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 7
«Калькулятор р-ичных чисел»

Выполнил:

Студент гр. ИП-013 _____/ Копытина Т.А. /
ФИО студента

«__» _____ 2024 г.

Проверил:

Старший преподаватель
кафедры ПМиК _____/ Агалаков А.А. /
ФИО преподавателя

«__» _____ 2024 г. Оценка_____

Новосибирск 2024

Содержание

Цель **3**
Задание **4**
Общие требования 4
Требования к варианту 6
Варианты выполнения 9
Методические указания к выполнению **10**
Диаграмма прецедентов UML. Сценарии прецедентов 10
Диаграмма последовательностей для прецедентов 12
Диаграмма классов для прецедентов 14
Спецификации к типам данных 15
Результаты работы программы **20**
Результат работы тестов **23**
Вывод **24**
Листинг **25**

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 р-ичных чисел».

Требования.

1. Калькулятор обеспечивает работу с числами в системах счисления с основанием в диапазоне от 2 до 16.
2. Основание системы счисления – настраиваемый параметр. Настройку можно установить в основном окне или добавить в меню «Настройка».

3. Исходные числа и результат вводятся и выводятся в формате фиксированная точка [-][< p - ичная дробь без знака >] Необходимо обеспечить возможность работы в режимах:

- «целые» (вводятся только p-ичные целые числа),
- «действительные» (вводятся p-ичные числа с целой и дробной частями).

4. Кнопки для ввода цифровой информации необходимо связать с используемой системой счисления. Для пользователя необходимо сделать доступными кнопки только для ввода цифр используемой системы счисления.
5. При смене системы счисления отображаемое число должно выражаться в новой системе счисления.

Необходимо предусмотреть следующие варианты (прецеденты) использования калькулятора:

1. Выполнение одиночных операций:

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

Пример. 5 + 2 = 7 (p = 10)

2. Выполнение операций с одним операндом:

«операнд» «операция» «=» «результат»

Пример. 5 * = 25 (p = 10)

3. Повторное выполнение последней операции:

«=»«результат» «=» «результат»

Пример. 5 + 4 = 9 = 13 = 17(p = 10)

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

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

Пример. 2 + 3 = 5 = 8 + = 16(p = 10)

5. Вычисление функций:

«операнд» «Sqr» «результат»

Пример. 5 «Sqr» 25 (p = 10)

6. Вычисление выражений:

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

Пример.

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

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		
7	Р-ичное число	1-6	да	нет	да	нет

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».

Диаграмма последовательностей для прецедентов

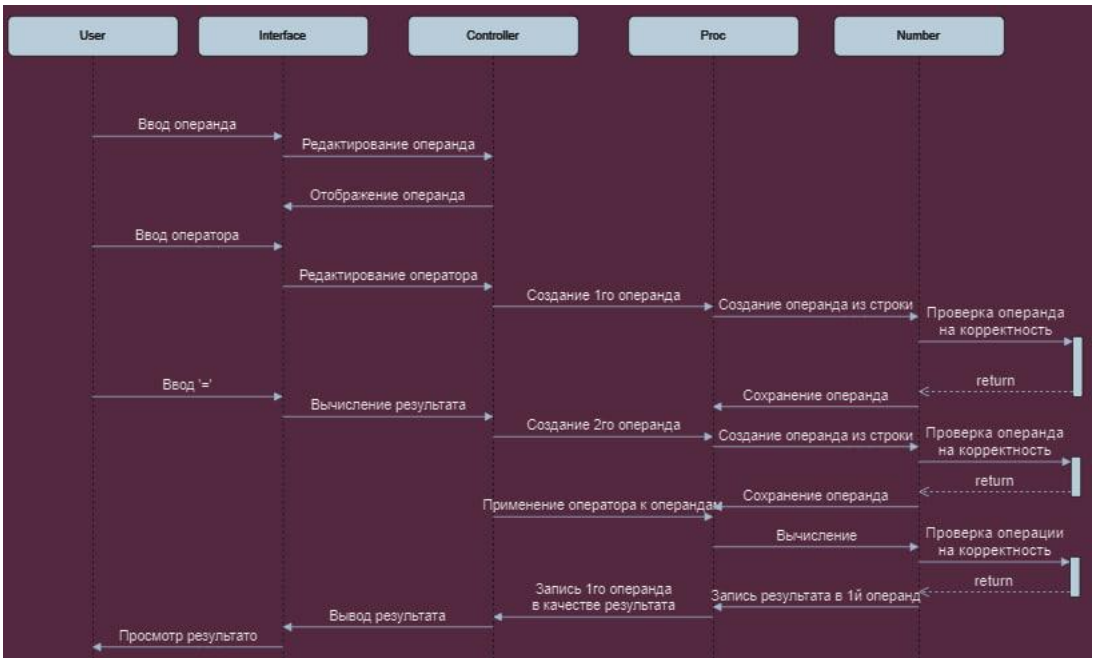

Рис 1. Вычисление выражений и подсчет результатов

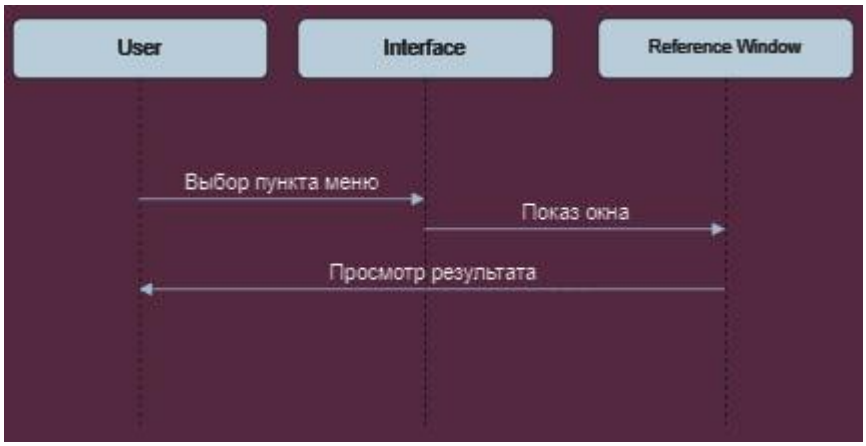

Рис 2. Просмотр справки

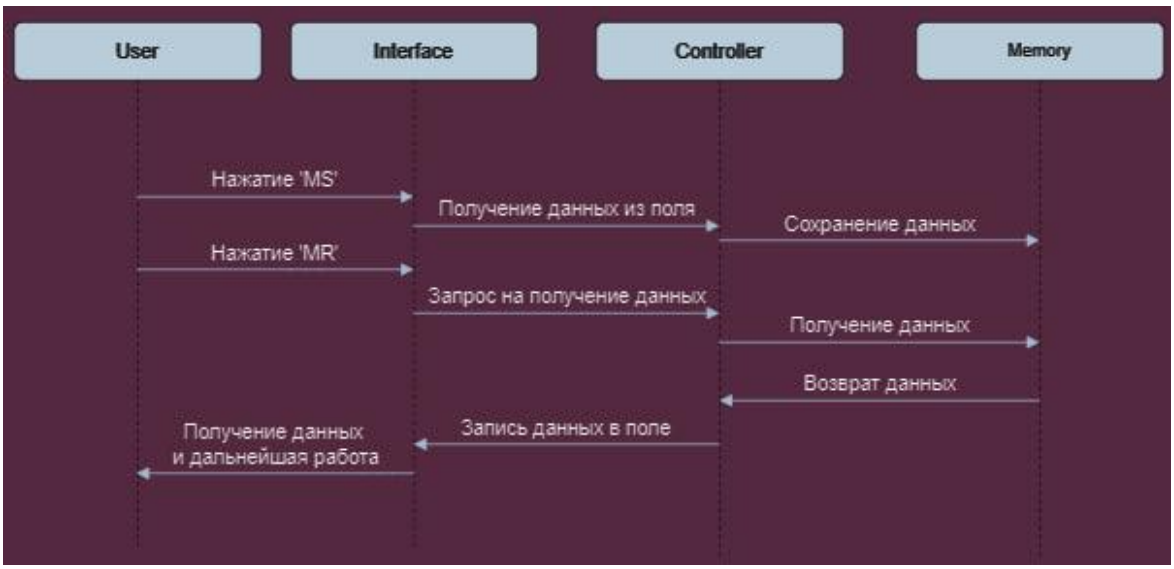

Рис 3. Работа с памятью

Рис 4. Использование настроек

Диаграмма классов для прецедентов

Рис 5. Диаграмма классов

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

Спецификации к типам данных

Спецификация типа данных «p-ичные числа».

ADT TPNumber

Данные

P-ичное число TPNumber - это действительное число (n) со знаком в системе счисления с основанием (b) (в диапазоне 2..16), содержащее целую и дробную части. Точность представления числа – (c >= 0). P-ичные числа неизменяемые.

Операции

Операции могут вызываться только объектом p-ичное число (тип TPNumber), указатель на который в них передаётся по умолчанию. При описании операций этот объект называется this «само число».

КонструкторЧисло	
Вход:	Вещественное число (a), основание системы счисления (b), точность представления числа (c)
Предусловия:	Основание системы счисления (b) должно принадлежать интервалу [2..16], точность представления числа c >= 0.
Процесс:	Инициализирует поля объекта this p-ичное число: система счисления (b), точность представления (c). В поле (n) числа заносится (a). Например: TPNumber(a,3,3) = число a в системе счисления 3 с тремя разрядами после троичной точки. TPNumber (a,3,2) = число a в системе счисления 3 с двумя разрядами после троичной точки.
Постусловия:	Объект инициализирован начальными значениями.
Выход:	Нет.
КонструкторСтрока	
Вход:	Строковые представления: p–ичного числа (a), основания системы счисления (b), точности представления числа (c).

Предусловия:	Основание системы счисления (b) должно принадлежать интервалу [2..16], точность представления числа c >= 0
Процесс:	Инициализирует поля объекта this p-ичное число: основание системы счисления (b), точностью представления (c). В поле (n) числа this заносится результат преобразования строки (a) в числовое представление. b-ичное число (a) и основание системы счисления (b) представлены в формате строки. Например: TPNumber (“20”,”3”,”6”) = 20 в системе счисления 3, точность 6 знаков после запятой.
Постусловия:	Объект инициализирован начальными значениями.
Выход:	Нет.
Копировать	
Вход:	Нет.
Предусловия:	Нет.
Процесс:	Создаёт копию самого числа this (тип TPNumber).
Выход:	p-ичное число.
Постусловия:	Нет.
Умножить	
Вход:	P-ичное число d с основанием и точностью такими же, как у самого числа this.
Предусловия:	Нет.
Процесс:	Создаёт и возвращает p-ичное число (тип TPNumber), полученное умножением полей (n) самого числа this и числа d.
Выход:	P-ичное число (тип TPNumber).
Постусловия:	Нет.
Вычесть	
Вход:	P-ичное число d с основанием и точностью такими же, как у самого числа this.

Предусловия:	Нет.
Процесс:	Создаёт и возвращает p-ичное число (тип TPNumber), полученное вычитанием полей (n) самого числа this и числа d.
Выход:	P-ичное число (тип TPNumber).
Постусловия:	Нет.
Делить	
Вход:	P-ичное число d с основанием и точностью такими же, как у самого числа.
Предусловия:	Поле (n) числа (d) не равно 0.
Процесс:	Создаёт и возвращает p-ичное число (тип TPNumber), полученное делением полей (n) самого числа this на поле (n) числа d.
Выход:	P-ичное число (тип TPNumber).
Постусловия:	Нет.
Квадрат	
Вход:	Нет.
Предусловия:	Нет.
Процесс:	Создаёт p-ичное число, в поле (n) которого заносится значение, полученное как квадрат поля (n) самого числа this.
Выход:	P-ичное число (тип TPNumber).
Постусловия:	Нет.
Обратное	
Вход:	Нет.
Предусловия:	Поле (n) самого числа не равно 0.
Процесс:	Создаёт p-ичное число, в поле (n) которого заносится значение, полученное как 1/(n) самого числа this.
Выход:	P-ичное число (тип TPNumber).

Постусловия:	Нет.
ВзятьОснованиеЧисло	
Вход:	Нет.
Предусловия:	Нет.
Процесс:	Возвращает значение поля (b) самого числа this.
Выход:	Целочисленное значение.
Постусловия:	Нет.
ВзятьТочностьЧисло	
Вход:	Нет.
Предусловия:	Нет.
Процесс:	Возвращает значение поля (c) самого числа this.
Выход:	Целочисленное значение.
Постусловия:	Нет.
ВзятьОснованиеСтрока	
Вход:	Нет.
Предусловия:	Нет.
Процесс:	Возвращает значение поля (b) самого числа this в формате строки, изображающей (b) в десятичной системе счисления.
Выход:	Строка.
Постусловия:	Нет.
ВзятьЗнаменательСтрока	
Вход:	Нет.
Предусловия:	Нет.
Процесс:	Возвращает значение числителя дроби в строковом формате.

Выход:	Строка.
Постусловия:	Нет.
ВзятьТочностьСтрока	
Вход:	Нет.
Предусловия:	Нет.
Процесс:	Возвращает значение поля (c) самого числа this в формате строки, изображающей (c) в десятичной системе счисления
Выход:	Строка.
Постусловия:	Нет.

end TPNumber

Результаты работы программы

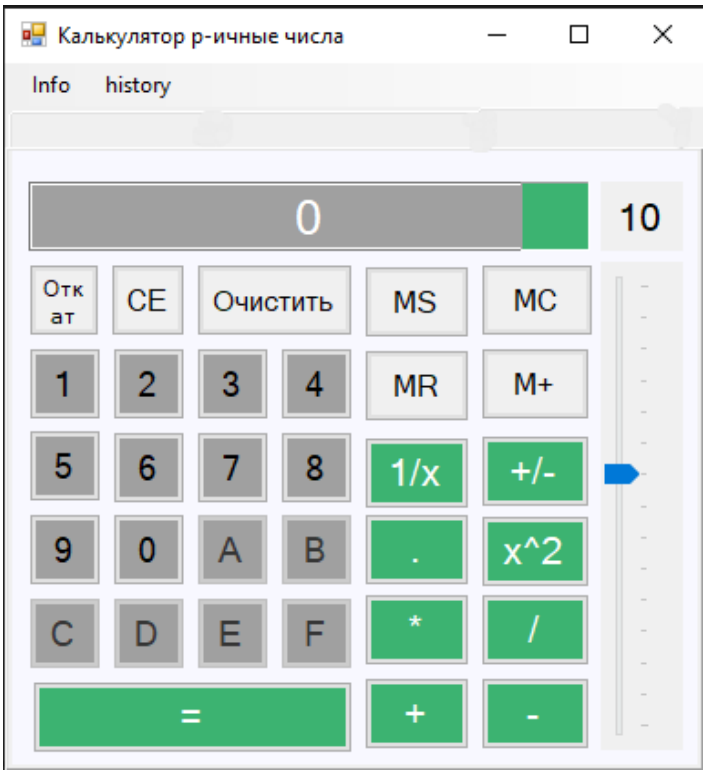

Рис. 6 - Главное окно программы

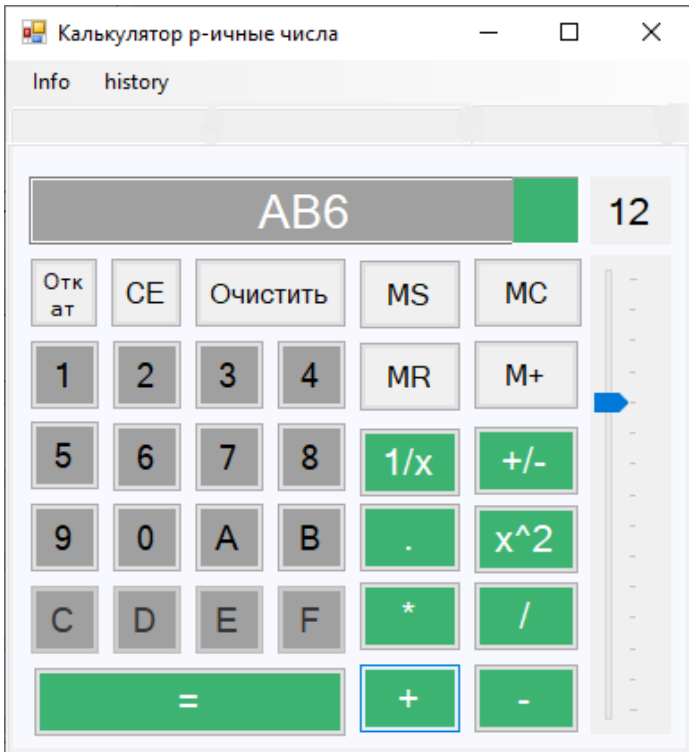

Рис. 7 – Введено число “AB6” в 12-ичной СС и нажато “+” (сложение)

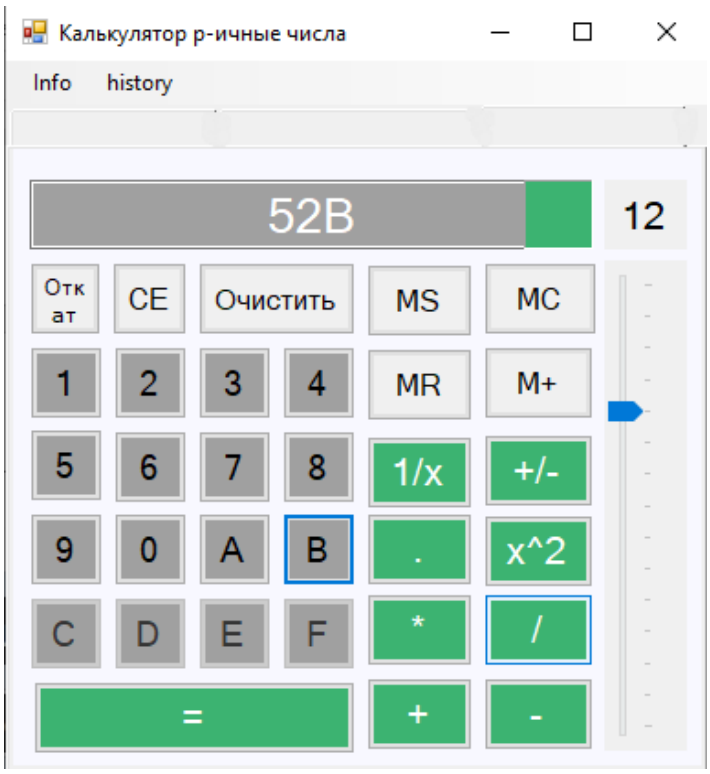

Рис. 8 – Введено число “52B” в 12-ичной СС

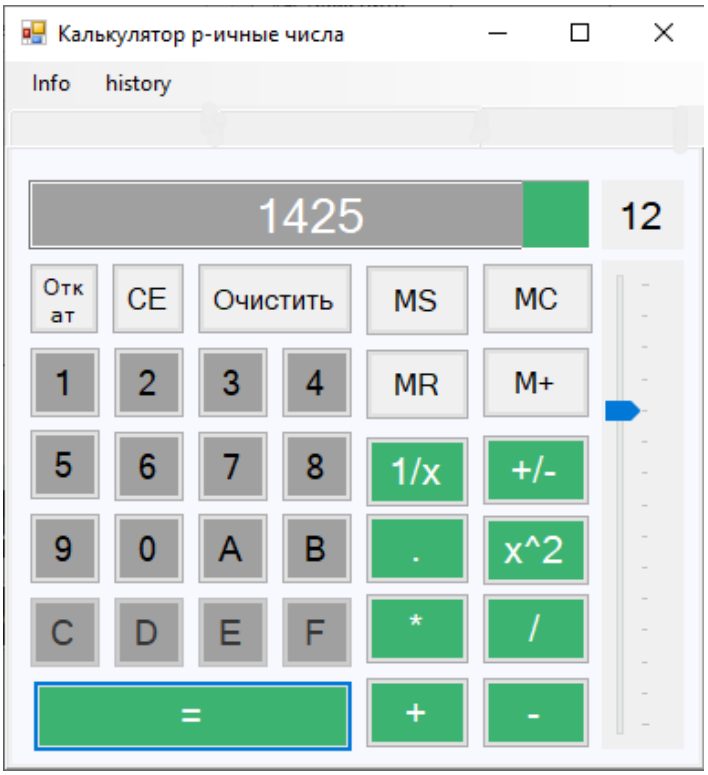

Рис 9. – Результат сложения при нажатии на “=” (показ результата операций)

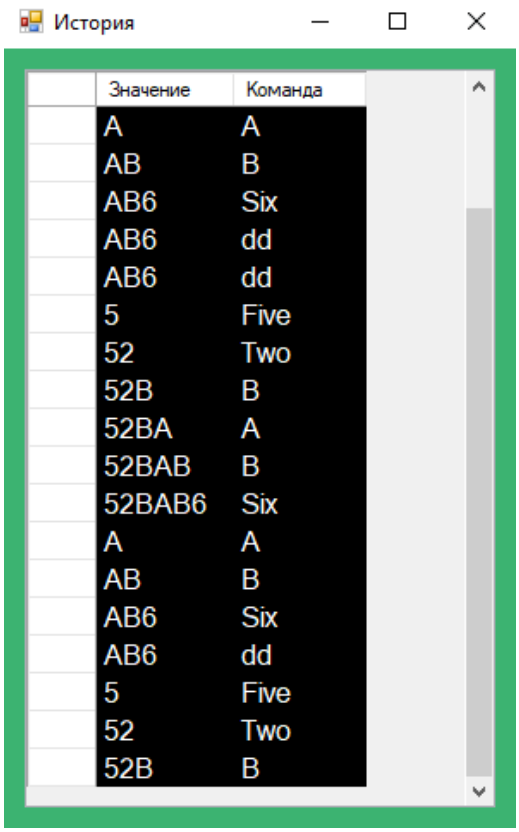

Рис 10. – История произведенных операций

Результат работы тестов

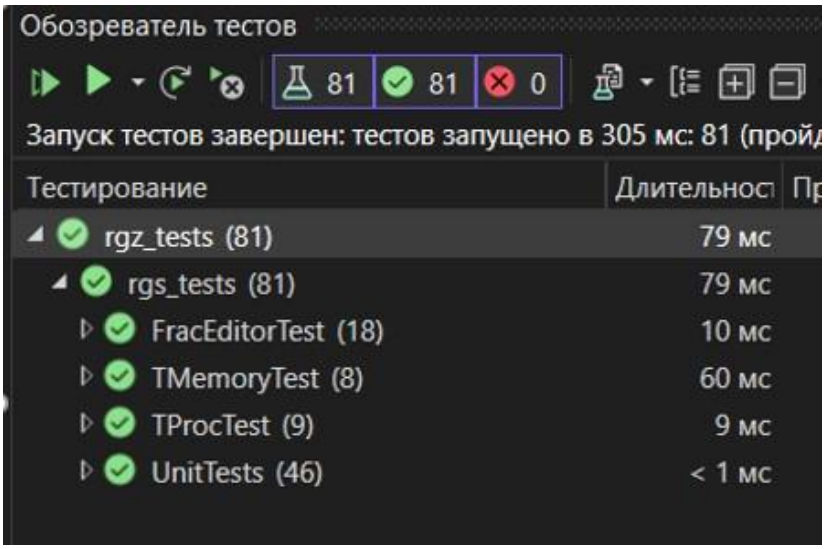

Рис 11. – Результат тестирования программы

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

Program.cs:

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

namespace calculator
{
    static class Program
    {
        [STAThread]
        static void Main()
        {

Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDe
fault(false);
            Application.Run(new
Form1());
        }
    }
}
```

ADT_Proc.cs:

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

namespace calculator
{
    public class ADT_Proc<T> where T :
ANumber, new()
    {
        public enum Operations
        {
            None, Add, Sub, Mul, Div
        }
        public enum Functions
        {
            Rev, Sqr
        }

        T left_result_operand;
        T right_operand;
        Operations 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 Operations Operation
        {
            get
            {
                return operation;
            }

            set
            {
                operation = value;
            }
        }

        public ADT_Proc()
        {
            operation = Operations.None;
            left_result_operand = new
T();
            right_operand = new T();
        }

        public ADT_Proc(T leftObj, T
rightObj)
        {
            operation = Operations.None;
            left_result_operand =
leftObj;
            right_operand = rightObj;
        }

        public void ResetProc()
        {
            operation = Operations.None;
            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 Operations.Add:

left_result_operand = a.Add(b);
                        break;
                    case Operations.Sub:
```

```
left_result_operand = a.Sub(b);
                        break;
                    case Operations.Mul:

left_result_operand = a.Mul(b);
                        break;
                    case Operations.Div:

left_result_operand = a.Div(b);
                        break;
                    default:

left_result_operand = right_operand;
                        break;
                }
            }
            catch
            {
                throw new
System.OverflowException();
            }
        }

        public void DoFunction(Functions
function)
        {
            dynamic a = right_operand;
            switch (function)
            {
                case Functions.Rev:
                    a = a.Reverse();
                    right_operand =
(T)a;
                    break;
                case Functions.Sqr:
                    a = a.Square();
                    right_operand =
(T)a;
                    break;
                default:
                    break;
            }
        }
    }
}
```

TPNumberEditor.cs

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

namespace calculator
{
    public class TPNumberEditor :
AEditor
    {
        private string pNumber;

        public int Notation;
        public int Precision;

        Regex ZeroPNumber = new
Regex("^-?(0+|.?0+|0+.(0+)?)$");
        const string Separator = ".";

        public override string Number
        {
            get
            {
                return pNumber;
            }

            set
            {
                pNumber = new
TPNumber(value, Notation,
Precision).ToString();
            }
        }
        public TPNumberEditor()
        {
            pNumber = "0";
            Notation = 10;
            Precision = 5;
        }

        public TPNumberEditor(double
num, int not, int pre)
        {
            if (not < 2 || not > 16 ||
pre < 0 || pre > 10)
            {
                pNumber = "0";
                Notation = 10;
                Precision = 5;
            }
            else
            {
                Notation = not;
                Precision = pre;
                pNumber =
TPNumber.ADT_Convert_10_p.Do(num, not,
pre);
            }
        }

        public override bool IsZero()
        {
            return
ZeroPNumber.IsMatch(pNumber);
        }
        public override string
ToggleMinus()
        {
            if (pNumber.ElementAt(0) ==
'-')
                pNumber =
pNumber.Remove(0, 1);
            else
                pNumber = "-" + pNumber;
            return pNumber;
        }

        public override string
AddNumber(int num)
```

```
        {
            if (num < 0 || num >=
Notation)
                return pNumber;
            if (num == 0)
                AddZero();
            else if (pNumber == "0" ||
pNumber == "-0")
                pNumber =
pNumber.First() == '-' ? "-" +
TPNumber.ADT_Convert_10_p.Int_to_char(nu
m).ToString() :
TPNumber.ADT_Convert_10_p.Int_to_char(nu
m).ToString();
            else
                pNumber +=
TPNumber.ADT_Convert_10_p.Int_to_char(nu
m).ToString();
            return pNumber;
        }

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

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

        public override string AddZero()
        {
            if
(pNumber.Contains(Separator) &&
pNumber.Last().ToString() == Separator)
                return pNumber;
            if (pNumber == "0" ||
pNumber == "0.")
                return pNumber;
            pNumber += "0";
            return pNumber;
        }
        public override string
ToString()
        {
            return pNumber;
        }
        public override string Clear()
        {
            pNumber = "0";
            return pNumber;
        }

        public override string
Edit(Command command)
        {
            switch (command)
            {
                case Command.cZero:
                    AddZero();
                    break;
                case Command.cOne:
                    AddNumber(1);
                    break;
                case Command.cTwo:
                    AddNumber(2);
                    break;
                case Command.cThree:
                    AddNumber(3);
                    break;
                case Command.cFour:
                    AddNumber(4);
                    break;
                case Command.cFive:
                    AddNumber(5);
                    break;
                case Command.cSix:
                    AddNumber(6);
                    break;
                case Command.cSeven:
                    AddNumber(7);
                    break;
                case Command.cEight:
                    AddNumber(8);
                    break;
                case Command.cNine:
                    AddNumber(9);
                    break;
                case Command.cA:
                    AddNumber(10);
                    break;
                case Command.cB:
                    AddNumber(11);
                    break;
                case Command.cC:
                    AddNumber(12);
                    break;
                case Command.cD:
                    AddNumber(13);
                    break;
                case Command.cE:
                    AddNumber(14);
                    break;
                case Command.cF:
                    AddNumber(15);
                    break;
                case Command.cSign:
                    ToggleMinus();
```

```
                    break;
                case Command.cSeparator:
                    AddSeparator();
                    break;
                case Command.cBS:
                    RemoveSymbol();
                    break;
                case Command.cCE:
                    Clear();
                    break;
                default:
                    break;
            }
            return Number;
        }
    }
}
```

ANumber.cs

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

namespace calculator
{
    public abstract class ANumber
    {
        public abstract void
SetString(string str);
    }
}
```

TPNumber.cs

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

namespace calculator
{
    public class TPNumber : ANumber
    {
        public class
ADT_Convert_10_p
        {
            public static string
Do(double n, int p, int c)
            {
                if (p < 2 || p > 16)
                    throw new
IndexOutOfRangeException();
                if (c < 0 || c > 10)
                    throw new
IndexOutOfRangeException();

                long leftSide =
(long)n;

                double rightSide = n
- leftSide;
                if (rightSide < 0)
                    rightSide *= -1;

                string
leftSideString = Int_to_p(leftSide,
p);
                string
rightSideString =
Flt_to_p(rightSide, p, c);

                return
leftSideString + (rightSideString ==
String.Empty ? "" : ".") +
rightSideString;
            }

            public static char
Int_to_char(int d)
            {
                if (d > 15 || d < 0)
                {
                    throw new
IndexOutOfRangeException();
                }
                string allSymbols =
"0123456789ABCDEF";
                return
allSymbols.ElementAt(d);
            }

            public static string
Int_to_p(long n, int p)
            {
                if (p < 2 || p > 16)
                    throw new
IndexOutOfRangeException();
                if (n == 0)
                    return "0";
                if (p == 10)
                    return
n.ToString();

                bool isNegative =
false;
                if (n < 0)
                {
```

```
                    isNegative =
true;
                    n *= -1;
                }

                string buf = "";
                while (n > 0)
                {
                    buf +=
Int_to_char((int)n % p);
                    n /= p;
                }

                if (isNegative)
                    buf += "-";

                char[] chs =
buf.ToCharArray();
                Array.Reverse(chs);
                return new
string(chs);
            }

            public static string
Flt_to_p(double n, int p, int c)
            {
                if (p < 2 || p > 16)
                    throw new
IndexOutOfRangeException();
                if (c < 0 || c > 10)
                    throw new
IndexOutOfRangeException();

                string pNumber =
String.Empty;
                for (int i = 0; i <
c; i++)
                {
                    pNumber +=
Int_to_char((int)(n * p));
                    n = n * p -
(int)(n * p);
                }
                pNumber =
pNumber.TrimEnd('0');
                return pNumber;
            }
        }

        public class
ADT_Convert_p_10
        {
            public static double
Dval(string p_num, int p)
            {
                if (p < 2 || p > 16)
                    throw new
IndexOutOfRangeException();

                bool minus = false;
                if
(p_num.Contains("-"))
                {
                    minus = true;
                    p_num =
p_num.Replace("-", "");
                }
                double buf = 0d;
                if
(p_num.Contains("."))
                {
                    string[] lr =
p_num.Split('.');
                    if (lr[0].Length
== 0)
                        throw new
Exception();
                    char[] chs =
lr[0].ToCharArray();

Array.Reverse(chs);
                    for (int i = 0;
i < chs.Length; i++)
                    {
                        if
(Char_to_num(chs[i]) > p)
                            throw
new Exception();
                        buf +=
Char_to_num(chs[i]) * Math.Pow(p,
i);
                    }
                    char[] chsr =
lr[1].ToCharArray();
                    for (int i = 0;
i < chsr.Length; i++)
                    {
                        if
(Char_to_num(chsr[i]) > p)
                            throw
new Exception();
                        buf +=
Char_to_num(chsr[i]) * Math.Pow(p, -
(i + 1));
                    }
                }
                else
                {
                    char[] chs =
p_num.ToCharArray();
```

```
Array.Reverse(chs);
                    for (int i = 0;
i < chs.Length; i++)
                    {
                        if
(Char_to_num(chs[i]) > p)
                            throw
new Exception();
                        buf +=
Char_to_num(chs[i]) * Math.Pow(p,
i);
                    }
                }
                if (minus)
                {
                    buf = buf * -1;
                }
                return buf;
            }

            public static double
Char_to_num(char ch)
            {
                string allNums =
"0123456789ABCDEF";
                if
(!allNums.Contains(ch))
                    throw new
IndexOutOfRangeException();
                return
allNums.IndexOf(ch);

            }

            public static double
Convert(string p_num, int p, double
weight)
            {
                return 0d;
            }
        }

        public double Number;
        public int Notation;
        public int Precision;

        public TPNumber()
        {
            Number = 0f;
            Notation = 10;
            Precision = 5;
        }
        public TPNumber(double num,
int not, int pre)
        {
            if (not < 2 || not > 16
|| pre < 0 || pre > 10)
            {
                Number = 0f;
                Notation = 10;
                Precision = 5;
            }
            else
            {
                Number = num;
                Notation = not;
                Precision = pre;
            }
        }
        public TPNumber(TPNumber
anotherTPNumber)
        {
            Number =
anotherTPNumber.Number;
            Notation =
anotherTPNumber.Notation;
            Precision =
anotherTPNumber.Precision;
        }
        public TPNumber(string str,
int not, int pre)
        {
            try
            {
                Number =
ADT_Convert_p_10.Dval(str, not);
                Notation = not;
                Precision = pre;
            }
            catch
            {
                throw new
System.OverflowException();
            }
        }

        public TPNumber Add(TPNumber
a)
        {
            TPNumber tmp = a.Copy();
            if (a.Notation !=
this.Notation || a.Precision !=
Precision)
            {
                tmp.Number = 0.0;
                return tmp;
            }
            tmp.Number = Number +
a.Number;
```

```
            return tmp;
        }

        public TPNumber Mul(TPNumber
a)
        {
            TPNumber tmp = a.Copy();
            if (a.Notation !=
this.Notation || a.Precision !=
this.Precision)
            {
                tmp.Number = 0.0;
                return tmp;
            }
            tmp.Number = Number *
a.Number;
            return tmp;
        }

        public TPNumber Div(TPNumber
a)
        {
            TPNumber tmp = a.Copy();
            if (a.Notation !=
Notation || a.Precision !=
Precision)
            {
                tmp.Number = 0.0;
                return tmp;
            }
            tmp.Number = Number /
a.Number;
            return tmp;
        }

        public TPNumber Sub(TPNumber
a)
        {
            TPNumber tmp = a.Copy();
            if (a.Notation !=
Notation || a.Precision !=
Precision)
            {
                tmp.Number = 0.0;
                return tmp;
            }
            tmp.Number = Number -
a.Number;
            return tmp;
        }
        public object Square()
        {
            return new
TPNumber(Number * Number, Notation,
Precision);
        }
        public object Reverse()
        {
            return new TPNumber(1 /
Number, Notation, Precision);
        }
        public bool IsZero()
        {
            return Number == 0;
        }

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

        public override string
ToString()
        {
            string str;
            try
            {
                str =
ADT_Convert_10_p.Do(Number,
Notation, Precision);

            }
            catch
            {
                throw new
System.OverflowException();
            }
            return str;
        }

        public override void
SetString(string str)
        {
            try
            {
                Number =
ADT_Convert_p_10.Dval(str,
Notation);
            }
            catch
            {
                throw new
System.OverflowException();
            }
}

        private bool check(double a,
int b, int c)
        {
```

```
            string a_str =
a.ToString();
            if (!checkOnBase(a_str,
b))
            {
                return false;
            }
            if (!checkOnC(a_str, c))
            {
                return false;
            }
            if
(!checkOnSymbol(a_str))
            {
                return false;
            }
            return true;
        }
        private bool check(string a,
string b, string c)
        {
            int b_int =
Convert.ToInt32(b);
            int c_int =
Convert.ToInt32(c);
            if (!checkOnBase(a,
b_int))
            {
                return false;
            }
            if (!checkOnC(a, c_int))
            {
                return false;
            }
            if (!checkOnSymbol(a))
            {
                return false;
            }
            return true;
        }

        private bool
checkOnBase(string a, int b)
        {
            foreach (char iter in a)
            {
                int move =
Math.Abs('A' - iter);
                int iter_int = iter
- '0';
                if (iter >= 'A' &&
iter <= 'Z')
                {
                    iter_int = 10 +
move;
                }
                if (iter == ',')
                {
                    continue;
                }
                if (iter_int >= b)
                {
                    return false;
                }
            }
            return true;
        }

        private bool checkOnC(string
a, int c)
        {
            if (checkPoint(a) && c >
0)
            {
                string[] spliter =
a.Split(',');
                if
(spliter[1].Length == c)
                {
                    return true;
                }
            }
            return false;
        }
        private bool
checkPoint(string n)
        {
            int i;
            for (i = 0; i < n.Length
&& n[i] != ','; i++) { }
            if (i < n.Length)
            {
                return true;
            }
            return false;
        }
        private bool
checkOnSymbol(string a)
        {
            foreach (char iter in a)
            {
                if (iter >= 'a' &&
iter <= 'z')
                {
                    return false;
                }
            }
            return true;
        }
    }
```

```
}
```

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 calculator
{
    public partial class Form1 : Form
    {
        ADT_Control<TFrac, TFracEditor>
fracController;
        ADT_Control<TComplex,
TComplexEditor> complexController;
        ADT_Control<TPNumber,
TPNumberEditor> pNumberController;

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


        const string TAG_FRAC = "FRAC_";
        const string TAG_COMPLEX =
"COMPLEX_";
        const string TAG_PNUMBER =
"PNUMBER_";

        public Form1()
        {
            fracController = new
ADT_Control<TFrac, TFracEditor>();
            complexController = new
ADT_Control<TComplex, TComplexEditor>();
            pNumberController = new
ADT_Control<TPNumber, TPNumberEditor>();
            InitializeComponent();
        }

        private string
NumberBeautifier(string Tag, 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();

            switch (Tag)
            {
                case TAG_PNUMBER:
                    break;
                case TAG_FRAC:
                    if (fracMode ==
true)
                        toReturn = v;
                    else if (new
TFrac(v).Denominator == 1)
                        toReturn = new
TFrac(v).Numerator.ToString();
                    break;
                case TAG_COMPLEX:
                    if (complexMode ==
true)
                        toReturn = v;
                    else if (new
TComplex(v).Imaginary == 0)
                        toReturn = new
TComplex(v).Real.ToString();
                    break;
            }

            return toReturn;
        }

        private void
CopyToolStripMenuItem_Click(object
sender, EventArgs e)
        {
            memmory_buffer =
tB_Frac.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)
                tB_Frac.Text =
fracController.ExecCommandEditor(CharToE
ditorCommand(i));
        }

        private static AEditor.Command
CharToEditorCommand(char ch)
        {
            AEditor.Command command =
AEditor.Command.cNone;
            switch (ch)
            {
                case '0':
                    command =
AEditor.Command.cZero;
                    break;
                case '1':
                    command =
AEditor.Command.cOne;
                    break;
```

```
                case '2':
                    command =
AEditor.Command.cTwo;
                    break;
                case '3':
                    command =
AEditor.Command.cThree;
                    break;
                case '4':
                    command =
AEditor.Command.cFour;
                    break;
                case '5':
                    command =
AEditor.Command.cFive;
                    break;
                case '6':
                    command =
AEditor.Command.cSix;
                    break;
                case '7':
                    command =
AEditor.Command.cSeven;
                    break;
                case '8':
                    command =
AEditor.Command.cEight;
                    break;
                case '9':
                    command =
AEditor.Command.cNine;
                    break;
                case 'A':
                    command =
AEditor.Command.cA;
                    break;
                case 'B':
                    command =
AEditor.Command.cB;
                    break;
                case 'C':
                    command =
AEditor.Command.cC;
                    break;
                case 'D':
                    command =
AEditor.Command.cD;
                    break;
                case 'E':
                    command =
AEditor.Command.cE;
                    break;
                case 'F':
                    command =
AEditor.Command.cF;
                    break;
                case '.':
                    command =
AEditor.Command.cSeparator;
                    break;
                case '-':
                    command =
AEditor.Command.cSign;
                    break;
                case 'i':
                    command =
AEditor.Command.cToggleComplexMode;
                    break;
            }
            return command;
        }
        private static
ADT_Proc<T>.Operations
CharToOperationsCommand<T>(char ch)
where T : ANumber, new()
        {
            ADT_Proc<T>.Operations
command = ADT_Proc<T>.Operations.None;
            switch (ch)
            {
                case '+':
                    command =
ADT_Proc<T>.Operations.Add;
                    break;
                case '-':
                    command =
ADT_Proc<T>.Operations.Sub;
                    break;
                case '*':
                    command =
ADT_Proc<T>.Operations.Mul;
                    break;
                case '/':
                    command =
ADT_Proc<T>.Operations.Div;
                    break;
            }

            return command;
        }

        private static AEditor.Command
KeyCodeToEditorCommand(Keys ch)
        {
            AEditor.Command command =
AEditor.Command.cNone;
            switch (ch)
            {
                case Keys.Back:
                    command =
AEditor.Command.cBS;
                    break;
                case Keys.Delete:
                case Keys.Escape:
                    command =
AEditor.Command.cCE;
                    break;
            }

            return command;
        }


        private void
AboutToolStripMenuItem1_Click(object
sender, EventArgs e)
        {
            MessageBox.Show("Калькулятор
дробных чисел\n" +
                    " Ргр: Тодоров ИП-017\n,
"Справка", MessageBoxButtons.OK,
MessageBoxIcon.Information);
        }

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

```
            Button button =
(Button)sender;
            string Tag =
button.Tag.ToString();

            if
(Tag.StartsWith(TAG_FRAC))
            {
                string tag_command =
Tag.Replace(TAG_FRAC, string.Empty);

Enum.TryParse(tag_command, out
AEditor.Command ParsedEnum);
                tB_Frac.Text =
fracController.ExecCommandEditor(ParsedE
num);
            }
            else if
(Tag.StartsWith(TAG_COMPLEX))
            {
                string tag_command =
Tag.Replace(TAG_COMPLEX, string.Empty);

Enum.TryParse(tag_command, out
AEditor.Command ParsedEnum);
                tB_Complex.Text =
complexController.ExecCommandEditor(Pars
edEnum);
            }
            else if
(Tag.StartsWith(TAG_PNUMBER))
            {
                string tag_command =
Tag.Replace(TAG_PNUMBER, string.Empty);

Enum.TryParse(tag_command, out
AEditor.Command ParsedEnum);
                tB_PNumber.Text =
pNumberController.ExecCommandEditor(Pars
edEnum);
            }

        }

        private void
Button_Number_Operation(object sender,
EventArgs e)
        {

            Button button =
(Button)sender;
            string Tag =
button.Tag.ToString();

            if
(Tag.StartsWith(TAG_FRAC))
            {
                string tag_command =
Tag.Replace(TAG_FRAC, string.Empty);

Enum.TryParse(tag_command, out
ADT_Proc<TFrac>.Operations ParsedEnum);
                tB_Frac.Text =
NumberBeautifier(TAG_FRAC,
fracController.ExecOperation(ParsedEnum)
);
            }
            else if
(Tag.StartsWith(TAG_COMPLEX))
            {
                string tag_command =
Tag.Replace(TAG_COMPLEX, string.Empty);

Enum.TryParse(tag_command, out
ADT_Proc<TComplex>.Operations
ParsedEnum);
                tB_Complex.Text =
NumberBeautifier(TAG_COMPLEX,
complexController.ExecOperation(ParsedEn
um));
            }
            else if
(Tag.StartsWith(TAG_PNUMBER))
            {
                string tag_command =
Tag.Replace(TAG_PNUMBER, string.Empty);

Enum.TryParse(tag_command, out
ADT_Proc<TPNumber>.Operations
ParsedEnum);
                tB_PNumber.Text =
NumberBeautifier(TAG_PNUMBER,
pNumberController.ExecOperation(ParsedEn
um));
            }
        }

        private void
Button_Number_Function(object sender,
EventArgs e)
        {
            Button button =
(Button)sender;
            string Tag =
button.Tag.ToString();

            if
(Tag.StartsWith(TAG_FRAC))
            {
                string tag_command =
Tag.Replace(TAG_FRAC, string.Empty);

Enum.TryParse(tag_command, out
ADT_Proc<TFrac>.Functions ParsedEnum);
                tB_Frac.Text =
NumberBeautifier(TAG_FRAC,
fracController.ExecFunction(ParsedEnum))
;
            }
            else if
(Tag.StartsWith(TAG_COMPLEX))
            {
                string tag_command =
Tag.Replace(TAG_COMPLEX, string.Empty);

Enum.TryParse(tag_command, out
ADT_Proc<TComplex>.Functions
ParsedEnum);
                tB_Complex.Text =
NumberBeautifier(TAG_COMPLEX,
complexController.ExecFunction(ParsedEnu
m));
            }
            else if
(Tag.StartsWith(TAG_PNUMBER))
            {
                string tag_command =
Tag.Replace(TAG_PNUMBER, string.Empty);

Enum.TryParse(tag_command, out
```

```
ADT_Proc<TPNumber>.Functions
ParsedEnum);
                tB_PNumber.Text =
NumberBeautifier(TAG_PNUMBER,
pNumberController.ExecFunction(ParsedEnu
m));
            }
        }

        private void
Button_Memory(object sender, EventArgs
e)
        {
            Button button =
(Button)sender;
            string Tag =
button.Tag.ToString();
            if
(Tag.StartsWith(TAG_FRAC))
            {
                string tag_command =
Tag.Replace(TAG_FRAC, string.Empty);

Enum.TryParse(tag_command, out
TMemory<TFrac>.Commands ParsedEnum);
                dynamic exec =
fracController.ExecCommandMemory(ParsedE
num, tB_Frac.Text);
                if (ParsedEnum ==
TMemory<TFrac>.Commands.Copy)
                    tB_Frac.Text =
exec.Item1.ToString();
                label_Frac_Memory.Text =
exec.Item2 == true ? "M" : string.Empty;
            }
            else if
(Tag.StartsWith(TAG_COMPLEX))
            {
                string tag_command =
Tag.Replace(TAG_COMPLEX, string.Empty);

Enum.TryParse(tag_command, out
TMemory<TComplex>.Commands ParsedEnum);
                dynamic exec =
complexController.ExecCommandMemory(Pars
edEnum, tB_Complex.Text);
                if (ParsedEnum ==
TMemory<TComplex>.Commands.Copy)
                    tB_Complex.Text =
exec.Item1.ToString();

label_Complex_Memory.Text = exec.Item2
== true ? "M" : string.Empty;
            }
            else if
(Tag.StartsWith(TAG_PNUMBER))
            {
                string tag_command =
Tag.Replace(TAG_PNUMBER, string.Empty);

Enum.TryParse(tag_command, out
TMemory<TPNumber>.Commands ParsedEnum);
                dynamic exec =
pNumberController.ExecCommandMemory(Pars
edEnum, tB_Complex.Text);
                if (ParsedEnum ==
TMemory<TPNumber>.Commands.Copy)
                    tB_PNumber.Text =
exec.Item1.ToString();

label_PNumber_Memory.Text = exec.Item2
== true ? "M" : string.Empty;
            }
        }

        private void Button_Reset(object
sender, EventArgs e)
        {
            Button button =
(Button)sender;
            string Tag =
button.Tag.ToString();
            if
(Tag.StartsWith(TAG_FRAC))
            {
                tB_Frac.Text =
fracController.Reset();
                label_Frac_Memory.Text =
string.Empty;
            }
            else if
(Tag.StartsWith(TAG_COMPLEX))
            {
                tB_Complex.Text =
complexController.Reset();

label_Complex_Memory.Text =
string.Empty;
            }
            else if
(Tag.StartsWith(TAG_PNUMBER))
            {
                tB_PNumber.Text =
pNumberController.Reset();

label_PNumber_Memory.Text =
string.Empty;
            }

        }

        private void
Button_Calculate(object sender,
EventArgs e)
        {
            Button button =
(Button)sender;
            string Tag =
button.Tag.ToString();
            if
(Tag.StartsWith(TAG_FRAC))
            {
                tB_Frac.Text =
NumberBeautifier(TAG_FRAC,
fracController.Calculate());
            }
            else if
(Tag.StartsWith(TAG_COMPLEX))
            {
                tB_Complex.Text =
NumberBeautifier(TAG_COMPLEX,
complexController.Calculate());
            }
            else if
(Tag.StartsWith(TAG_PNUMBER))
            {
                tB_PNumber.Text =
pNumberController.Calculate();
            }
```

```
        }

        private void
TrackBar_PNumber_ValueChanged(object
sender, EventArgs e)
        {
            label_PNumber_P.Text =
trackBar_PNumber.Value.ToString();

pNumberController.Edit.Notation =
trackBar_PNumber.Value;
            tB_PNumber.Text =
pNumberController.Reset();
            label_PNumber_Memory.Text =
string.Empty;
            string AllowedEndings =
"0123456789ABCDEF";
            foreach (Control i in
tabPage_PNumber.Controls.OfType<Button>(
))
            {
                if
(AllowedEndings.Contains(i.Name.ToString
().Last()) &&
i.Name.ToString().Substring(i.Name.ToStr
ing().Length - 2, 1) == "_")
                {
                    int j =
AllowedEndings.IndexOf(i.Name.ToString()
.Last());
                    if (j <
trackBar_PNumber.Value)
                    {
                        i.Enabled =
true;
                    }
                    if ((j >=
trackBar_PNumber.Value) && (j <= 15))
                    {
                        i.Enabled =
false;
                    }
                }
            }

pNumberController.Proc.Left_Result_opera
nd.Notation = trackBar_PNumber.Value;

pNumberController.Proc.Right_operand.Not
ation = trackBar_PNumber.Value;
        }

        private void
Form1_KeyDown(object sender,
KeyEventArgs e)
        {


            switch
(tabControl.SelectedIndex)
            {
                case 0: {
                    if (e.KeyCode ==
Keys.Enter)

b_Frac_Eval.PerformClick();
                    else
                    {
                        AEditor.Command
command =
KeyCodeToEditorCommand(e.KeyCode);
                        if (command !=
AEditor.Command.cNone)
                            tB_Frac.Text
=
fracController.ExecCommandEditor(command
);
                    }
                    break;
                }
                case 1: {
                    if (e.KeyCode ==
Keys.Enter)

b_Complex_Eval.PerformClick();
                    else
                    {
                        AEditor.Command
command =
KeyCodeToEditorCommand(e.KeyCode);
                        if (command !=
AEditor.Command.cNone)

tB_Complex.Text =
complexController.ExecCommandEditor(comm
and);
                    }
                    break;
                }
                case 2: {
                    if (e.KeyCode ==
Keys.Enter)

b_Complex_Eval.PerformClick();
                    else
                    {
                        AEditor.Command
command =
KeyCodeToEditorCommand(e.KeyCode);
                        if (command !=
AEditor.Command.cNone)

tB_Complex.Text =
pNumberController.ExecCommandEditor(comm
and);
                    }
                    break;
                }
                default:
                    break;
            }
        }

        private void
Form1_KeyPress(object sender,
KeyPressEventArgs e)
        {
            switch
(tabControl.SelectedIndex)
            {
                case 0: {
                    if (e.KeyChar >= '0'
&& e.KeyChar <= '9' || e.KeyChar == '.')
                        tB_Frac.Text =
fracController.ExecCommandEditor(CharToE
ditorCommand(e.KeyChar));
                    else if
(operation_signs.Contains(e.KeyChar))
```

```
                        tB_Frac.Text =
NumberBeautifier(TAG_FRAC,
fracController.ExecOperation(CharToOpera
tionsCommand<TFrac>(e.KeyChar)));
                    break;
                }
                case 1: {
                    if ((e.KeyChar >=
'0' && e.KeyChar <= '9') || e.KeyChar ==
'.')
                        tB_Complex.Text
=
complexController.ExecCommandEditor(Char
ToEditorCommand(e.KeyChar));
                    else if
(operation_signs.Contains(e.KeyChar))
                        tB_Complex.Text
= NumberBeautifier(TAG_COMPLEX,
complexController.ExecOperation(CharToOp
erationsCommand<TComplex>(e.KeyChar)));
                    break;
                }
                case 2: {
                    if ((e.KeyChar >=
'0' && e.KeyChar <= '9') || (e.KeyChar
>= 'A' && e.KeyChar <= 'F') ||
(e.KeyChar == '.'))
                        tB_PNumber.Text
=
pNumberController.ExecCommandEditor(Char
ToEditorCommand(e.KeyChar));
                    else if
(operation_signs.Contains(e.KeyChar))
                        tB_PNumber.Text
= NumberBeautifier(TAG_PNUMBER,
pNumberController.ExecOperation(CharToOp
erationsCommand<TPNumber>(e.KeyChar)));
                    break;
                }
                default:
                    break;
            }
        }
        private void
HistoryToolStripMenuItem1_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(currentRe
cord[0], currentRecord[1]);
            }
            for (int i = 0; i <
complexController.history.Count(); i++)
            {
                List<string>
currentRecord =
complexController.history[i].ToList();

history.dataGridView1.Rows.Add(currentRe
cord[0], currentRecord[1]);
            }
            for (int i = 0; i <
pNumberController.history.Count(); i++)
            {
                List<string>
currentRecord =
pNumberController.history[i].ToList();

history.dataGridView1.Rows.Add(currentRe
cord[0], currentRecord[1]);
            }
        }

        private void
дробьFracTSMI_Click(object sender,
EventArgs e)
        {
            дробьFracTSMI.Checked =
true;
            числоFracTSMI.Checked =
false;
            fracMode = true;
        }

        private void
числоFracTSMI_Click(object sender,
EventArgs e)
        {
            дробьFracTSMI.Checked =
false;
            числоFracTSMI.Checked =
true;
            fracMode = false;
        }

        private void
комплексноеComplexTSMI_Click(object
sender, EventArgs e)
        {

комплексноеComplexTSMI.Checked = true;

действительноеComplexTSMI.Checked =
false;
            complexMode = true;
        }

        private void
действительноеComplexTSMI_Click(object
sender, EventArgs e)
        {

комплексноеComplexTSMI.Checked = false;

действительноеComplexTSMI.Checked =
true;
            complexMode = false;
        }
    }
}
```

THistory.cs:

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

namespace calculator
{
    public class THistory
    {
        public struct Record
        {
            private string oper;
            private string str_res;
            public Record(string
operation, string str_res)
            {
                str_res =
str_res.Remove(0, 1);
                this.oper = operation;
                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 record_string)
        {
            Record record = new
Record(o, record_string);
            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();
        }
    }
}
```

TMemory.cs:

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

namespace calculator
{
    public class TMemory<T> where T :
ANumber, new()
    {
        public enum Commands {
            Store, Add, Clear, Copy
        }

        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(Commands
command, T newNumber) {
            switch (command) {
                case Commands.Store:
                    state = true;
                    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);
        }
    }
}
```