction());
}

TEST_METHOD(FracEditor_removeLastDigit_3)
{
    std::string expected = "-3/";
    TFracEditor editor;
    editor.addNumber(3);
    editor.addDivider();
    editor.addNumber(-7);

```

```

        editor.removeLastDigit();
        Assert::AreEqual(expected, editor.getFraction());
    }

TEST_METHOD(FracEditor_editFraction)
{
    std::string expected = "0";
    TFracEditor editor;
    editor.editFraction(Operations::ADD_ZERO);
    Assert::AreEqual(expected, editor.getFraction());
}

TEST_METHOD(FracEditor_editFraction_2)
{
    std::string expected = "1/2";
    TFracEditor editor;
    editor.addNumber(8);
    editor.editFraction(Operations::ADD_DIVIDER);
    editor.addNumber(16);
    Assert::AreEqual(expected, editor.getFraction());
}

TEST_METHOD(FracEditor_editFraction_3)
{
    std::string expected = "1/";
    TFracEditor editor;
    editor.addNumber(8);
    editor.editFraction(Operations::ADD_DIVIDER);
    editor.addNumber(16);
    editor.editFraction(Operations::REMOVE_LAST_DIGIT);
    Assert::AreEqual(expected, editor.getFraction());
}

TEST_METHOD(FracEditor_editFraction_4)
{
    std::string expected = "-1/2";
    TFracEditor editor;
    editor.addNumber(8);
    editor.editFraction(Operations::ADD_DIVIDER);
    editor.addNumber(16);
    editor.editFraction(Operations::ADD_SIGN);
    Assert::AreEqual(expected, editor.getFraction());
}

TEST_METHOD(FracEditor_editFraction_5)
{
    std::string expected = "0/1";
    TFracEditor editor;
    editor.addNumber(8);
    editor.editFraction(Operations::ADD_DIVIDER);
    editor.addNumber(16);
    editor.editFraction(Operations::CLEAR);
    Assert::AreEqual(expected, editor.getFraction());
}

TEST_METHOD(FracEditor_getFraction)
{
    std::string expected = "0/1";
    TFracEditor editor;
    Assert::AreEqual(expected, editor.getFraction());
}

TEST_METHOD(FracEditor_getFraction_2)
{
    std::string expected = "8";

```

```

        TFracEditor editor;
        editor.addNumber(8);
        Assert::AreEqual(expected, editor.getFraction());
    }

    TEST_METHOD(FracEditor_setFraction)
    {
        std::string expected = "5/2";
        TFracEditor editor;
        editor.setFraction("5/2");
        Assert::AreEqual(expected, editor.getFraction());
    }

    TEST_METHOD(FracEditor_setFraction_2)
    {
        std::string expected = "-5/2";
        TFracEditor editor;
        editor.setFraction("-5/2");
        Assert::AreEqual(expected, editor.getFraction());
    }

    TEST_METHOD(FracEditor_setFraction_3)
    {
        std::string expected = "-5/2";
        TFracEditor editor;
        editor.setFraction("-10/4");
        Assert::AreEqual(expected, editor.getFraction());
    }

    TEST_METHOD(FracEditor_setFraction_4)
    {
        std::string expected = "0/1";
        TFracEditor editor;
        editor.setFraction("1");
        Assert::AreEqual(expected, editor.getFraction());
    }
};
}

```