Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Сибирский государственный университет телекоммуникаций и
информатики»
(СибГУТИ)

Кафедра прикладной математики и кибернетики

Современные Технологии Программирования 2

Расчетно-графическое Задание
Вариант №9

Выполнил:
Студент IV курса ИВТ,
группы ИП-713
Михеев Никита Алексеевич

Работу проверил:
Ассистент кафедры ПМиК
Агалаков Антон Александрович

Новосибирск 2020 г.

Оглавление

1. Задание

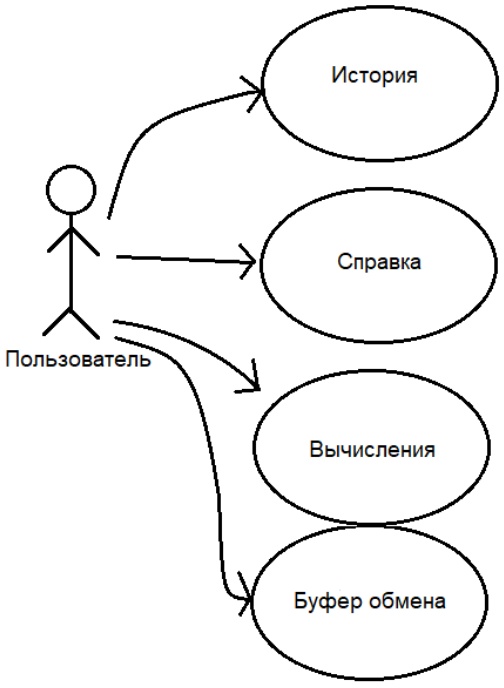
Спроектировать и реализовать калькулятор для выполнения вычислений над числами, заданными в соответствии с вариантом, используя классы C#, C++ и библиотеку визуальных компонентов для построения интерфейса.

Тип числа – «Калькулятор простых дробей».

Требования.

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.

2. 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 вида n/1;
2. Пользователь выбирает операцию (оператор);
3. Пользователь может ввести второй операнд;
4. Пользователь нажимает кнопку «=» или Enter на клавиатуре;
5. Система сохраняет вводимые данные и результат в историю

Сценарий для прецедента «История»:

1. Пользователь выбирает пункт меню «История»;
2. Открывается экран с историей, сохраненной в системе;
3. Пользователь может пролистать историю или закрыть ее.

Сценарий для прецедента «Справка»:

1. Пользователь выбирает пункт меню «Справка»;
2. Открывается экран со справкой, сохраненной в системе;
3. Пользователь может прочитать справку или закрыть ее.

Сценарий для прецедента «Буфер обмена»:

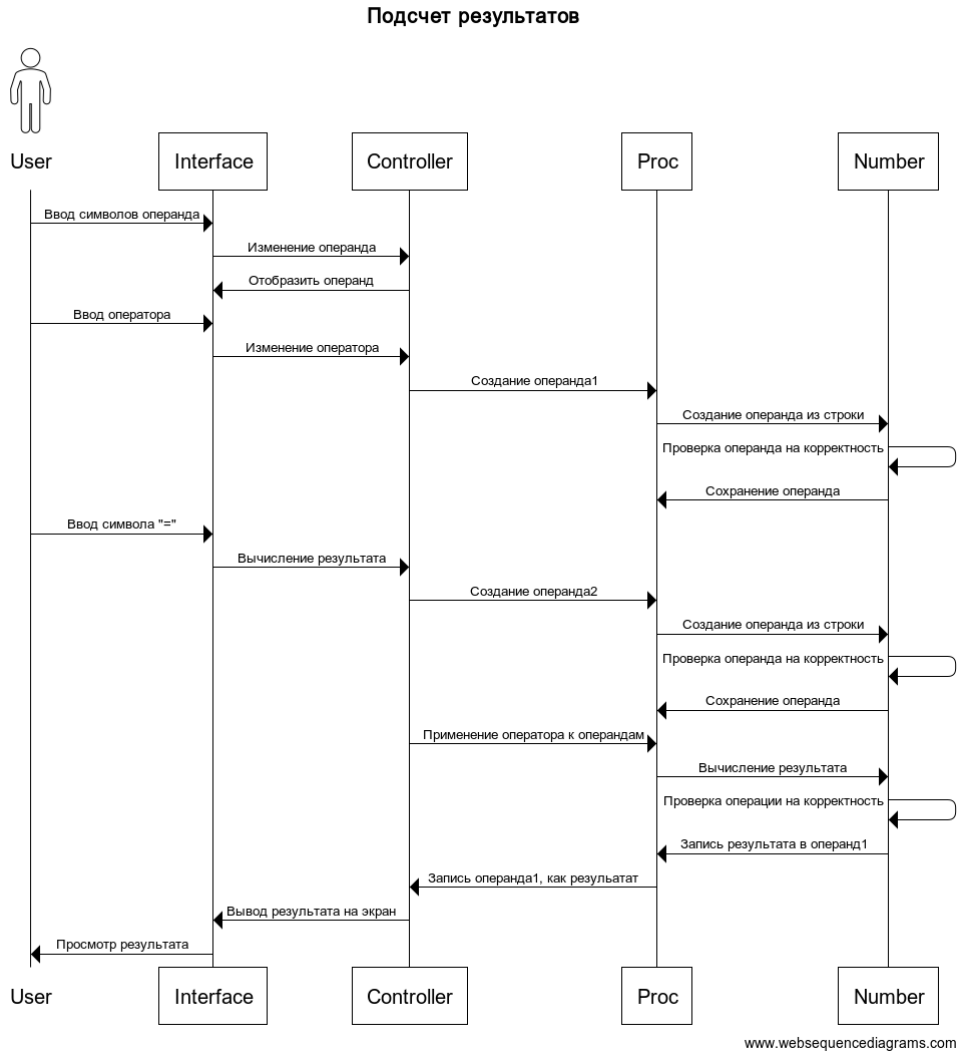
1. Пользователь вводит дробь (операнд) с символом разделителем. Если символ-разделитель не введен, то дробь будет вида n/1;
2. Пользователь выбирает пункт меню «Копировать» или комбинацию клавиш Ctrl+C;
3. Система сохраняет данные из поля для ввода в буфер обмена.

Альтернативный поток событий:

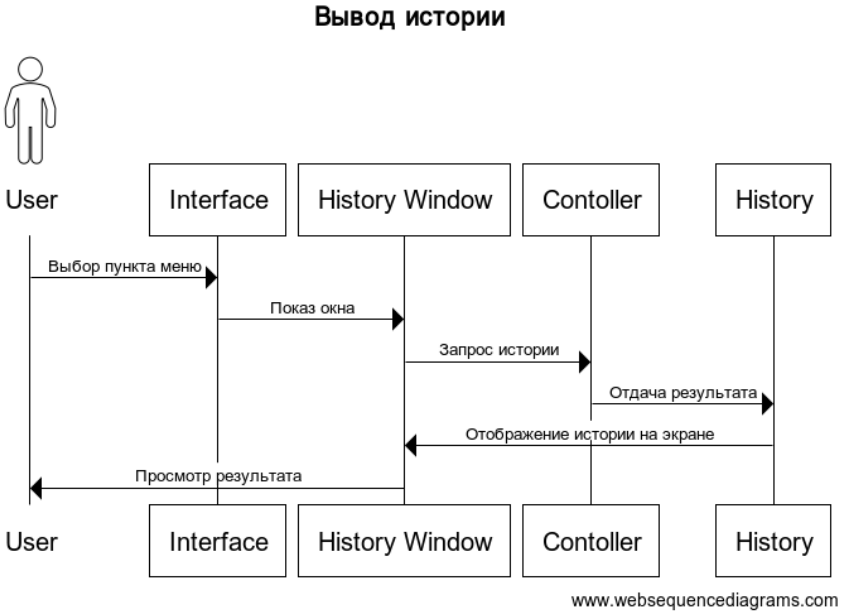
1. Пользователь выбирает пункт меню «Вставить» или комбинацию клавиш Ctrl+V;
2. Система получает из буфера обмена дробь;
3. Система выводит на экран данные;
4. Пользователь может выбрать операцию (операнд);
5. Пользователь нажимает кнопку «=» или Enter на клавиатуре.

3. Диаграмма последовательностей для прецедентов

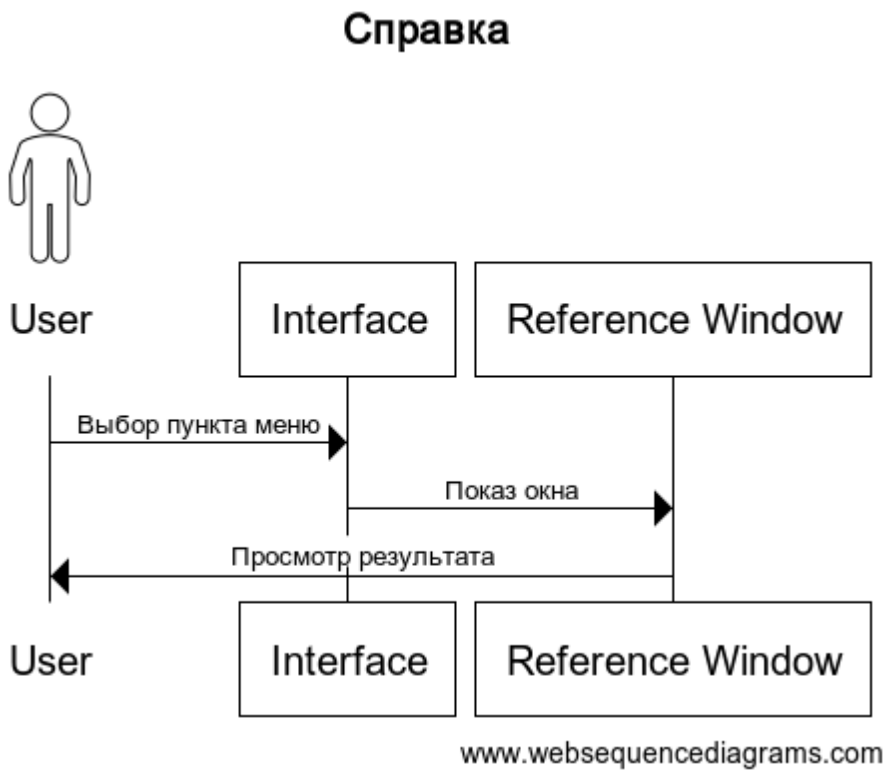
Выполнение вычислений и подсчет результатов:



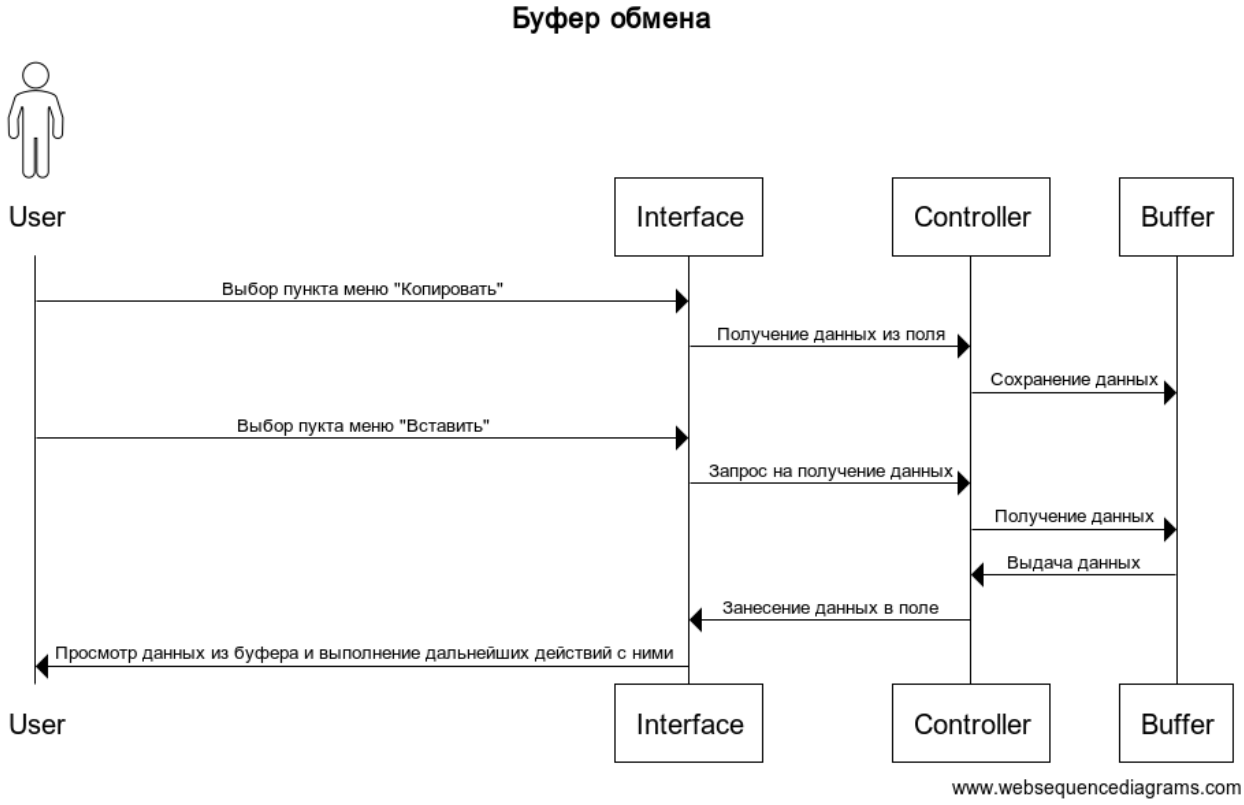
Просмотр истории:



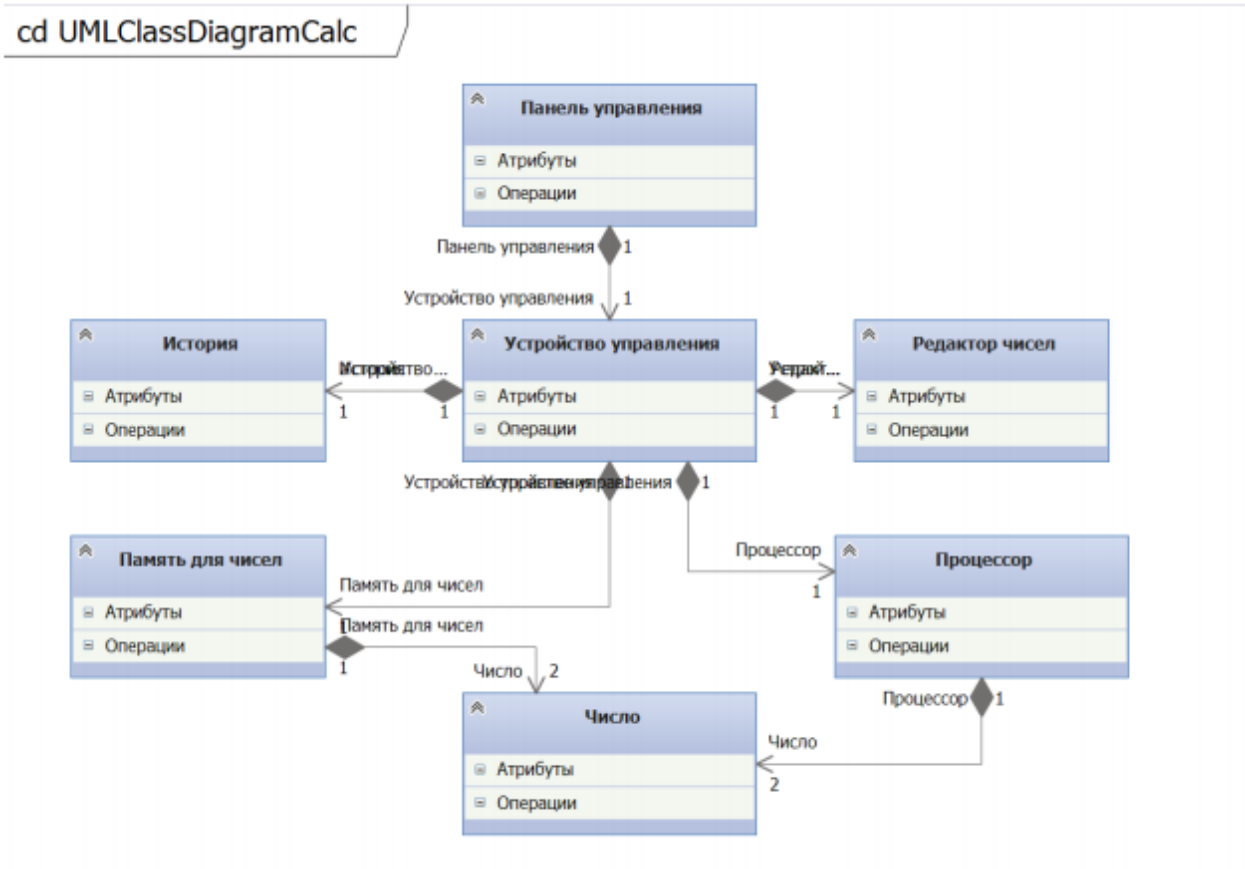
Просмотр справки:



Работа с буфером обмена:



4. Диаграмма классов UML для калькулятора



Здесь класс число в зависимости от варианта может быть: p-ичное число, простая дробь, комплексное число. У моего варианта тип числа: простая дробь.

5. Спецификации к типам данных

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.
	Простая дробь (тип TFrac). Например: c = 2/1, Копировать(c) = 2/1
Постусловия:	Нет

Сложить	
Вход:	Простая дробь d (тип TFrac).
Предусловия:	Нет

Процесс:	Создаёт и возвращает простую дробь (тип TFrac), полученную сложением самой дроби q = a1/b1 с d =a2/b2: ((a1/b1)+(a2/b2)=(a1*b2 + a2*b1)/(b1* b2)). Например: q = 1/2, d = -3/4 q.Сложить(d) = -1/4.
Выход:	Простая дробь (тип TFrac).
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 (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 (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 (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 (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 (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 (тип TFrac).
Постусловия:	Нет

Равно	
Вход:	Нет
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:	Нет
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:	
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

6. Текст программы

TFrac.cs:

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

namespace MPT_RGZ
{
    public class TFrac
    {
        private long numerator;
        public long Numerator { get; }
        private long denominator;
        public long Denominator { get; }

        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> FracParts =
frac.Split('/').ToList();
                numerator = Convert.ToInt64(FracParts[0]);
                denominator = Convert.ToInt64(FracParts[1]);
                if (denominator == 0)
                {
                    numerator = 0;
                    denominator = 1;
                    return;
                }
                long gcdResult = GCD(numerator, denominator);
                if (gcdResult > 1)
                {
                    numerator /= gcdResult;
                    denominator /= gcdResult;
                }
                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(TFrac anotherFrac)
        {
            numerator = anotherFrac.numerator;
            denominator = anotherFrac.denominator;
        }

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

        public TFrac Add(TFrac a)
        => new TFrac(numerator * a.denominator + denominator *
a.numerator, denominator * a.denominator);

        public TFrac Mul(TFrac a)
        => new TFrac(numerator * a.numerator, denominator *
a.denominator);

        public TFrac Sub(TFrac a)
        => new TFrac(numerator * a.denominator - denominator *
a.numerator, denominator * a.denominator);

        public TFrac Div(TFrac a)
        => new TFrac(numerator * a.denominator, denominator *
a.numerator);

        public TFrac Square()
        => new TFrac(numerator * numerator, denominator *
denominator);

        public TFrac Reverse() => new TFrac(denominator,
numerator);

        public TFrac Minus() => new TFrac(-numerator,
denominator);

        public bool Equal(TFrac a) => numerator == a.numerator
&& denominator == a.denominator;
```

```
        public static bool operator >(TFrac a, TFrac b) =>
(Convert.ToDouble(a.numerator) /
Convert.ToDouble(a.denominator)) >
(Convert.ToDouble(b.numerator) /
Convert.ToDouble(b.denominator));

        public static bool operator <(TFrac a, TFrac b) =>
(Convert.ToDouble(a.numerator) /
Convert.ToDouble(a.denominator)) <
(Convert.ToDouble(b.numerator) /
Convert.ToDouble(b.denominator));

        public long getNumeratorNum() => numerator;

        public long getDenominatorNum() => denominator;

        public string getNumeratorString() =>
numerator.ToString();

        public string getDenominatorString() =>
denominator.ToString();

        public override string ToString() =>
getNumeratorString() + "/" + getDenominatorString();
    }
}
```

TFracEditor.cs:

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

namespace MPT_RGZ
{
    public class TFracEditor
    {
        private string fraction;
        public string Fraction {
            get => fraction;
            set
            {
                fraction = new TFrac(value).ToString();
            }
        }

        const string ZeroFraction = "0/";
        const string Separator = "/";
        const int LeftSideOnlyLimit = 14;
        const int BothSideLimit = 22;
        public enum Command
```

```
        {
            Zero, One, Two, Three, Four, Five, Six, Seven,
Eight, Nine, Sign, Separator, BS, CE, None
        }

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

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

        public TFracEditor(string frac)
        {
            fraction = new TFrac(frac).ToString();
        }

        public bool hasZero() =>
fraction.StartsWith(ZeroFraction) || fraction == "0";

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

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

        public string AddZero()
```

```
        {
            if (fraction.Contains(Separator) &&
fraction.Last().ToString() == Separator)
                return fraction;
            if (fraction == "0" || fraction == "0/")
                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(Enum command)
        {
            switch(command)
            {
                case Command.Zero:
                    AddZero();
                    break;
                case Command.One:
                    AddNumber(1);
                    break;
                case Command.Two:
                    AddNumber(2);
                    break;
                case Command.Three:
                    AddNumber(3);
                    break;
                case Command.Four:
                    AddNumber(4);
                    break;
                case Command.Five:
                    AddNumber(5);
                    break;
                case Command.Six:
                    AddNumber(6);
```

```
                    break;
                case Command.Seven:
                    AddNumber(7);
                    break;
                case Command.Eight:
                    AddNumber(8);
                    break;
                case Command.Nine:
                    AddNumber(9);
                    break;
                case Command.Sign:
                    AddSign();
                    break;
                case Command.Separator:
                    AddSeparator();
                    break;
                case Command.BS:
                    RemoveSymbol();
                    break;
                case Command.CE:
                    Clear();
                    break;
                default:
                    break;
            }
            return fraction;
        }

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

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

TMemory.cs:

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

namespace MPT_RGZ
{
    public class TMemory<T> where T : TFrac, new()
```

```
    {
        public enum NumStates { OFF, ON }

        public enum Commands { Store, Add, Clear, Copy }

        T number;
        NumStates state;
        public T FNumber
        {
            get { state = NumStates.ON; return number; }
            set { number = value; state = NumStates.ON; }
        }
        public NumStates FState
        {
            get => state;
            set => state = value;
        }

        public TMemory()
        {
            number = new T();
            state = NumStates.OFF;
        }

        public TMemory(T num)
        {
            number = num;
            state = NumStates.OFF;
        }

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

        public void Clear()
        {
            number = new T();
            state = NumStates.OFF;
        }

        public (T, NumStates) Edit(Commands command, T
newNumber)
        {
            switch(command)
            {
                case Commands.Store:
                    state = NumStates.ON;
```

```
                    number = newNumber;
                    break;
                case Commands.Add:
                    dynamic a = number;
                    dynamic b = newNumber;
                    number = a.Add(b);
                    break;
                case Commands.Clear:
                    Clear();
                    break;
            }
            return (number, state);
        }
    }
}
```

TProc.cs:

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

namespace MPT_RGZ
{
    public class TProc<T> where T: TFrac, new()
    {
        public enum TOprtn {None, Add, Sub, Mul, Div }
        public enum TFunc {Rev, Sqr }

        T lop_res;
        T rop;
        TOprtn operation;
        public T Lop_Res { get => lop_res; set => lop_res =
value; }
        public T Rop { get => rop; set => rop = value; }
        public TOprtn Operation { get => operation; set =>
operation = value; }

        public TProc()
        {
            operation = TOprtn.None;
            lop_res = new T();
            rop = new T();
        }

        public TProc(T leftObj, T rightObj)
        {
            operation = TOprtn.None;
            lop_res = leftObj;
            rop = rightObj;
```

```
}

public void ResetProc()
{
    operation = TOprtn.None;
    T newObj = new T();
    lop_res = rop = newObj;
}

public void DoOper()
{
    try
    {
        dynamic a = lop_res;
        dynamic b = rop;
        switch (operation)
        {
            case TOprtn.Add:
                lop_res = a.Add(b);
                break;
            case TOprtn.Sub:
                lop_res = a.Sub(b);
                break;
            case TOprtn.Mul:
                lop_res = a.Mul(b);
                break;
            case TOprtn.Div:
                lop_res = a.Div(b);
                break;
        }
    }
    catch
    {
        throw new System.OverflowException();
    }
}

public void DoFunc(TFunc func)
{
    dynamic a = rop;
    switch (func)
    {
        case TFunc.Rev:
            a = a.Reverse();
            rop = (T)a;
            break;
        case TFunc.Sqr:
            a = a.Square();
            rop = (T)a;
            break;
        default:
            break;
```

```
            }
        }
    }
}
```

TCtrl.cs:

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

namespace MPT_RGZ
{
    public class TCtrl<T, TEditor>
        where T : TFrac, new()
        where TEditor: TFracEditor, new()
    {
        public enum TCtrlState { cStart, cEditing, FunDone,
cValDone, cExpDone, cOpDone, cOpChange, cError }

        TCtrlState calcState;
        TEditor editor;
        TProc<T> proc;
        TMemory<T> memory;
        TFrac number;
        public THistory history = new THistory();

        public TCtrlState CurState { get => calcState; set =>
calcState = value; }
        public TProc<T> Proc { get => proc; set => proc = value;
}
        public TMemory<T> Memory { get => memory; set => memory
= value; }
        public TEditor Edit { get => editor; set => editor =
value; }

        public TCtrl()
        {
            Edit = new TEditor();
            Proc = new TProc<T>();
            Memory = new TMemory<T>();
            CurState = TCtrlState.cStart;
        }

        public string Reset()
        {
            Edit.Clear();
            Proc.ResetProc();
            Memory.Clear();
```

```
            CurState = TCtrlState.cStart;
            return Edit.ToString();
        }

        public string ExecComandEditor(TFracEditor.Command
command)
        {
            string toReturn;
            if(CurState == TCtrlState.cExpDone)
            {
                Proc.ResetProc();
                CurState = TCtrlState.cStart;
            }
            if (CurState != TCtrlState.cStart)
                CurState = TCtrlState.cEditing;
            toReturn = Edit.Edit(command);
            T tmp = new T();
            tmp.SetString(toReturn);
            proc.Rop = tmp;
            history.AddRecord(toReturn, command.ToString());
            return toReturn;
        }

        public string ExecOperation(TProc<T>.TOprtn oper)
        {
            if (oper == TProc<T>.TOprtn.None)
                return Edit.Fraction;
            string toReturn;
            try
            {
                switch (CurState)
                {
                    case TCtrlState.cStart:
                        Proc.Lop_Res = Proc.Rop;
                        Proc.Operation = oper;
                        CurState = TCtrlState.cOpDone;
                        Edit.Clear();
                        break;
                    case TCtrlState.cEditing:
                        Proc.DoOper();
                        Proc.Operation = oper;
                        Edit.Clear();
                        CurState = TCtrlState.cOpDone;
                        break;
                    case TCtrlState.FunDone:
                        if (Proc.Operation ==
TProc<T>.TOprtn.None)
                            Proc.Lop_Res = Proc.Rop;
                        else
                            Proc.DoOper();
                        Proc.Operation = oper;
                        Edit.Clear();
```

```
                    CurState = TCtrlState.cOpChange;
                    break;
                case TCtrlState.cOpDone:
                    CurState = TCtrlState.cOpChange;
                    Edit.Clear();
                    break;
                case TCtrlState.cValDone:
                    break;
                case TCtrlState.cExpDone:
                    Proc.Operation = oper;
                    Proc.Rop = Proc.Lop_Res;
                    CurState = TCtrlState.cOpChange;
                    Edit.Clear();
                    break;
                case TCtrlState.cOpChange:
                    Proc.Operation = oper;
                    Edit.Clear();
                    break;
                case TCtrlState.cError:
                    Proc.ResetProc();
                    return "ERR";
            }
            toReturn = Proc.Lop_Res.ToString();
        }
        catch
        {
            Reset();
            return "ERROR";
        }
        history.AddRecord(toReturn, oper.ToString());
        return toReturn;
    }

    public string ExeFunction(TProc<T>.TFunc func)
    {
        string toReturn;
        try
        {
            if(CurState == TCtrlState.cExpDone)
            {
                Proc.Rop = Proc.Lop_Res;
                Proc.Operation = TProc<T>.TOprtn.None;
            }
            Proc.DoFunc(func);
            CurState = TCtrlState.FunDone;
            toReturn = Proc.Rop.ToString();
        }
        catch
        {
            Reset();
            return "ERROR";
        }
```

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

        public string Calculate()
        {
            string ToReturn;
            try
            {
                if (CurState == TCtrlState.cStart)
                    Proc.Lop_Res = Proc.Rop;
                Proc.DoOper();
                CurState = TCtrlState.cExpDone;
                ToReturn = Proc.Lop_Res.ToString();
            }
            catch
            {
                Reset();
                return "ERROR";
            }
            return ToReturn;
        }

        public (T, TMemory<T>.NumStates)
ExecCommandMemory(TMemory<T>.Commands command, string str)
        {
            T tmp = new T();
            tmp.SetString(str);
            (T, TMemory<T>.NumStates) obj = (null,
TMemory<T>.NumStates.OFF);
            try
            {
                obj = Memory.Edit(command, tmp);
            }
            catch
            {
                Reset();
                return obj;
            }
            if(command == TMemory<T>.Commands.Copy)
            {
                Edit.Fraction = obj.Item1.ToString();
                Proc.Rop = obj.Item1;
            }
            return obj;
        }
    }
}
```

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 MPT_RGZ
{
    public partial class Form1 : Form
    {
        TCtrl<TFrac, TFracEditor> fracController;
        bool fracMode = true;
        const string operations = "+-/*";
        string clipboard = string.Empty;

        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 static TFracEditor.Command
CharToEditorCommand(char ch)
        {
            TFracEditor.Command command =
TFracEditor.Command.None;
            switch (ch)
            {
                case '0':
                    command = TFracEditor.Command.Zero;
                    break;
                case '1':
                    command = TFracEditor.Command.One;
                    break;
                case '2':
                    command = TFracEditor.Command.Two;
                    break;
                case '3':
                    command = TFracEditor.Command.Three;
                    break;
                case '4':
                    command = TFracEditor.Command.Four;
                    break;
                case '5':
```

```
                    command = TFracEditor.Command.Five;
                    break;
                case '6':
                    command = TFracEditor.Command.Six;
                    break;
                case '7':
                    command = TFracEditor.Command.Seven;
                    break;
                case '8':
                    command = TFracEditor.Command.Eight;
                    break;
                case '9':
                    command = TFracEditor.Command.Nine;
                    break;
                case '.':
                    command = TFracEditor.Command.Separator;
                    break;
                case '-':
                    command = TFracEditor.Command.Sign;
                    break;
            }
            return command;
        }

        private static TProc<T>.TOprtn
CharToOperationsCommand<T>(char ch) where T : TFrac, new()
        {
            TProc<T>.TOprtn command = TProc<T>.TOprtn.None;
            switch (ch)
            {
                case '+':
                    command = TProc<T>.TOprtn.Add;
                    break;
                case '-':
                    command = TProc<T>.TOprtn.Sub;
                    break;
                case '*':
                    command = TProc<T>.TOprtn.Mul;
                    break;
                case '/':
                    command = TProc<T>.TOprtn.Div;
                    break;
            }
            return command;
        }

        private static TFracEditor.Command
KeyCodeToEditorCommand(Keys ch)
        {
            TFracEditor.Command command =
TFracEditor.Command.None;
            switch (ch)
```

```
            {
                case Keys.Back:
                    command = TFracEditor.Command.BS;
                    break;
                case Keys.Delete:
                case Keys.Escape:
                    command = TFracEditor.Command.CE;
                    break;
            }
            return command;
        }

        public Form1()
        {
            fracController = new TCtrl<TFrac, TFracEditor>();
            InitializeComponent();
        }

        public void Button_Number_Edit(object sender, EventArgs
e)
        {
            Button button = (Button)sender;
            string fulltag = button.Tag.ToString();
            Enum.TryParse(fulltag, out TFracEditor.Command
ParsedEnum);
            textBox1.Text =
fracController.ExecComandEditor(ParsedEnum);
        }

        public void Button_Number_Operation(object sender,
EventArgs e)
        {
            Button button = (Button)sender;
            string fulltag = button.Tag.ToString();
            Enum.TryParse(fulltag, out TProc<TFrac>.TOprtn
ParsedEnum);
            textBox1.Text =
NumberBeautifier(fracController.ExecOperation(ParsedEnum));
        }

        public void Button_Number_Function(object sender,
EventArgs e)
        {
            Button button = (Button)sender;
            string fulltag = button.Tag.ToString();
            Enum.TryParse(fulltag, out TProc<TFrac>.TFunc
ParsedEnum);
            textBox1.Text =
NumberBeautifier(fracController.ExeFunction(ParsedEnum));
        }

        private void Button_Reset(object sender, EventArgs e)
```

```
        {
            Button button = (Button)sender;
            string fulltag = button.Tag.ToString();
            textBox1.Text = fracController.Reset();
            label1.Text = string.Empty;
        }

        private void Button_FinishEval(object sender, EventArgs
e)
        {
            Button button = (Button)sender;
            string FullTag = button.Tag.ToString();
            textBox1.Text =
NumberBeautifier(fracController.Calculate());
        }

        private void Button_Memory(object sender, EventArgs e)
        {
            Button button = (Button)sender;
            string FullTag = button.Tag.ToString();
            Enum.TryParse(FullTag, out TMemory<TFrac>.Commands
ParsedEnum);
            dynamic exec =
fracController.ExecCommandMemory(ParsedEnum, textBox1.Text);
            if (ParsedEnum == TMemory<TFrac>.Commands.Copy)
                textBox1.Text = exec.Item1.ToString();
            label1.Text = exec.Item2 ==
TMemory<TFrac>.NumStates.ON ? "M" : string.Empty;
        }

        private void Form1_KeyDown(object sender, KeyEventArgs
e)
        {
            if (e.KeyCode == Keys.Enter)
                button26.PerformClick();
            else
            {
                TFracEditor.Command comm =
KeyCodeToEditorCommand(e.KeyCode);
                if (comm != TFracEditor.Command.None)
                    textBox1.Text =
fracController.ExecComandEditor(comm);
            }
        }

        private void Form1_KeyPress(object sender,
KeyPressEventArgs e)
        {
            if (e.KeyChar >= '0' && e.KeyChar <= '9' ||
e.KeyChar == '.')
                textBox1.Text =
fracController.ExecComandEditor(CharToEditorCommand(e.KeyChar));
```

```
            else if (operations.Contains(e.KeyChar))
                textBox1.Text =
NumberBeautifier(fracController.ExecOperation(CharToOperationsCo
mmand<TFrac>(e.KeyChar)));
        }

        private void дробьToolStripMenuItem_Click(object sender,
EventArgs e)
        {
            дробьToolStripMenuItem.Checked = true;
            числоToolStripMenuItem.Checked = false;
            fracMode = true;
        }

        private void числоToolStripMenuItem_Click(object sender,
EventArgs e)
        {
            дробьToolStripMenuItem.Checked = false;
            числоToolStripMenuItem.Checked = true;
            fracMode = false;
        }

        private void копироватьToolStripMenuItem_Click(object
sender, EventArgs e)
        {
            clipboard = textBox1.Text;
        }

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

        private void справкаToolStripMenuItem1_Click(object
sender, EventArgs e)
        {
            MessageBox.Show("Калькулятор дробных чисел.\n" +
                "Разработал: Михеев Н.А.\n" +
                "Группа: ИП-713.",
                "Справка",
```

```
                MessageBoxButtons.OK,
                MessageBoxIcon.Information);
        }

        private void toolStripMenuItem1_Click(object sender,
EventArgs e)
        {
            Form2 history = new Form2();
            history.Show();
            if (fracController.history.Count() == 0)
            {
                MessageBox.Show("История пуста", "Внимание",
MessageBoxButtons.OK, MessageBoxIcon.Warning);
                return;
            }
            for (int i = 0; i < fracController.history.Count();
i++)
            {
                List<string> currentRecord =
fracController.history[i].ToList();
                history.dataGridView1.Rows.Add(currentRecord[0],
currentRecord[1]);
            }
        }
    }
}
```

THistory.cs:

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

namespace MPT_RGZ
{
    public class THistory
    {
        public struct Record
        {
            private string oper;
            private string str_res;
            public Record(string oper, string str_res)
            {
                this.oper = oper;
                this.str_res = str_res;
            }
            public List<string> ToList()
            {
                return new List<string> { oper, str_res };
            }
```

```
        }

        List<Record> L;
        public THistory()
        {
            L = new List<Record>();
        }

        public void AddRecord(string o, string r)
        {
            Record record = new Record(o, r);
            L.Add(record);
        }

        public Record this[int i]
        {
            get
            {
                if (i < 0 || i >= L.Count)
                    throw new IndexOutOfRangeException();
                return L[i];
            }
            set
            {
                if (i < 0 || i >= L.Count)
                    throw new IndexOutOfRangeException();
                L[i] = value;
            }
        }

        public void Clear()
        {
            L.Clear();
        }

        public int Count()
        {
            return L.Count();
        }
    }
}
```

7. Тестовые наборы данных для тестирования абстрактных типов данных, классов и приложения

Тесты для класса TFrac.cs:

Метод	Входные данные	Ожидаемый результат
Constructor	“1/2”	“1/2”
Constructor	“111/2”	“111/2”
Constructor	“1/999”	“1/999”

Constructor	“100/10”	“10/1”
Constructor	“-100/60”	“-5/3”
Constructor	“00000003/000004”	“3/4”
Constructor	2,4	“1/2”
Constructor	17,3	“17/3”
Constructor	100, 100	“1/1”
Constructor	-100, -99	“100/99”
Constructor	0,0	“0/1”
Constructor	100, -5	“-20/1”
Copy	1,4	“1/4”
Copy	100, -5	“-20/1”
Add	1,2; -3,4	“-1/4”
Add	-1,2; -1,2	“-1/1”
Add	-6,2; 6,2	“0/1”
Add	50,3; 0,1	“50/3”
Mul	-1,2; -1,2	“1/4”
Mul	1,6; 0,1	“0/1”
Mul	1,6; 1,6	“1/36”
Mul	-1,6; 12,1	“-2/1”
Sub	0,1;1,1	“-1/1”
Sub	5,1; 1,1	“4/1”
Sub	1,2; 1,2	“0/1”
Sub	-1,6; -1,6	“0/1”
Div	5,6; 1,1	“5/6”
Div	1,1; 5,6	“6/5”
Div	0,1; 5,6	“0/1”
Div	2,3; 7,4	“8/21”
Div	2,3; 2,3	“1/1”
Div	-1,3; -9,5	“5/27”
Reverse	-2,3	“-3/2”
Reverse	0,1	“0/1”
Reverse	5,6	“6/5”
Square	1,3	“1/9”
Square	0,1	“0/1”
Square	-2, 3	“4/9”
Equal	1,3; 1,3	True
Equal	0,6; 1,6	False
Equal	-1,6; -1,6	True
Equal	-1,7; 1,7	False
Equal	1,6; 0,1	False

Greater	1,6; 0,1	True
Greater	0,1; 0,1	False
Greater	-1,6; 0,1	False
Greater	17,3; 16,3	True
Greater	-2,3; -1,3	False

Тестовые наборы для класса TFracEditor.cs:

Метод	Входные данные	Ожидаемый результат
Init	“3/4”	“3/4”
Init	“-16/3”	“-16/3”
Init	“0/8”	“0/1”
Init	“-17/4”	“-17/4”
Init	“0/1”	“0/1”
Init	“666/6666”	“111/1111”
Init	“aaaa”	“0/1”
Init	“0/1	“0/1”
Init	“16/0000000”	“0/1”
hasZero	“14/3”	False
hasZero	“16/000000”	True
AddSign	“-14/3”	“14/3”
AddSign	“14/3”	“-14/3”
AddDelete	“123/123”	“-1/1013”
AddDelete	123,123	“12345/1111”
AddDelete	1234567,12345678	“0/1111”
AddDelete	“0/1”	“0/1012456789”
Clear	“2345678/345678	“0”

Тестовые наборы для TMemory.cs:

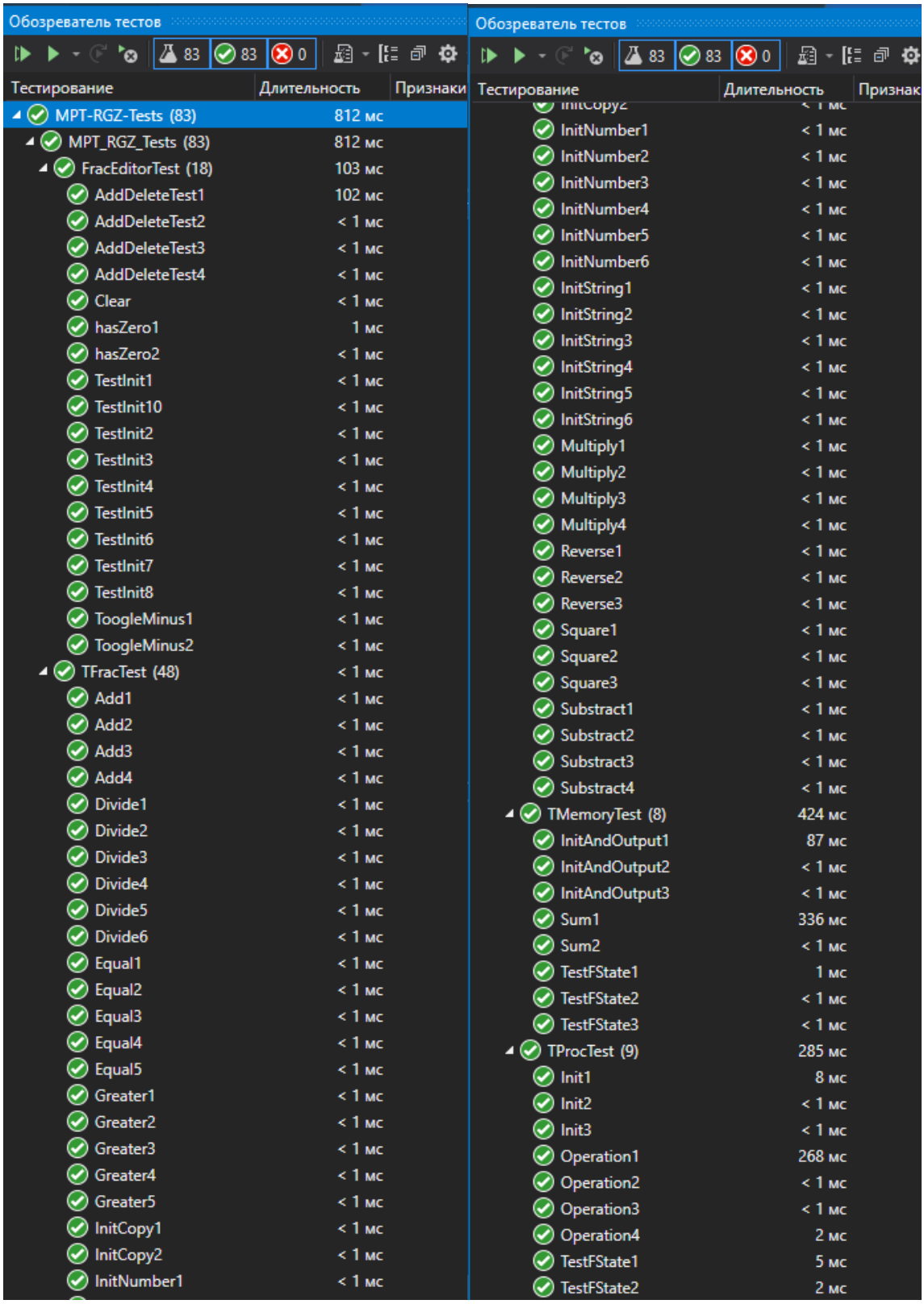
Метод	Входные данные	Ожидаемый результат
Init	22,33	“2/3”
Init	“”	“0/1”
Init	-1,5	“-1,5”
Sum	-1,5; 1,2	“3/10”
Sum	8,9; -16,3	“-40/9”
FState	8,9	OFF
FState	8,9	OFF
FState	8,9	ON

Тестовые наборы для TProc.cs:

Метод	Входные данные	Ожидаемый результат
Init	“”; “”	“0/1”
Init	11,3; “”	“11/3”

Init	16,4; 17,9	“17/9”
DoOper	1,2; 1,2	“1/1”
DoOper	3,4; 5,6	“-1/12”
DoOper	12,7; 5,9	“20/21”
DoOper	56,7; -22,3	“-12/11”
FState	56,7; -22,3	“-3/22”
FState	57,7; -22,3	“484/9”

Все тесты были запущены и успешно пройдены, проблем не обнаружено:



8. Инструкция пользователю

Пользователю необходимо для начала выбрать режим работы калькулятора: числа или дроби. Затем ввести с помощью клавиатуры или экранных клавиш число в формате a/b, используя знак разделения или, если выбран режим число, ввести a.

После ввода числа пользователь может выбрать операцию или функцию, которую он хочет совершить с числом.

После выбора операции (/, *, -, +) необходимо ввести второе чисто по принципу первого, или нажать клавишу «=» на экране или Enter на клавиатуре, таким образом операция будет произведена над первым введенным числом и его копией.

Если же выбрана была функция (клавиши Sqr, 1/x, +/-), она сразу будет выполнена над введённым числом.

Для повторного выполнения операции или функции над числом можно нажать «=» или Enter повторно.

У калькулятора так же присутствует память, с помощью кнопок MS – сохраняется число с экрана в память, MR – чисто из памяти помещается на экран, MC – очищается память, M+ - прибавляется число их памяти к текущему числу на экране.

С помощью клавиш C – можно полностью сбросить калькулятор до начального состояния, CE – очистить введенный оператор, не сбрасывая память, BS – стереть крайний введенный символ.

С помощью меню правка можно занести число с экрана в буфер-обмена или вставить из буфера. Настройка – выбор режима работы калькулятора (число, дробь). Справка – информация о программе и авторе. История – история введенных символов и операций.

9. Литература

1. Уильям Топ, Уильям Форд. Пол. Структуры данных в C++:Пер. с англ.- М.: ЗАО «Издательство БИНОМ», 2000г. - 816 с., ил.

2. А. Пол. Объектно-ориентированное программирование на C++, 2- е изд./Пер. с англ.-СПб.;М: «Невский диалект» - «Издательство БИНОМ», 1999г. - 462 с., ил.

3. C/C++, Программирование на языке высокого уровня /Т.А. Павловская - СПб.;Питер, 2002г. - 464 с.: ил.

4. Visual C++ на примерах /Г.Ф. Довбуш, А.Д. Хомоненко / Под ред. проф. А.Д. Хомоненко. -СПб.; БХВ-Петербург, 2008г. - 528 с.: ил.

5. Р. Лафоре. Объектно-ориентированное программирование в C++. Классика в Computer Science. 4-е изд. -СПб.;Питер, 2005г. - 924 с.: ил.

6. Холингвэрт, Джарод, Баттерфилд, Дэн,Сворт, Боб и др. C++ Builder 5. Руководство разработчика, том 1. Основы: Пер. с англ..:Уч. пос. М.: Издательский дом «Вильямс», 2001. - 880 с.: ил. – Парал. Тит. англ.