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

Институт информатики и вычислительной техники
09.03.01 "Информатика и вычислительная техника"
профиль "Программное обеспечение средств
вычислительной техники и автоматизированных систем"

Кафедра прикладной математики и кибернетики

Расчётно-графическое задание по дисциплине Современные технологии программирования

Вариант-12
«Калькулятор простых дробей»

Выполнил:

Студент гр. ИП-911 _____/Мироненко К.А./
ФИО студента

«__» _____ 2023 г.

Проверил:

Доцент кафедры ПМиК _____/ Зайцев М.Г. /
ФИО преподавателя

«__» _____ 2023 г. Оценка_____

Новосибирск 2022-23 уч. года

Содержание

Цель

Сформировать практические навыки:

- проектирования программ в технологии «абстрактных типов данных» и «объектно-ориентированного программирования» и построения диаграмм UML;
- реализации абстрактных типов данных с помощью классов C#, C++;
- использования библиотеки визуальных компонентов VCL для построения интерфейса,
- тестирования программ.

Задание

Спроектировать и реализовать калькулятор для выполнения вычислений над числами заданными в соответствии с вариантом, используя классы C#, C++ и библиотеку визуальных компонентов для построения интерфейса.

Общие требования

1. Калькулятор обеспечивает вычисление выражений с использованием операций: +, -, *. / и функций: Sqr (возведение в квадрат), Rev (1/x - вычисление обратного значения) без учёта приоритета операций. Приоритет функций одинаковый, выше приоритета операций. Операции имеют равный приоритет.

2. Предусмотреть возможность ввода операндов в выражение:
   - с клавиатуры,
   - с помощью командных кнопок интерфейса,
   - из буфера обмена,
   - из памяти.
3. Необходимо реализовать команду (=). которая завершает вычисление выражения. Она выполняет текущую операцию.
4. Необходимо реализовать команду C (начать вычисление нового выражения), которая устанавливает калькулятор в начальное состояние. Она сбрасывает текущую операцию и устанавливает нулевое значение для отображаемого числа и операндов.

5. Интерфейс выполнить в стиле стандартного калькулятора Windows (вид - обычный).

6. Приложение должно иметь основное окно для ввода исходных данных, операций и отображения результата, и окно для вывода сведений о разработчиках приложения.

7. Основное окно должно содержать список из трёх меню:

    - Правка:
      Содержит два пункта: «Копировать» и «Вставить». Этикоманды используются для работы с буфером обмена;

    - Настройка:
      Содержит команды выбора режима работы приложения;

    - Справка:
      Это команда для вызова справки о приложении.

8. Калькулятор должен обеспечивать возможность ввода исходных данных с помощью:

    - командных кнопок (мышью),

    - клавиатуры: цифровой и алфавитно-цифровой.

9. Вводимые числа выравнивать по правому краю.

10. Калькулятор должен быть снабжён памятью. Для работы с памятью необходимы команды:

    - MC («Очистить»),

    - MS («Сохранить»),

    - MR («Копировать»),

    - M+ («Добавить к содержимому памяти»).

11. Память может находиться в двух состояниях, которые отображаются на панели:

    - «Включена» (M). В памяти храниться занесённое значение

    - «Выключена» ( ). В памяти находится ноль.

    Состояние памяти меняется командами «Сохранить» и «Добавить к содержимому памяти».

12. Для редактирования вводимых значений необходимы команды:
    - BackSpase (удалить крайний справа символ отображаемого числа),
    - CE (заменить отображаемое число нулевым значением)
    - Добавить символ, допустимый в изображении числа (арабские цифры, знак, разделители).
13. Для просмотра выполненных за сеанс вычислений калькулятор необходимо снабдить «Историей».
14. Снабдите компоненты интерфейса всплывающими подсказками.

Требования к варианту

Тип числа – «Калькулятор простых дробей»

1. Калькулятор должен обеспечить ввод и редактирование целых чисел в обычной записи и рациональных дробей в записи:

   [-]<целое без знака>|[-]<числитель><разделитель><знаменатель>.

   <числитель>::=<целое без знака>

   <знаменатель>::=<целое без знака>

   <разделитель>::=’/’ | ‘|’

2. Предусмотреть настройку калькулятора на отображение результата в двух форматах: «дробь» или «число». В формате «дробь» результат всегда отображается в виде дроби. В формате «число» результат отображается в виде числа, если дробь может быть сокращена, так что знаменатель равен 1.

Необходимо предусмотреть следующие варианты использования (прецеденты) калькулятора:

1. Выполнение одиночных операций:

«операнд1» «операция» «операнд2» «=» «результат»

Пример. 5/1 + 2/1 = 7/1.

2. Выполнение операций с одним операндом:

«операнд» «операция» «=» «результат»

Пример. 5/1 * = 25/1.

3. Повторное выполнение операции:

«=» «результат» «=» «результат»

Пример. 5/1 + 4/1 = 9/1 = 13/1 = 17.

4. Выполнение операции над отображаемым значением в качестве обоих операндов:

«результат» «операция» «=» «результат»

Пример. 2/1 + 3/1 = 5/1 = 8/1 + = 16/1.

5. Вычисление функций:

«операнд» «Sqr» «результат»

Пример. 5/1 «Sqr» 25/1.

6. Вычисление выражений:

«операнд1» «функция1» «операция1» «операнд2» «функция2» «операция2» … «операндN» «операцияN» «=» «результат»

Ввод	6/1	Sqr	+	2/1	Sqr	/	10/1	+	6/1	=
Отображаемый результат	6/1	36/1	36/1	2/1	4/1	40/1	10/1	4/1	6/1	10/1

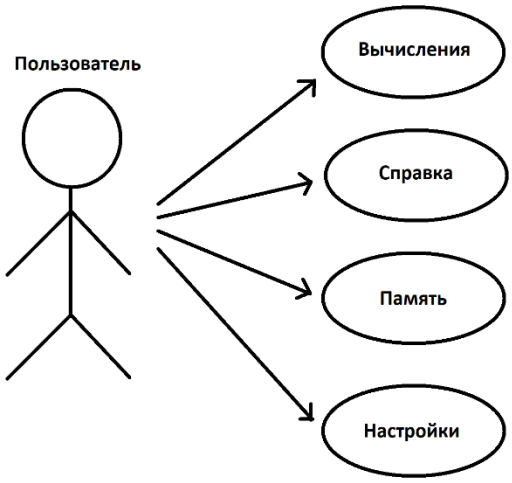
Отображаемое значение может сохраняться в памяти или добавляться к её содержимому.

Варианты выполнения

№ Варианта	**Тип числа**	**Прецеденты**	**Операнды могут браться из**		**История**	**Настройки**
			памяти	**буфера обмена**		
12	простая дробь	1-6	да	да	нет	да

Методические указания к выполнению
Диаграмма прецедентов UML. Сценарии прецедентов

Диаграмма прецедентов UML:



Сценарии прецедентов:

Сценарий для прецедента «Вычисления»:

1. Пользователь вводит дробь (операнд) с символом-разделителем. Если символ-разделитель не введен, то дробь будет преобразована в вид n/1;
2. Пользователь выбирает операцию (оператор);
3. Пользователь может ввести второй операнд;
4. Пользователь нажимает на «=»;
5. Система выполняет действия, заданные пользователем.

Сценарий для прецедента «Справка»:

1. Пользователь выбирает пункт в меню с названием «Справка»;
2. Открывается окно со справочной информацией;
3. Пользователь может прочитать справку или закрыть окно.

Сценарий для прецедента «Память»:

1. Пользователь вводит операнд;
2. Пользователь нажимает на кнопку «MS», тем самым сохраняя введенное число в память;

3. Пользователь может стереть все данные с поля и ввести число, сохраненное в памяти через кнопку «MR»;
4. Пользователь производит вычисления с данным операндом;
5. Пользователь может очистить память кнопкой «MC».

Альтернативный вариант событий:

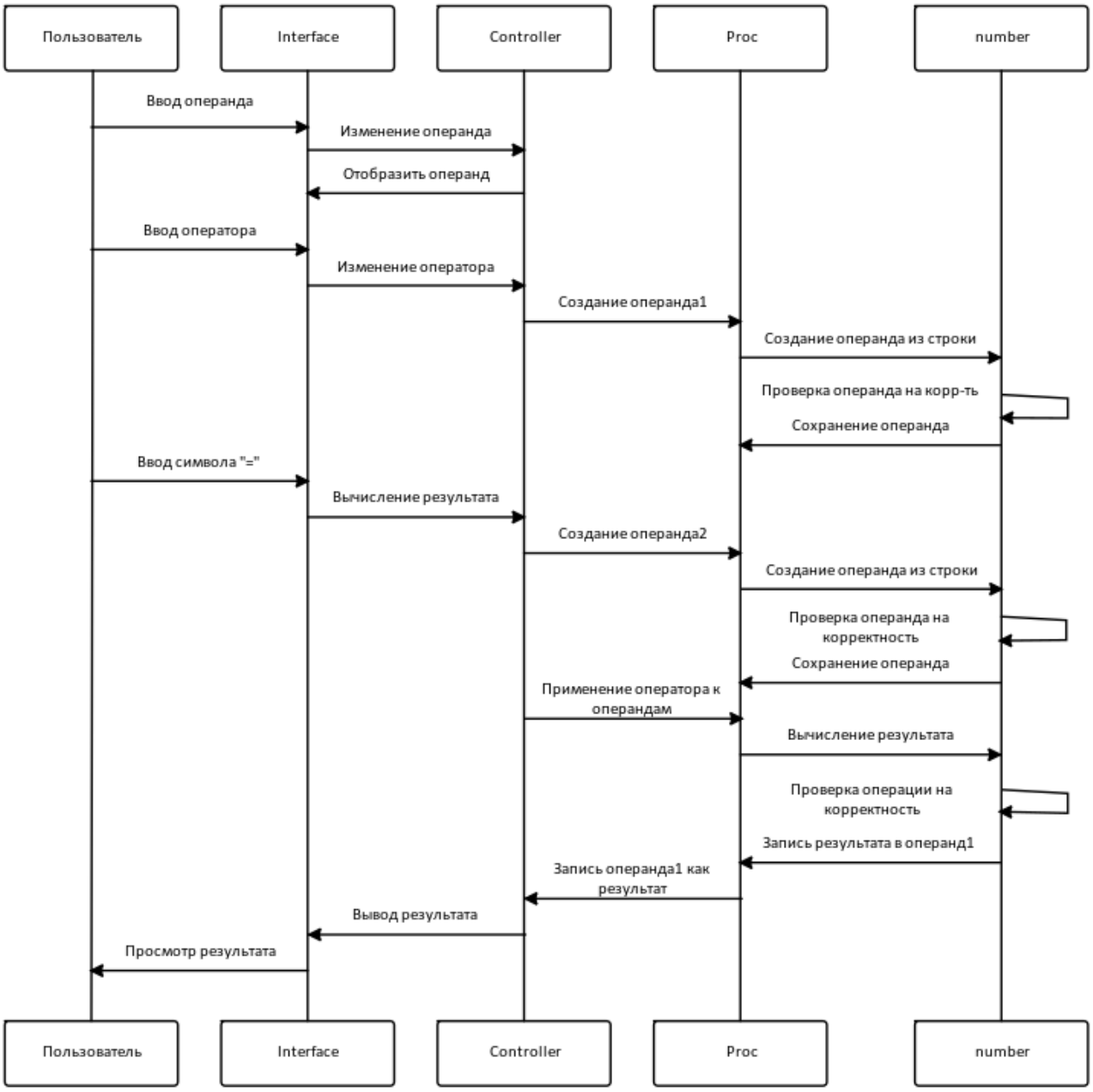
1. Пользователь совершает необходимые вычисления.
2. Получившийся результат сохраняет в память с помощью кнопки «MS»;
3. Пользователь очищает поле вычислений;
4. Пользователь совершает еще одно вычисление;
5. Пользователь вводит необходимую операцию;
6. Пользователь вводит сохраненный операнд их памяти через кнопку «MR»;
7. Пользователь нажимает «=» и получает результат;
8. Пользователь может очистить память кнопкой «MC».

Сценарий для прецедента «Настройки»:

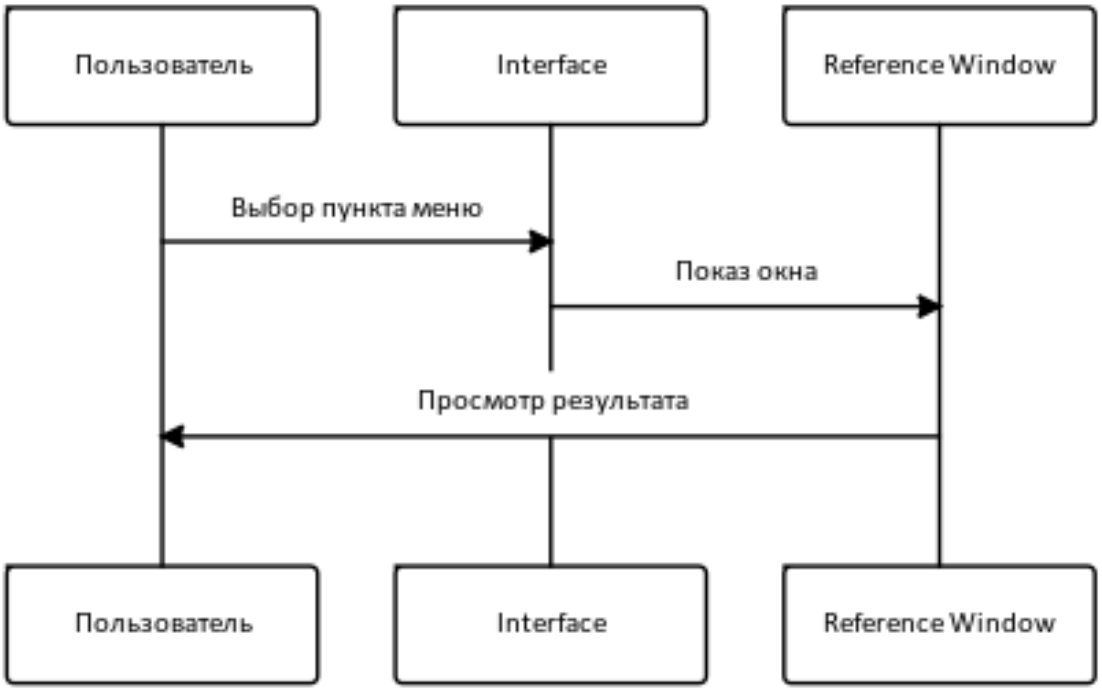
1. Пользователь выбирает пункт меню «Настройки»;
2. Пользователь выбирает один из двух режимов: «Дробь», «Число»;
3. Пользователь совершает необходимые ему вычисления.

Диаграмма последовательностей для прецедентов

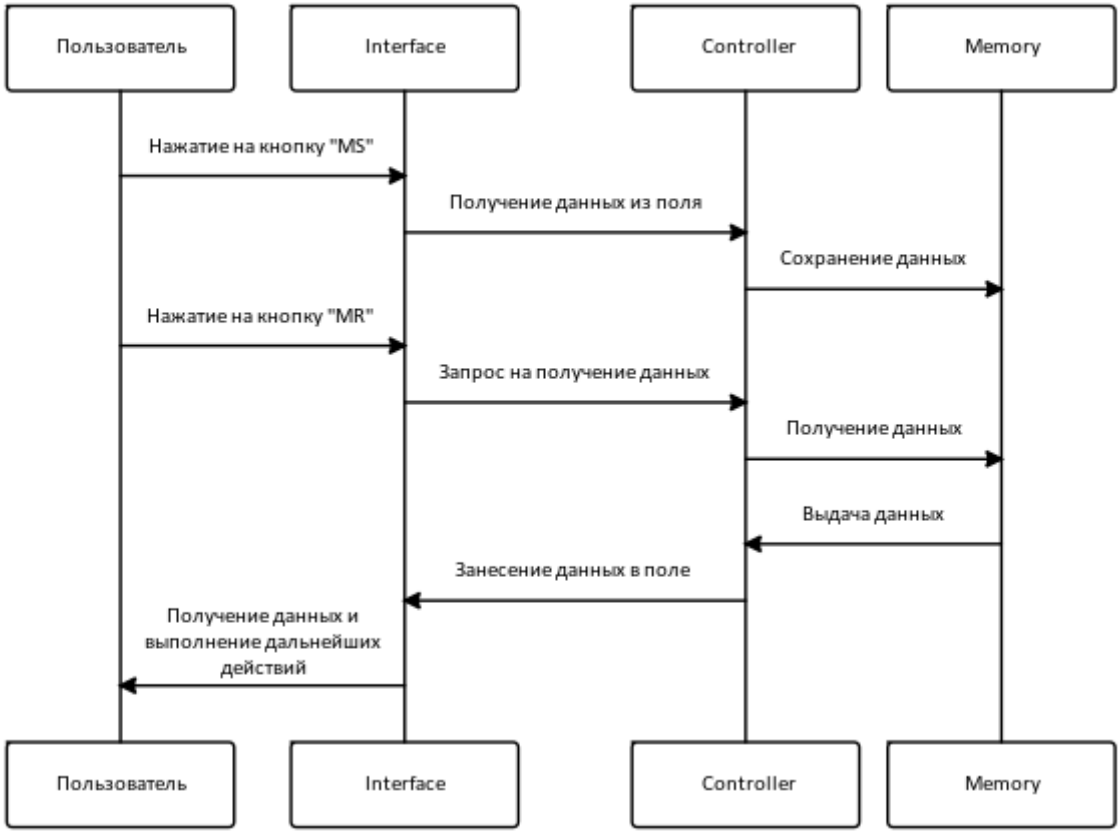
Вычисление выражений и подсчет результатов:



Просмотр справки:



Работа с памятью:



Использование настроек:

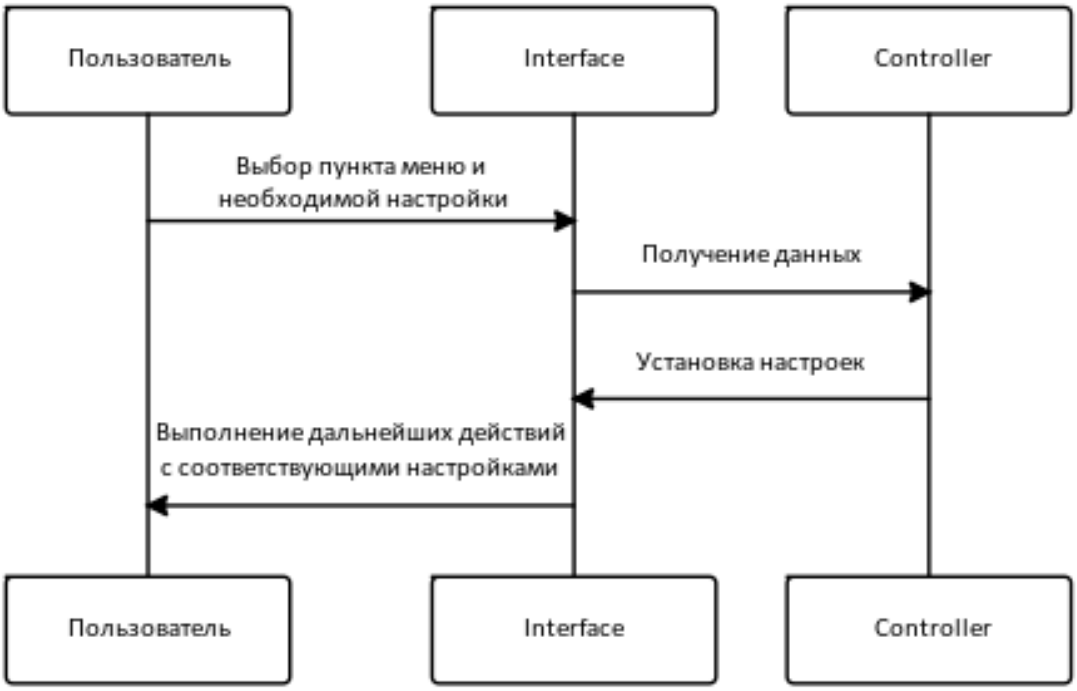
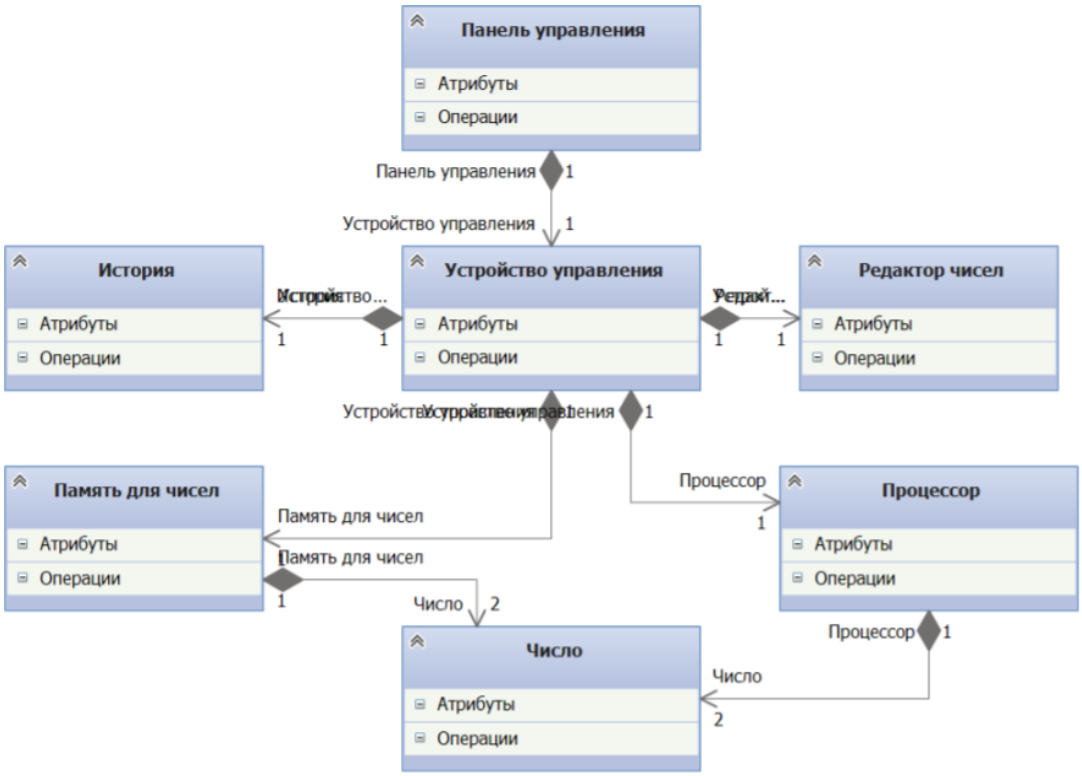


Диаграмма классов для прецедентов



Здесь класс число в зависимости от варианта может быть: ричное число, простая дробь, комплексное число.
Мой вариант: простая дробь.

Спецификации к типам данных

Спецификация типа данных «простые дроби».

ADT TFrac

Данные

Простая дробь (тип TFrac) - это пара целых чисел: числитель и знаменатель (a/b). Простые дроби изменяемые.

Операции

Операции могут вызываться только объектом простая дробь (тип TFrac), указатель на который в них передаётся по умолчанию. При описании операций этот объект называется «сама дробь».

Конструктор	
Начальные значения:	Пара целых чисел (a) и (b).
Процесс:	Инициализирует поля простой дроби (тип TFrac): числитель значением a, знаменатель - (b). В случае необходимости дробь предварительно сокращается. Например: *Конструктор*(6,3) = (2/1) *Конструктор*(0,3) = (0/3).
Конструктор	
Начальные значения:	Строковое представление простой дроби. Например: ‘7/9’.
Процесс:	Инициализирует поля простой дроби (тип TFrac) строкой f =’a/b’. Числитель значением a, знаменатель - b. В случае необходимости дробь предварительно сокращается. Например: Конструктор(‘6/3’) = 2/1 Конструктор (‘0/3’) = 0/3.

Копировать	
Вход:	Нет.
Предусловия:	Нет.
Процесс:	Создаёт копию самой дроби (тип TFrac) с числителем, и знаменателем такими же, как у самой дроби.
Выход:	Простая дробь (тип TFrac). Например: c = 2/1, Копировать(c) = 2/1
Постусловия:	Нет.
Умножить	
Вход:	Простая дробь d (тип TFrac).
Предусловия:	Нет.
Процесс:	Создаёт простую дробь (тип TFrac), полученную умножением самой дроби q = a1/b1 на d = a2/b2 ((a1/b1)*(a2/b2)=(a1* a2)/(b1* b2)).
Выход:	Простая дробь d (тип TFrac).
Постусловия:	Нет.
Вычесть	
Вход:	Простая дробь d (тип TFrac).
Предусловия:	Нет.
Процесс:	Создаёт и возвращает простую дробь (тип TFrac), полученную вычитанием d = a2/b2 из

	самой дроби q = a1/b1:((a1/b1)-(a2/b2)=(a1*b2-a2*b1)/(b1*b2)). Например: q = (1/2), d = (1/2) q.Вычесть(d) = (0/1).
Выход:	Простая дробь d (тип TFrac).
Постусловия:	Нет.
Делить	
Вход:	Простая дробь d (тип TFrac).
Предусловия:	Числитель числа d не равно 0.
Процесс:	Создаёт и возвращает простую дробь (тип TFrac), полученное делением самой дроби q = a1/b1 на дробь d = a2/b2: ((a1/b1)/(a2/b2)=(a1* b2)/(a2*b1)).
Выход:	Простая дробь d (тип TFrac).
Постусловия:	Нет.
Квадрат	
Вход:	Нет.
Предусловия:	Нет.
Процесс:	Создаёт и возвращает простую дробь (тип TFrac), полученную умножением самой дроби на себя: ((a/b)*(a/b)=(a* a)/(b* b)).
Выход:	Простая дробь d (тип TFrac).
Постусловия:	Нет.

Обратное	
Вход:	Нет.
Предусловия:	Нет.
Процесс:	Создаёт и возвращает простую дробь (тип TFrac), полученное делением единицы на саму дробь: 1/((a/b) = b/a.
Выход:	Простая дробь d (тип TFrac).
Постусловия:	Нет.
Минус	
Вход:	Нет.
Предусловия:	Нет.
Процесс:	Создаёт простую дробь, являющуюся разностью простых дробей z и q, где z - простая дробь (0/1), дробь, вызвавшая метод.
Выход:	Простая дробь d (тип TFrac).
Постусловия:	Нет.
Равно	
Вход:	Простая дробь d (тип TFrac).
Предусловия:	Нет.
Процесс:	Сравнивает саму простую дробь q и d. Возвращает значение True, если q и d - тождественные простые дроби, и значение False - в противном случае.
Выход:	Булевское значение.
Постусловия:	Нет.

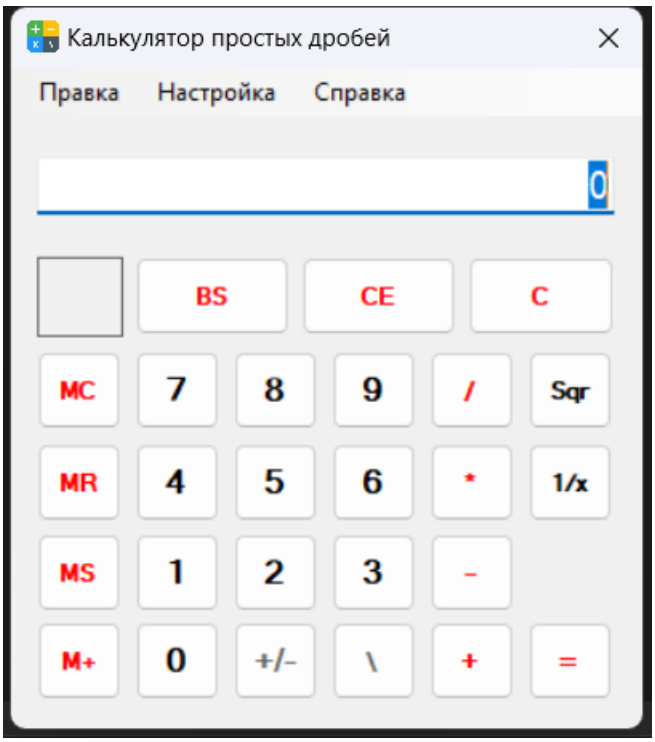
Больше	
Вход:	Простая дробь d (тип TFrac).
Предусловия:	Нет.
Процесс:	Сравнивает саму простую дробь q и d. Возвращает значение True, если q > d, - значение False - в противном случае.
Выход:	Булевское значение.
Постусловия:	Нет.
ВзятьЧислительЧисло	
Вход:	Нет.
Предусловия:	Нет.
Процесс:	Возвращает значение числителя дроби в числовом формате.
Выход:	Вещественное значение.
Постусловия:	Нет.
ВзятьЗнаменательЧисло	
Вход:	Нет.
Предусловия:	Нет.
Процесс:	Возвращает значение знаменателя дроби в числовом формате.
Выход:	Вещественное значение.
Постусловия:	Нет.
ВзятьЧислительСтрока	
Вход:	Нет.

Предусловия:	Нет.
Процесс:	Возвращает значение числителя дроби в строковом формате.
Выход:	Строка.
Постусловия:	Нет.
ВзятьЗнаменательСтрока	
Вход:	Нет.
Предусловия:	Нет.
Процесс:	Возвращает значение знаменателя дроби в строковом формате.
Выход:	Строка.
Постусловия:	Нет.
ВзятьДробьСтрока	
Вход:	Нет.
Предусловия:	Нет.
Процесс:	Возвращает значение простой дроби в строковом формате.
Выход:	Строка.
Постусловия:	Нет.

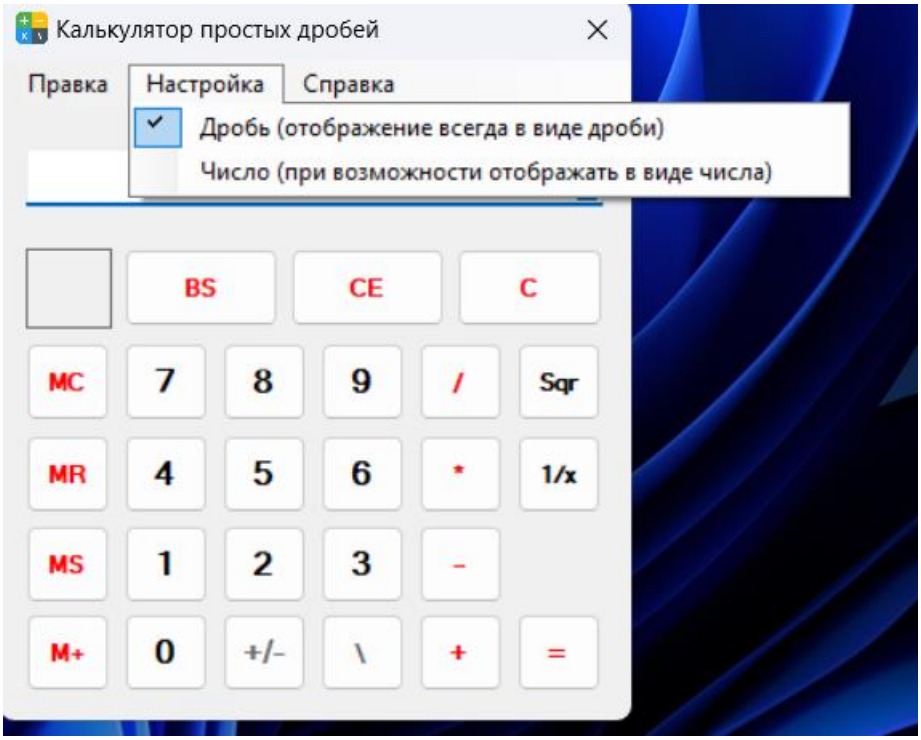
End TFracRatio

Результаты тестирования программы

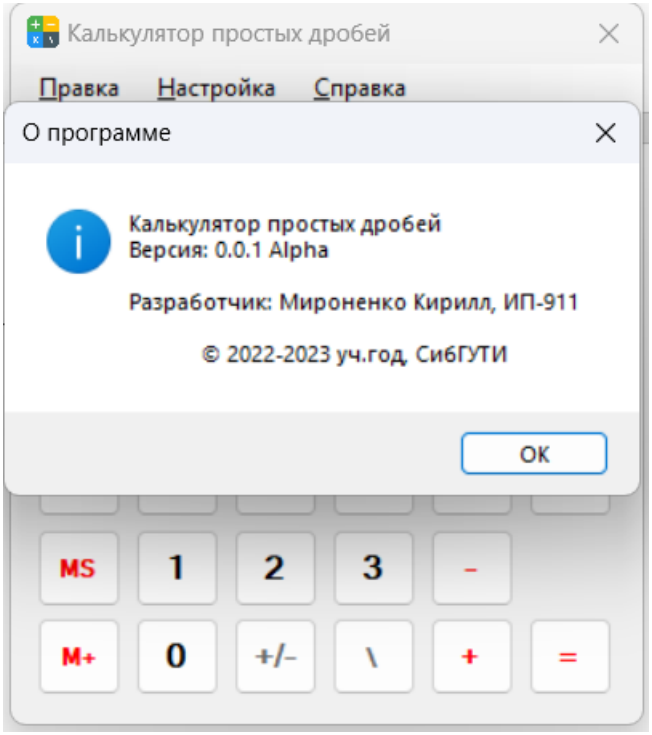
Начало работы:



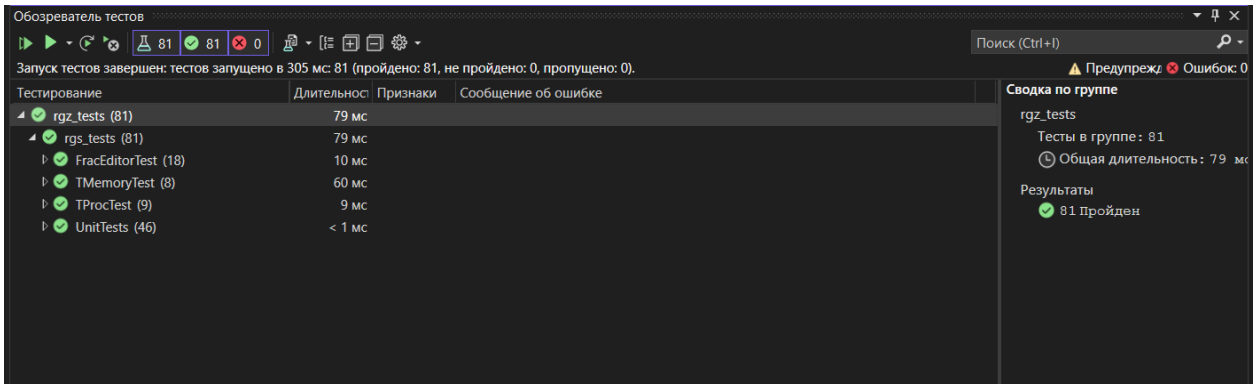
Настройка:



Окно справки:


Калькулятор простых дробей
Правка
Настройка
Справка
О программе
Калькулятор простых дробей
Версия: 0.0.1 Alpha
Разработчик: Мироненко Кирилл, ИП-911
© 2022-2023 уч.год, СибГУТИ
OK
MS
1
2
3
-
M+
0
+/-
\
+
=

Результат работы тестов



Вывод

По итогам расчётно-графического задания были проведено проектирование программ в технологии «абстрактных типов данных», и «объектно-ориентированного программирования» и построение диаграмм UML. Произведена реализация абстрактных типов данных с помощью классов C#. Были использованы библиотеки визуальных компонентов VCL для построения интерфейса. Также было выполнено тестрирование всех классов и закреплены знания по разработке тестов для методов класса в проекте.

Листинг

Исходный код программы

Program.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace rgz
{
    static class Program
    {
        /// <summary>
        /// Главная точка входа для приложения.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new Form1());
        }
    }
}
```

Form1.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace rgz
{
    public partial class Form1 : Form
    {
        ADT_Control<TFrac, TEditor> fracController;

        const string operation_signs = "+-/*";
        bool fracMode = true;
        string memmory_buffer = string.Empty;

        public Form1()
        {
            fracController = new ADT_Control<TFrac, TEditor>();
            InitializeComponent();
        }

        private string NumberBeautifier(string v)
        {
            if (v == "ERROR")
                return v;
            string toReturn = v;
            if (fracMode == true)
```

```
            toReturn = v;
        else if (new TFrac(v).getDenominatorNum() == 1)
            toReturn = new TFrac(v).getNumeratorString();

        return toReturn;
    }

    private void CopyToolStripMenuItem_Click(object sender, EventArgs e)
    {
        memmory_buffer = textBox1.Text;

    }

    private void EnterToolStripMenuItem_Click(object sender, EventArgs e)
    {
        if (memmory_buffer == string.Empty)
        {
            MessageBox.Show("Буфер обмена пуст.\n" +
                "Нечего вставить.",
                "Ошибка",
                MessageBoxButtons.OK,
                MessageBoxIcon.Exclamation);
            return;
        }
        foreach (char i in memmory_buffer)
            textBox1.Text = fracController.ExecComandEditor(CharToEditorCommand(i));
    }

    private static int CharToEditorCommand(char ch)
    {
        int command = 66;
        switch (ch)
        {
            case '0':
                command = 0;
                break;
            case '1':
                command = 1;
                break;
            case '2':
                command = 2;
                break;
            case '3':
                command = 3;
                break;
            case '4':
                command = 4;
                break;
            case '5':
                command = 5;
                break;
            case '6':
                command = 6;
                break;
            case '7':
                command = 7;
                break;
            case '8':
                command = 8;
                break;
            case '9':
                command = 9;
                break;
            case '.':
                command = 10;
```

```
                break;
            case '-':
                command = 11;
                break;
        }

        return command;
    }
    private static int CharToOperationsCommand<T>(char ch) where T : TFrac, new()
    {
        int command = 0;
        switch (ch)
        {
            case '+':
                command = 1;
                break;
            case '-':
                command = 2;
                break;
            case '*':
                command = 3;
                break;
            case '/':
                command = 4;
                break;
        }

        return command;
    }

    private static int KeyCodeToEditorCommand(Keys ch)
    {
        int command = 14;
        switch (ch)
        {
            case Keys.Back:
                command = 12;
                break;
            case Keys.Delete:
            case Keys.Escape:
                command = 13;
                break;
        }

        return command;
    }


    private void AboutToolStripMenuItem1_Click(object sender, EventArgs e)
    {
        MessageBox.Show("Калькулятор простых дробей\nВерсия: 0.0.1
Alpha\n\nРазработчик: Мироненко Кирилл, ИП-911\n\n            © 2022-2023 уч.год,
СибГУТИ", "О программе", MessageBoxButtons.OK, MessageBoxIcon.Information);

    }

    private void Button_Number_Edit(object sender, EventArgs e)
    {
        Button button = (Button)sender;
        int tag_command = Convert.ToInt32(button.Tag.ToString());
        textBox1.Text = fracController.ExecComandEditor(tag_command);
    }

    private void Button_Number_Operation(object sender, EventArgs e)
    {
```

```
            Button button = (Button)sender;
            int tag_command = Convert.ToInt32(button.Tag.ToString());
            textBox1.Text = NumberBeautifier(fracController.ExecOperation(tag_command));
        }

        private void Button_Number_Function(object sender, EventArgs e)
        {
            Button button = (Button)sender;
            int tag_command = Convert.ToInt32(button.Tag.ToString());
            textBox1.Text = NumberBeautifier(fracController.ExecFunction(tag_command));
        }

        private void Button_Memory(object sender, EventArgs e)
        {
            Button button = (Button)sender;
            int tag_command = Convert.ToInt32(button.Tag.ToString());
            dynamic exec = fracController.ExecCommandMemory(tag_command, textBox1.Text);
            if (tag_command == 3)
                textBox1.Text = exec.Item1.ToString();
            label1.Text = exec.Item2 == true ? "M" : string.Empty;
        }

        private void Button_Reset(object sender, EventArgs e)
        {
            textBox1.Text = fracController.Reset();
            label1.Text = string.Empty;
        }

        private void Button_Calculate(object sender, EventArgs e)
        {
            textBox1.Text = NumberBeautifier(fracController.Calculate());
        }

        private void Form1_KeyDown(object sender, KeyEventArgs e)
        {

            if (e.KeyCode == Keys.Enter)
                CalculateButton.PerformClick();
            else
            {
                int command = KeyCodeToEditorCommand(e.KeyCode);
                if (command != 14)
                    textBox1.Text = fracController.ExecComandEditor(command);
            }

        }

        private void Form1_KeyPress(object sender, KeyPressEventArgs e)
        {
            if (e.KeyChar == (char)Keys.Enter)
                e.Handled = true;
            if (e.KeyChar >= '0' && e.KeyChar <= '9' || e.KeyChar == '.')
                textBox1.Text =
fracController.ExecComandEditor(CharToEditorCommand(e.KeyChar));
            else if (operation_signs.Contains(e.KeyChar))
                textBox1.Text =
NumberBeautifier(fracController.ExecOperation(CharToOperationsCommand<TFrac>(e.KeyChar)))
;
        }

        private void FracToolStripMenuItem_Click(object sender, EventArgs e)
        {
            FracToolStripMenuItem.Checked = true;
            NumToolStripMenuItem.Checked = false;
            fracMode = true;
```

```
        }

        private void NumToolStripMenuItem_Click(object sender, EventArgs e)
        {
            FracToolStripMenuItem.Checked = false;
            NumToolStripMenuItem.Checked = true;
            fracMode = false;
        }

        private void Form1_KeyUp(object sender, KeyEventArgs e)
        {
            //MessageBox.Show($"KeyUp code: {e.KeyCode}, value: {e.KeyValue}, modifiers:
{e.Modifiers}" + "\r\n");

        }


    }
}
```

TEditor.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace rgz
{
    public class TEditor
    {

        const string Separator = "/";
        const string ZeroFraction = "0/";
        const int max_numerator_length = 14;
        const int max_denominator_length = 22;
        private string fraction;

        public string Fraction
        {
            get
            {
                return fraction;
            }

            set
            {
                fraction = new TFrac(value).ToString();
            }
        }

        public TEditor()
        {
            fraction = "0";
        }

        public TEditor(long a, long b)
        {
            fraction = new TFrac(a, b).ToString();
        }

        public TEditor(string frac)
        {
```

```
            fraction = new TFrac(frac).ToString();
        }

        public void SetEditor(TFrac frac)
        {
            fraction = frac.ToString();
        }

        public bool IsZero()
        {
            return fraction.StartsWith(ZeroFraction) || fraction.StartsWith("-" +
ZeroFraction) || fraction == "0" || fraction == "-0";
        }

        public string ToggleMinus()
        {
            if (fraction[0] == '-')
                fraction = fraction.Remove(0, 1);
            else
                fraction = '-' + fraction;

            return fraction;
        }

        public string AddNumber(long a)
        {
            if (!fraction.Contains(Separator) && fraction.Length > max_numerator_length)
                return fraction;
            else if (fraction.Length > max_denominator_length)
                return fraction;
            if (a < 0 || a > 9)
                return fraction;
            if (a == 0)
                AddZero();
            else if (IsZero())
                fraction = fraction.First() == '-' ? "-" + a.ToString() : a.ToString();
            else
                fraction += a.ToString();

            return fraction;
        }

        public string AddZero()
        {
            if (IsZero())
                return fraction;
            if (fraction.Last().ToString() == Separator)
                return fraction;
            fraction += "0";

            return fraction;
        }

        public string RemoveSymbol()
        {
            if (fraction.Length == 1)
                fraction = "0";
            else if (fraction.Length == 2 && fraction.First() == '-')
                fraction = "-0";
            else
                fraction = fraction.Remove(fraction.Length - 1);

            return fraction;
        }
```

```
public string Clear()
{
    fraction = "0";

    return fraction;
}

public string Edit(int command)
{
    switch (command)
    {
        case 0:
            AddZero();
            break;
        case 1:
            AddNumber(1);
            break;
        case 2:
            AddNumber(2);
            break;
        case 3:
            AddNumber(3);
            break;
        case 4:
            AddNumber(4);
            break;
        case 5:
            AddNumber(5);
            break;
        case 6:
            AddNumber(6);
            break;
        case 7:
            AddNumber(7);
            break;
        case 8:
            AddNumber(8);
            break;
        case 9:
            AddNumber(9);
            break;
        case 10:
            ToggleMinus();
            break;
        case 11:
            AddSeparator();
            break;
        case 12:
            RemoveSymbol();
            break;
        case 13:
            Clear();
            break;
        default:
            break;
    }

    return fraction;
}

public string AddSeparator()
{
    if (!fraction.Contains(Separator))
        fraction += Separator;
```

```
            return fraction;
        }

        public override string ToString()
        {
            return Fraction;
        }
    }
}
```

TFrac.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text.RegularExpressions;

namespace rgz
{
    public class TFrac
    {
        private long numerator;
        private long denominator;

        /// Числитель
        public long Numerator
        {
            get
            {
                return numerator;
            }
            set
            {
                numerator = value;
            }
        }

        /// Знаменатель
        public long Denominator
        {
            get
            {
                return denominator;
            }
            set
            {
                denominator = value;
            }
        }

        static void Swap<T>(ref T lhs, ref T rhs)
        {
            T temp;
            temp = lhs;
            lhs = rhs;
            rhs = temp;
        }

        public static long GCD(long a, long b)
        {
            a = Math.Abs(a);
            b = Math.Abs(b);
            while (b > 0)
            {
```

```
        a %= b;
        Swap(ref a, ref b);
    }
    return a;
}

public TFrac()
{
    numerator = 0;
    denominator = 1;
}

public TFrac(long a, long b)
{
    if (a < 0 && b < 0)
    {
        a *= -1;
        b *= -1;
    }
    else if (b < 0 && a > 0)
    {
        b *= -1;
        a *= -1;
    }
    else if (a == 0 && b == 0 || b == 0 || a == 0 && b == 1)
    {
        numerator = 0;
        denominator = 1;
        return;
    }
    numerator = a;
    denominator = b;
    long gcdRes = GCD(a, b);
    if (gcdRes > 1)
    {
        numerator /= gcdRes;
        denominator /= gcdRes;
    }
}

public TFrac(string frac)
{
    Regex FracRegex = new Regex(@"^-?(\d+)/(\d+)$");
    Regex NumberRegex = new Regex(@"^-?\d+/?$");
    if (FracRegex.IsMatch(frac))
    {
        List<string> FracSplited = frac.Split('/').ToList();
        numerator = Convert.ToInt64(FracSplited[0]);
        denominator = Convert.ToInt64(FracSplited[1]);
        if (denominator == 0)
        {
            numerator = 0;
            denominator = 1;
            return;
        }
        long gcd = GCD(numerator, denominator);
        if (gcd > 1)
        {
            numerator /= gcd;
            denominator /= gcd;
        }
        return;
    }
    else if (NumberRegex.IsMatch(frac))
    {
```

```
                if (long.TryParse(frac, out long NewNumber))
                    numerator = NewNumber;
                else
                    numerator = 0;
                denominator = 1;
                return;
            }
            else
            {
                numerator = 0;
                denominator = 1;
                return;
            }
        }

        public TFrac Copy()
        {
            return (TFrac)this.MemberwiseClone();
        }

        public void SetString(string str)
        {
            TFrac TempFrac = new TFrac(str);
            numerator = TempFrac.numerator;
            denominator = TempFrac.denominator;
        }

        public TFrac Add(TFrac a)
        {
            return new TFrac(numerator * a.denominator + denominator * a.numerator,
denominator * a.denominator);
        }

        public TFrac Mul(TFrac b)
        {
            return new TFrac(numerator * b.numerator, denominator * b.denominator);
        }

        public TFrac Sub(TFrac b)
        {
            return new TFrac(numerator * b.denominator - denominator * b.numerator,
denominator * b.denominator);
        }

        public TFrac Div(TFrac b)
        {
            return new TFrac(numerator * b.denominator, denominator * b.numerator);
        }

        public TFrac Square()
        {
            return new TFrac(numerator * numerator, denominator * denominator);
        }

        public TFrac Reverse()
        {
            return new TFrac(denominator, numerator);
        }

        public TFrac Minus()
        {
            return new TFrac(-numerator, denominator);
        }

        public bool Equal(TFrac b)
```

```
        {
            return numerator == b.numerator && denominator == b.denominator;
        }

        public static bool operator >(TFrac a, TFrac b)
        {
            return (Convert.ToDouble(a.numerator) / Convert.ToDouble(a.denominator)) >
(Convert.ToDouble(b.numerator) / Convert.ToDouble(b.denominator));
        }

        public static bool operator <(TFrac a, TFrac b)
        {
            return (Convert.ToDouble(a.numerator) / Convert.ToDouble(a.denominator)) <
(Convert.ToDouble(b.numerator) / Convert.ToDouble(b.denominator));
        }

        public static implicit operator string(TFrac v)
        {
            throw new NotImplementedException();
        }

        public long getNumeratorNum()
        {
            return numerator;
        }

        public long getDenominatorNum()
        {
            return denominator;
        }

        public string getNumeratorString()
        {
            return numerator.ToString();
        }

        public string getDenominatorString()
        {
            return denominator.ToString();
        }

        public override string ToString()
        {
            return getNumeratorString() + "/" + getDenominatorString();
        }
    }
}
```

TMemory.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace rgz
{
    public class TMemory<T> where T : TFrac, new()
    {
        T number;
        bool state;
        public T FNumber
        {
```

```
    get
    {
        state = true;
        return number;
    }
    set
    {
        number = value;
        state = true;
    }
}
public bool FState
{
    get
    {
        return state;
    }

    set
    {
        state = value;
    }
}

public TMemory()
{
    number = new T();
    state = false;
}

public TMemory(T num)
{
    number = num;
    state = false;
}

public T Add(T num)
{
    state = true;
    dynamic a = number;
    dynamic b = num;
    number = a.Add(b);
    return number;
}

public void Clear()
{
    number = new T();
    state = false;
}

public (T, bool) Edit(int command, T newNumber)
{
    switch (command)
    {
        case 0:
            state = true;
            number = newNumber;
            break;
        case 1:
            dynamic a = number;
            dynamic b = newNumber;
            number = a.Add(b);
            break;
        case 2:
```

```
                    Clear();
                    break;
            }
            return (number, state);
        }
    }
}
```

ADT_Control.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace rgz
{
    public class ADT_Control<T, TEditor_>
        where T : TFrac, new()
        where TEditor_ : TEditor, new()
    {
        public enum ADT_Control_State { cStart, cEditing, FunDone, cValDone, cExpDone,
cOpDone, cOpChange, cError }

        ADT_Control_State calcState;
        TEditor editor;
        ADT_Proc<T> proc;
        TMemory<T> memory;
        //public THistory history = new THistory();

        public ADT_Control_State CurState
        {
            get
            {
                return calcState;
            }
            set
            {
                calcState = value;
            }
        }
        public ADT_Proc<T> Proc
        {
            get
            {
                return proc;
            }
            set
            {
                proc = value;
            }
        }

        public TMemory<T> Memory
        {
            get
            {
                return memory;
            }
            set
            {
                memory = value;
            }
```

```
}

public TEditor Edit
{
    get
    {
        return editor;
    }
    set
    {
        editor = value;
    }
}

public ADT_Control()
{
    Edit = new TEditor();
    Proc = new ADT_Proc<T>();
    Memory = new TMemory<T>();
    CurState = ADT_Control_State.cStart;
}

public string Reset()
{
    Edit.Clear();
    Proc.ResetProc();
    Memory.Clear();
    CurState = ADT_Control_State.cStart;
    return Edit.ToString();
}

public string ExecComandEditor(int command)
{
    string toReturn;
    if (CurState == ADT_Control_State.cExpDone)
    {
        Proc.ResetProc();
        CurState = ADT_Control_State.cStart;
    }
    if (CurState != ADT_Control_State.cStart)
        CurState = ADT_Control_State.cEditing;
    toReturn = Edit.Edit(command);
    T tmp = new T();
    tmp.SetString(toReturn);
    proc.Right_operand = tmp;
  //  history.AddRecord(toReturn, command.ToString());

    return toReturn;
}

public string ExecOperation(int operation)
{
    if (operation == 0)
        return Edit.Fraction;
    string toReturn;
    try
    {
        switch (CurState)
        {
            case ADT_Control_State.cStart:
                Proc.Left_Result_operand = Proc.Right_operand;
                Proc.Operation = operation;
                CurState = ADT_Control_State.cOpDone;
                Edit.Clear();
                break;
```

```
            case ADT_Control_State.cEditing:
                Proc.DoOperation();
                Proc.Operation = operation;
                Edit.Clear();
                CurState = ADT_Control_State.cOpDone;
                break;
            case ADT_Control_State.FunDone:
                if (Proc.Operation == 0)
                    Proc.Left_Result_operand = Proc.Right_operand;
                else
                    Proc.DoOperation();
                Proc.Operation = operation;
                Edit.Clear();
                CurState = ADT_Control_State.cOpChange;
                Proc.Right_operand = Proc.Left_Result_operand;
                break;
            case ADT_Control_State.cOpDone:
                CurState = ADT_Control_State.cOpChange;
                Edit.Clear();
                break;
            case ADT_Control_State.cValDone:
                break;
            case ADT_Control_State.cExpDone:
                Proc.Operation = operation;
                Proc.Right_operand = Proc.Left_Result_operand;
                CurState = ADT_Control_State.cOpChange;
                Edit.Clear();
                break;
            case ADT_Control_State.cOpChange:
                Proc.Operation = operation;
                Edit.Clear();
                break;
            case ADT_Control_State.cError:
                Proc.ResetProc();
                return "ERR";
        }
        toReturn = Proc.Left_Result_operand.ToString();
    }
    catch
    {
        Reset();
        return "ERROR";
    }
  //  history.AddRecord(toReturn, oper.ToString());

    return toReturn;
}

public string ExecFunction(int function)
{
    string toReturn;
    try
    {
        if (CurState == ADT_Control_State.cExpDone)
        {
            Proc.Right_operand = Proc.Left_Result_operand;
            Proc.Operation = 0;
        }
        Proc.DoFunction(function);
        CurState = ADT_Control_State.FunDone;
        toReturn = Proc.Right_operand.ToString();
    }
    catch
    {
        Reset();
```

```
                return "ERROR";
            }
          //  history.AddRecord(toReturn, func.ToString());

            return toReturn;
        }

        public string Calculate()
        {
            string ToReturn;
            try
            {
                if (CurState == ADT_Control_State.cStart)
                    Proc.Left_Result_operand = Proc.Right_operand;
                Proc.DoOperation();
                CurState = ADT_Control_State.cExpDone;
                Edit.SetEditor(Proc.Left_Result_operand);
                ToReturn = Proc.Left_Result_operand.ToString();
            }
            catch
            {
                Reset();
                return "ERROR";
            }

            return ToReturn;
        }

        public (T, bool) ExecCommandMemory(int command, string str)
        {
            T tmp = new T();
            tmp.SetString(str);
            (T, bool) obj = (null, false);
            try
            {
                obj = Memory.Edit(command, tmp);
            }
            catch
            {
                Reset();
                return obj;
            }
            if (command == 3)
            {
                Edit.Fraction = obj.Item1.ToString();
                Proc.Right_operand = obj.Item1;
            }

            return obj;
        }
    }
}
```

ADT_Proc.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace rgz
{
    public class ADT_Proc<T> where T : TFrac, new()
```

```
{
    T left_result_operand;
    T right_operand;
    int operation;

    public T Left_Result_operand
    {
        get
        {
            return left_result_operand;
        }

        set
        {
            left_result_operand = value;
        }
    }
    public T Right_operand
    {
        get
        {
            return right_operand;
        }

        set
        {
            right_operand = value;
        }
    }

    public int Operation
    {
        get
        {
            return operation;
        }

        set
        {
            operation = value;
        }
    }

    public ADT_Proc()
    {
        operation = 0;
        left_result_operand = new T();
        right_operand = new T();
    }

    public ADT_Proc(T leftObj, T rightObj)
    {
        operation = 0;
        left_result_operand = leftObj;
        right_operand = rightObj;
    }

    public void ResetProc()
    {
        operation = 0;
        T newObj = new T();
        left_result_operand = right_operand = newObj;
    }

    public void DoOperation()
```

```
        {
            try
            {
                dynamic a = left_result_operand;
                dynamic b = right_operand;
                switch (operation)
                {
                    case 1:
                        left_result_operand = a.Add(b);

                        break;
                    case 2:
                        left_result_operand = a.Sub(b);
                        break;
                    case 3:
                        left_result_operand = a.Mul(b);
                        break;
                    case 4:
                        left_result_operand = a.Div(b);
                        break;
                    default:
                        left_result_operand = right_operand;
                        break;
                }
            }
            catch
            {
                throw new System.OverflowException();
            }
        }

        public void DoFunction(int function)
        {
            dynamic a = right_operand;
            switch (function)
            {
                case 0:
                    a = a.Reverse();
                    right_operand = (T)a;
                    break;
                case 1:
                    a = a.Square();
                    right_operand = (T)a;
                    break;
                default:
                    break;
            }
        }
    }
}
```

Исходный код тестов

UnitTests.cs

```
using Microsoft.VisualStudio.TestTools.UnitTesting;
using rgz;

namespace rgs_tests
{
    [TestClass]
    public class UnitTest1
    {
```

```
[TestMethod]
public void InitString1()
{
    string fracString = "1/2";
    TFrac fracClass = new TFrac(fracString);
    Assert.AreEqual(fracString, fracClass.ToString());
}

[TestMethod]
public void InitString2()
{
    string fracString = "111/2";
    TFrac fracClass = new TFrac(fracString);
    Assert.AreEqual(fracString, fracClass.ToString());
}

[TestMethod]
public void InitString3()
{
    string fracString = "-100/60";
    TFrac fracClass = new TFrac(fracString);
    string Expect = "-5/3";
    Assert.AreEqual(Expect, fracClass.ToString());
}

[TestMethod]
public void InitString4()
{
    string fracString = "00000003/000004";
    TFrac fracClass = new TFrac(fracString);
    string Expect = "3/4";
    Assert.AreEqual(Expect, fracClass.ToString());
}

[TestMethod]
public void InitString5()
{
    string fracString = "-00000003/000004";
    TFrac fracClass = new TFrac(fracString);
    string Expect = "-3/4";
    Assert.AreEqual(Expect, fracClass.ToString());
}

[TestMethod]
public void InitNumber1()
{
    TFrac fracClass = new TFrac(1, 2);
    string Expect = "1/2";
    Assert.AreEqual(Expect, fracClass.ToString());
}

[TestMethod]
public void InitNumber2()
{
    TFrac fracClass = new TFrac(100, 100);
    string Expect = "1/1";
    Assert.AreEqual(Expect, fracClass.ToString());
}

[TestMethod]
public void InitNumber3()
{
    TFrac fracClass = new TFrac(-100, -99);
    string Expect = "100/99";
    Assert.AreEqual(Expect, fracClass.ToString());
```

```
}

[TestMethod]
public void InitNumber4()
{
    TFrac fracClass = new TFrac(0, 0);
    string Expect = "0/1";
    Assert.AreEqual(Expect, fracClass.ToString());
}

[TestMethod]
public void InitNumber5()
{
    TFrac fracClass = new TFrac(50, -5);
    string fracCompar = "-10/1";
    Assert.AreEqual(fracCompar, fracClass.ToString());
}

[TestMethod]
public void Add1()
{
    TFrac fracClass1 = new TFrac(1, 4);
    TFrac fracClass2 = new TFrac(-3, 4);
    fracClass2 = fracClass1.Add(fracClass2);
    string answer = "-1/2";
    Assert.AreEqual(answer, fracClass2.ToString());
}

[TestMethod]
public void Add2()
{
    TFrac fracClass1 = new TFrac(-1, 2);
    TFrac fracClass2 = new TFrac(-1, 2);
    fracClass2 = fracClass1.Add(fracClass2);
    string answer = "-1/1";
    Assert.AreEqual(answer, fracClass2.ToString());
}

[TestMethod]
public void Add3()
{
    TFrac fracClass1 = new TFrac(-6, 2);
    TFrac fracClass2 = new TFrac(6, 2);
    fracClass2 = fracClass1.Add(fracClass2);
    string answer = "0/1";
    Assert.AreEqual(answer, fracClass2.ToString());
}

[TestMethod]
public void Add4()
{
    TFrac fracClass1 = new TFrac(50, 3);
    TFrac fracClass2 = new TFrac(0, 1);
    fracClass2 = fracClass1.Add(fracClass2);
    string answer = "50/3";
    Assert.AreEqual(answer, fracClass2.ToString());
}

[TestMethod]
public void Add5()
{
    TFrac fracClass1 = new TFrac(0, 1);
    TFrac fracClass2 = new TFrac(0, 1);
    fracClass2 = fracClass1.Add(fracClass2);
    string answer = "0/1";
```

```
    Assert.AreEqual(answer, fracClass2.ToString());
}

[TestMethod]
public void Multiply1()
{
    TFrac fracClass1 = new TFrac(-1, 2);
    TFrac fracClass2 = new TFrac(-1, 2);
    fracClass2 = fracClass1.Mul(fracClass2);
    string answer = "1/4";
    Assert.AreEqual(answer, fracClass2.ToString());
}

[TestMethod]
public void Multiply2()
{
    TFrac fracClass1 = new TFrac(1, 6);
    TFrac fracClass2 = new TFrac(0, 1);
    fracClass2 = fracClass1.Mul(fracClass2);
    string answer = "0/1";
    Assert.AreEqual(answer, fracClass2.ToString());
}

[TestMethod]
public void Multiply3()
{
    TFrac fracClass1 = new TFrac(1, 6);
    TFrac fracClass2 = new TFrac(1, 6);
    fracClass2 = fracClass1.Mul(fracClass2);
    string answer = "1/36";
    Assert.AreEqual(answer, fracClass2.ToString());
}

[TestMethod]
public void Multiply4()
{
    TFrac fracClass1 = new TFrac(-1, 6);
    TFrac fracClass2 = new TFrac(12, 1);
    fracClass2 = fracClass1.Mul(fracClass2);
    string answer = "-2/1";
    Assert.AreEqual(answer, fracClass2.ToString());
}

[TestMethod]
public void Multiply5()
{
    TFrac fracClass1 = new TFrac(-1, 6);
    TFrac fracClass2 = new TFrac(12, 1);
    fracClass2 = fracClass1.Mul(fracClass2);
    string answer = "-2/1";
    Assert.AreEqual(answer, fracClass2.ToString());
}

[TestMethod]
public void Substract1()
{
    TFrac fracClass1 = new TFrac(0, 1);
    TFrac fracClass2 = new TFrac(1, 1);
    fracClass2 = fracClass1.Sub(fracClass2);
    string answer = "-1/1";
    Assert.AreEqual(answer, fracClass2.ToString());
}

[TestMethod]
public void Substract2()
```

```
{
    TFrac fracClass1 = new TFrac(5, 1);
    TFrac fracClass2 = new TFrac(1, 1);
    fracClass2 = fracClass1.Sub(fracClass2);
    string answer = "4/1";
    Assert.AreEqual(answer, fracClass2.ToString());
}

[TestMethod]
public void Substract3()
{
    TFrac fracClass1 = new TFrac(1, 2);
    TFrac fracClass2 = new TFrac(1, 2);
    fracClass2 = fracClass1.Sub(fracClass2);
    string answer = "0/1";
    Assert.AreEqual(answer, fracClass2.ToString());
}

[TestMethod]
public void Substract4()
{
    TFrac fracClass1 = new TFrac(-1, 6);
    TFrac fracClass2 = new TFrac(-1, 6);
    fracClass2 = fracClass1.Sub(fracClass2);
    string answer = "0/1";
    Assert.AreEqual(answer, fracClass2.ToString());
}

[TestMethod]
public void Substract5()
{
    TFrac fracClass1 = new TFrac(-1, 6);
    TFrac fracClass2 = new TFrac(2, 6);
    fracClass2 = fracClass1.Sub(fracClass2);
    string answer = "-1/2";
    Assert.AreEqual(answer, fracClass2.ToString());
}

[TestMethod]
public void Divide1()
{
    TFrac fracClass1 = new TFrac(5, 6);
    TFrac fracClass2 = new TFrac(1, 1);
    fracClass2 = fracClass1.Div(fracClass2);
    string answer = "5/6";
    Assert.AreEqual(answer, fracClass2.ToString());
}

[TestMethod]
public void Divide2()
{
    TFrac fracClass1 = new TFrac(1, 1);
    TFrac fracClass2 = new TFrac(5, 6);
    fracClass2 = fracClass1.Div(fracClass2);
    string answer = "6/5";
    Assert.AreEqual(answer, fracClass2.ToString());
}

[TestMethod]
public void Divide3()
{
    TFrac fracClass1 = new TFrac(0, 1);
    TFrac fracClass2 = new TFrac(5, 6);
    fracClass2 = fracClass1.Div(fracClass2);
    string answer = "0/1";
```

```
    Assert.AreEqual(answer, fracClass2.ToString());
}

[TestMethod]
public void Divide4()
{
    TFrac fracClass1 = new TFrac(2, 3);
    TFrac fracClass2 = new TFrac(7, 4);
    fracClass2 = fracClass1.Div(fracClass2);
    string answer = "8/21";
    Assert.AreEqual(answer, fracClass2.ToString());
}

[TestMethod]
public void Divide5()
{
    TFrac fracClass1 = new TFrac(2, 3);
    TFrac fracClass2 = new TFrac(2, 3);
    fracClass2 = fracClass1.Div(fracClass2);
    string answer = "1/1";
    Assert.AreEqual(answer, fracClass2.ToString());
}

[TestMethod]
public void Reverse1()
{
    TFrac fracClass = new TFrac(-2, 3);
    fracClass = fracClass.Reverse() as TFrac;
    string answer = "-3/2";
    Assert.AreEqual(answer, fracClass.ToString());
}

[TestMethod]
public void Reverse2()
{
    TFrac fracClass = new TFrac(0, 1);
    fracClass = fracClass.Reverse() as TFrac;
    string answer = "0/1";
    Assert.AreEqual(answer, fracClass.ToString());
}

[TestMethod]
public void Reverse3()
{
    TFrac fracClass = new TFrac(5, 6);
    fracClass = fracClass.Reverse() as TFrac;
    string answer = "6/5";
    Assert.AreEqual(answer, fracClass.ToString());
}

[TestMethod]
public void Square1()
{
    TFrac fracClass = new TFrac(2, 3);
    fracClass = fracClass.Square() as TFrac;
    string answer = "4/9";
    Assert.AreEqual(answer, fracClass.ToString());
}

[TestMethod]
public void Square2()
{
    TFrac fracClass = new TFrac(0, 1);
    fracClass = fracClass.Square() as TFrac;
    string answer = "0/1";
```

```
    Assert.AreEqual(answer, fracClass.ToString());
}

[TestMethod]
public void Square3()
{
    TFrac fracClass = new TFrac(-2, 3);
    fracClass = fracClass.Square() as TFrac;
    string answer = "4/9";
    Assert.AreEqual(answer, fracClass.ToString());
}

[TestMethod]
public void Equal1()
{
    TFrac fracClass1 = new TFrac(1, 3);
    TFrac fracClass2 = new TFrac(1, 3);
    Assert.IsTrue(fracClass1.Equal(fracClass2));
}

[TestMethod]
public void Equal2()
{
    TFrac fracClass1 = new TFrac(0, 6);
    TFrac fracClass2 = new TFrac(1, 6);
    Assert.IsFalse(fracClass1.Equal(fracClass2));
}

[TestMethod]
public void Equal3()
{
    TFrac fracClass1 = new TFrac(-1, 6);
    TFrac fracClass2 = new TFrac(-1, 6);
    Assert.IsTrue(fracClass1.Equal(fracClass2));
}

[TestMethod]
public void Equal4()
{
    TFrac fracClass1 = new TFrac(-1, 7);
    TFrac fracClass2 = new TFrac(1, 7);
    Assert.IsFalse(fracClass1.Equal(fracClass2));
}

[TestMethod]
public void Equal5()
{
    TFrac fracClass1 = new TFrac(1, 6);
    TFrac fracClass2 = new TFrac(0, 1);
    Assert.IsFalse(fracClass1.Equal(fracClass2));
}

[TestMethod]
public void Greater1()
{
    TFrac fracClass1 = new TFrac(1, 6);
    TFrac fracClass2 = new TFrac(0, 1);
    Assert.IsTrue(fracClass1 > fracClass2);
}

[TestMethod]
public void Greater2()
{
    TFrac fracClass1 = new TFrac(0, 1);
    TFrac fracClass2 = new TFrac(0, 1);
```

```
        Assert.IsFalse(fracClass1 > fracClass2);
    }

    [TestMethod]
    public void Greater3()
    {
        TFrac fracClass1 = new TFrac(-1, 6);
        TFrac fracClass2 = new TFrac(0, 1);
        Assert.IsFalse(fracClass1 > fracClass2);
    }

    [TestMethod]
    public void Greater4()
    {
        TFrac fracClass1 = new TFrac(17, 3);
        TFrac fracClass2 = new TFrac(16, 3);
        Assert.IsTrue(fracClass1 > fracClass2);
    }

    [TestMethod]
    public void Greater5()
    {
        TFrac fracClass1 = new TFrac(-2, 3);
        TFrac fracClass2 = new TFrac(-1, 3);
        Assert.IsFalse(fracClass1 > fracClass2);
    }
}

[TestClass]
public class FracEditorTest
{
    [TestMethod]
    public void TestInit1()
    {
        TEditor testClass = new TEditor();
        string input = "3/4";
        testClass.Fraction = input;
        Assert.AreEqual(input, testClass.Fraction);
    }
    [TestMethod]
    public void TestInit2()
    {
        TEditor testClass = new TEditor();
        string input = "-16/3";
        testClass.Fraction = input;
        Assert.AreEqual(input, testClass.Fraction);
    }
    [TestMethod]
    public void TestInit3()
    {
        TEditor testClass = new TEditor();
        string input = "0/8";
        testClass.Fraction = input;
        string result = "0/1";
        Assert.AreEqual(result, testClass.Fraction);
    }
    [TestMethod]
    public void TestInit4()
    {
        TEditor testClass = new TEditor();
        string input = "-17/4";
        testClass.Fraction = input;
        Assert.AreEqual(input, testClass.Fraction);
    }
```

```
[TestMethod]
public void TestInit5()
{
    TEditor testClass = new TEditor();
    string input = "0/1";
    testClass.Fraction = input;
    Assert.AreEqual(input, testClass.Fraction);
}

[TestMethod]
public void TestInit6()
{
    TEditor testClass = new TEditor();
    string input = "666/6666";
    testClass.Fraction = input;
    string result = "111/1111";
    Assert.AreEqual(result, testClass.Fraction);
}

[TestMethod]
public void TestInit7()
{
    TEditor testClass = new TEditor();
    string input = "aaaa";
    testClass.Fraction = input;
    string result = "0/1";
    Assert.AreEqual(result, testClass.Fraction);
}

[TestMethod]
public void TestInit8()
{
    TEditor testClass = new TEditor();
    string input = "0/1";
    testClass.Fraction = input;
    Assert.AreEqual(input, testClass.Fraction);
}

[TestMethod]
public void TestInit10()
{
    TEditor testClass = new TEditor();
    string input = "16/000000";
    testClass.Fraction = input;
    string result = "0/1";
    Assert.AreEqual(result, testClass.Fraction);
}

[TestMethod]
public void hasZero1()
{
    TEditor testClass = new TEditor("14/3");
    Assert.AreEqual(false, testClass.IsZero());
}
[TestMethod]
public void hasZero2()
{
    TEditor testClass = new TEditor("16/00000");
    Assert.AreEqual(true, testClass.IsZero());
}

[TestMethod]
public void ToogleMinus1()
{
    TEditor testClass = new TEditor("14/3");
```

```
    testClass.ToggleMinus();
    string result = "-14/3";
    Assert.AreEqual(result, testClass.ToString());
}
[TestMethod]
public void ToogleMinus2()
{
    TEditor testClass = new TEditor("-14/3");
    testClass.ToggleMinus();
    string result = "14/3";
    Assert.AreEqual(result, testClass.ToString());
}

[TestMethod]
public void AddDeleteTest1()
{
    TEditor testClass = new TEditor("123/123");
    testClass.AddNumber(0);
    testClass.AddNumber(1);
    testClass.AddNumber(3);
    testClass.AddSeparator();
    testClass.ToggleMinus();
    string result = "-1/1013";
    Assert.AreEqual(result, testClass.ToString());
}
[TestMethod]
public void AddDeleteTest2()
{
    TEditor testClass = new TEditor(123, 123);
    testClass.RemoveSymbol();
    testClass.RemoveSymbol();
    testClass.RemoveSymbol();
    testClass.RemoveSymbol();
    testClass.RemoveSymbol();
    testClass.RemoveSymbol();
    testClass.RemoveSymbol();
    testClass.AddNumber(1);
    testClass.AddNumber(2);
    testClass.AddNumber(3);
    testClass.AddNumber(4);
    testClass.AddNumber(5);
    testClass.AddSeparator();
    testClass.AddNumber(1);
    testClass.AddNumber(1);
    testClass.AddNumber(1);
    testClass.AddNumber(1);
    string result = "12345/1111";
    Assert.AreEqual(result, testClass.ToString());
}

[TestMethod]
public void AddDeleteTest3()
{
    TEditor testClass = new TEditor(1234567, 12345678);
    for (int i = 0; i < 100; ++i)
        testClass.RemoveSymbol();
    for (int i = 0; i < 100; ++i)
        testClass.AddSeparator();
    testClass.AddNumber(1);
    testClass.AddNumber(1);
    testClass.AddNumber(1);
    testClass.AddNumber(1);
    string result = "1111";
    Assert.AreEqual(result, testClass.ToString());
}
```

```
    [TestMethod]
    public void AddDeleteTest4()
    {
        TEditor testClass = new TEditor("0/1");
        for (int i = 0; i < 100; ++i)
            testClass.AddNumber(i);
        string result = "123456789";
        Assert.AreEqual(result, testClass.ToString());
    }
    [TestMethod]
    public void Clear()
    {
        TEditor testClass = new TEditor("2345678/345678");
        testClass.Clear();
        string result = "0";
        Assert.AreEqual(result, testClass.ToString());
    }
}

[TestClass]
public class TMemoryTest
{
    [TestMethod]
    public void InitAndOutput1()
    {
        TFrac frac = new TFrac(22, 33);
        TMemory<TFrac> memory = new TMemory<TFrac>(frac);
        string answer = "2/3";
        Assert.AreEqual(answer, memory.FNumber.ToString());
    }
    [TestMethod]
    public void InitAndOutput2()
    {
        TFrac frac = new TFrac();
        TMemory<TFrac> memory = new TMemory<TFrac>(frac);
        string answer = "0/1";
        Assert.AreEqual(answer, memory.FNumber.ToString());
    }
    [TestMethod]
    public void InitAndOutput3()
    {
        TFrac frac = new TFrac(-1, 5);
        TMemory<TFrac> memory = new TMemory<TFrac>(frac);
        string answer = "-1/5";
        Assert.AreEqual(answer, memory.FNumber.ToString());
    }

    [TestMethod]
    public void Sum1()
    {
        TFrac frac = new TFrac(-1, 5);
        TMemory<TFrac> memory = new TMemory<TFrac>(frac);
        TFrac summator = new TFrac(1, 2);
        memory.Add(summator);
        string answer = "3/10";
        Assert.AreEqual(answer, memory.FNumber.ToString());
    }

    [TestMethod]
    public void Sum2()
    {
        TFrac frac = new TFrac(8, 9);
        TMemory<TFrac> memory = new TMemory<TFrac>(frac);
        TFrac summator = new TFrac(-16, 3);
        memory.Add(summator);
```

```
        string answer = "-40/9";
        Assert.AreEqual(answer, memory.FNumber.ToString());
    }

    [TestMethod]
    public void TestFState1()
    {
        TFrac frac = new TFrac(8, 9);
        TMemory<TFrac> memory = new TMemory<TFrac>(frac);
        memory.Clear();
        bool expected = false;
        Assert.AreEqual(expected, memory.FState);
    }

    [TestMethod]
    public void TestFState2()
    {
        TFrac frac = new TFrac(8, 9);
        TMemory<TFrac> memory = new TMemory<TFrac>(frac);
        bool expected = false;
        Assert.AreEqual(expected, memory.FState);
    }

    [TestMethod]
    public void TestFState3()
    {
        TFrac frac = new TFrac(8, 9);
        TMemory<TFrac> memory = new TMemory<TFrac>(frac);
        memory.Add(frac);
        bool expected = true;
        Assert.AreEqual(expected, memory.FState);
    }
}

[TestClass]
public class TProcTest
{
    [TestMethod]
    public void Init1()
    {
        TFrac leftFrac = new TFrac();
        TFrac rightFrac = new TFrac();
        ADT_Proc<TFrac> proc = new ADT_Proc<TFrac>(leftFrac, rightFrac);
        string answer = "0/1";
        Assert.AreEqual(answer, proc.Left_Result_operand.ToString());
        Assert.AreEqual(answer, proc.Right_operand.ToString());
    }

    [TestMethod]
    public void Init2()
    {
        TFrac leftFrac = new TFrac(11, 3);
        TFrac rightFrac = new TFrac();
        ADT_Proc<TFrac> proc = new ADT_Proc<TFrac>(leftFrac, rightFrac);
        string answer = "11/3";
        Assert.AreEqual(answer, proc.Left_Result_operand.ToString());
    }

    [TestMethod]
    public void Init3()
    {
        TFrac leftFrac = new TFrac(16, 4);
        TFrac rightFrac = new TFrac(17, 9);
        ADT_Proc<TFrac> proc = new ADT_Proc<TFrac>(leftFrac, rightFrac);
        string answer = "17/9";
```

```
    Assert.AreEqual(answer, proc.Right_operand.ToString());
}

[TestMethod]
public void Operation1()
{
    TFrac leftFrac = new TFrac(1, 2);
    TFrac rightFrac = new TFrac(1, 2);
    ADT_Proc<TFrac> proc = new ADT_Proc<TFrac>(leftFrac, rightFrac);
    proc.Operation = 1;
    proc.DoOperation();
    string answer = "1/1";
    Assert.AreEqual(answer, proc.Left_Result_operand.ToString());
}

[TestMethod]
public void Operation2()
{
    TFrac leftFrac = new TFrac(3, 4);
    TFrac rightFrac = new TFrac(5, 6);
    ADT_Proc<TFrac> proc = new ADT_Proc<TFrac>(leftFrac, rightFrac);
    proc.Operation = 2;
    proc.DoOperation();
    string answer = "-1/12";
    Assert.AreEqual(answer, proc.Left_Result_operand.ToString());
}

[TestMethod]
public void Operation3()
{
    TFrac leftFrac = new TFrac(12, 7);
    TFrac rightFrac = new TFrac(5, 9);
    ADT_Proc<TFrac> proc = new ADT_Proc<TFrac>(leftFrac, rightFrac);
    proc.Operation = 3;
    proc.DoOperation();
    string answer = "20/21";
    Assert.AreEqual(answer, proc.Left_Result_operand.ToString());
}

[TestMethod]
public void Operation4()
{
    TFrac leftFrac = new TFrac(56, 7);
    TFrac rightFrac = new TFrac(-22, 3);
    ADT_Proc<TFrac> proc = new ADT_Proc<TFrac>(leftFrac, rightFrac);
    proc.Operation = 4;
    proc.DoOperation();
    string answer = "-12/11";
    Assert.AreEqual(answer, proc.Left_Result_operand.ToString());
}

[TestMethod]
public void TestFState1()
{
    TFrac leftFrac = new TFrac(56, 7);
    TFrac rightFrac = new TFrac(-22, 3);
    ADT_Proc<TFrac> proc = new ADT_Proc<TFrac>(leftFrac, rightFrac);
    proc.DoFunction(0);
    string answer = "-3/22";
    Assert.AreEqual(answer, proc.Right_operand.ToString());
}

[TestMethod]
public void TestFState2()
{
```

```
            TFrac leftFrac = new TFrac(56, 7);
            TFrac rightFrac = new TFrac(-22, 3);
            ADT_Proc<TFrac> proc = new ADT_Proc<TFrac>(leftFrac, rightFrac);
            proc.DoFunction(1);
            string answer = "484/9";
            Assert.AreEqual(answer, proc.Right_operand.ToString());
        }
    }
}
```