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

Проект по дисциплине
Современные технологии программирования2

«Универсальный калькулятор»

Выполнил: студент 4 курса
гр. ИП-013
Копытина Татьяна Алексеевна
Проверил: старший
преподаватель
к. ПМиК Агалаков А.А.

Новосибирск 2024 г.

Содержание

Цель

Сформировать практические навыки:

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

Задание

Спроектировать и реализовать универсальный калькулятор для выполнения вычислений над числами разного типа, 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.

Необходимо предусмотреть следующие варианты использования (прецеденты) калькулятора:

1. Выполнение одиночных операций:

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

Пример. 5/1 + 2/1 = 7/1.

2. Выполнение операций с одним операндом:

«операнд» «операция» «=» «результат»

Пример. 5/1 * = 25/1.

3. Повторное выполнение операции:

«=» «результат» «=» «результат»

Пример. 5/1 + 4/1 = 9/1 = 13/1 = 17.

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

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

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

5. Вычисление функций:

«операнд» «Sqr» «результат»

Пример. 5/1 «Sqr» 25/1.

6. Вычисление выражений:

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

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

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

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.

Тип числа – «Калькулятор комплексных чисел».

Требования.

1. Калькулятор обеспечивает ввод комплексных чисел в записи:

[-]<действительная часть><разделитель >[-]<мнимая часть>

<действительная часть>::= <действительное число без знака с целой и/или дробной частями>

<мнимая часть>::=<действительное число без знака с целой и/или дробной частями>

<разделитель>::= ‘i*’

2. Предусмотреть настройку калькулятора на отображение результата в двух форматах: “комплексное” или “действительное” число. В формате «комплексное» результат всегда отображается в виде комплексного числа. В формате «действительное» результат отображается в виде действительного, если мнимая часть равна 0.
3. Калькулятор должен вычислять функции: Pwr - возведение в целую степень, Root - извлечение целого корня (Предусмотреть возможность вывода всех корней), Mdl - вычисление модуля комплексного числа, Cnr - вычисление аргумента комплексного числа в градусах, Cnr - вычисление аргумента комплексного числа в радианах. Предусмотреть ввод показателя степени для возведения в степень и извлечения корня. Результат вычисления указанных выше функций отображайте в отдельных компонентах. Эти операции вычисляются отдельно, а не в составе выражения.

Необходимо предусмотреть следующие варианты использования калькулятора (прецеденты):

1. Выполнение одиночных операций:
   «операнд1» «операция» «операнд2» «=» «результат»
   Пример. 5 + 2 = 7.
2. Выполнение операций с одним операндом:
   «операнд» «операция» «=» «результат»
   Пример. 5 * = 25.
3. Повторное выполнение операции:
   «=» «результат» «=» «результат»
   Пример. 5 + 4 = 9 = 13 = 17.
4. Выполнение операции над отображаемым значением в качестве обоих операндов:
   «результат» «операция» «=» «результат»
   Пример. 2 + 3 = 5 = 8 + = 16.

5. Вычисление функций:

   «операнд» «Sqr» «результат»

   Пример. 5 «Sqr» 25 (p = 10)

6. Вычисление выражений:

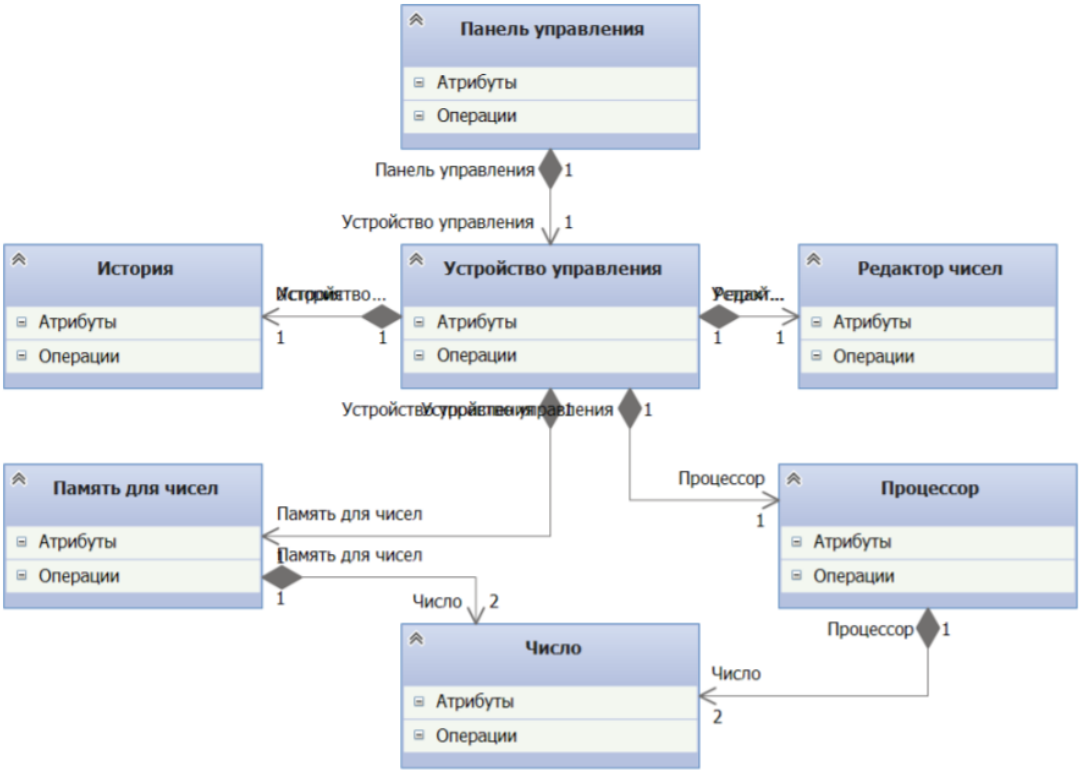
   «операнд1» «функция1» «операция1» «операнд2» «функция2» «операция2» …«операндN» «операцияN» «=»«результат»

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.

Диаграмма классов для прецедентов



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 «р - ичное число».

ADT TPNumber

Данные

Р-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). Р-ичные числа неизменяемые.

Операции

Операции могут вызываться только объектом р-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 р-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

Спецификация типа данных «простые дроби».

ADT TFrac

Данные

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

Операции

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

Конструктор	
Начальные значения:	Пара целых чисел (a) и (b).
Процесс:	Инициализирует поля простой дроби (тип TFrac): числитель значением a, знаменатель - (b). В случае необходимости дробь предварительно сокращается. Например:

	Конструктор(6,3) = (2/1) *Конструктор*(0,3) = (0/3).
Конструктор	
Начальные значения:	Строковое представление простой дроби. Например: ‘7/9’.
Процесс:	Инициализирует поля простой дроби (тип TFrac) строкой f =’a/b’. Числитель значением a, знаменатель - b. В случае необходимости дробь предварительно сокращается. Например: Конструктор(‘6/3’) = 2/1 Конструктор (‘0/3’) = 0/3.
Копировать	
Вход:	Нет.
Предусловия:	Нет.
Процесс:	Создаёт копию самой дроби (тип TFrac) с числителем, и знаменателем такими же, как у самой дроби.
Выход:	Простая дробь (тип TFrac). Например: c = 2/1, Копировать(c) = 2/1
Постусловия:	Нет.
Умножить	
Вход:	Простая дробь d (тип TFrac).
Предусловия:	Нет.
Процесс:	Создаёт простую дробь (тип TFrac), полученную умножением самой дроби q = a1/b1 на d = a2/b2 ((a1/b1)*(a2/b2)=(a1* a2)/(b1* b2)).
Выход:	Простая дробь d (тип TFrac).
Постусловия:	Нет.

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

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

Постусловия:	Нет.
ВзятьЧислительЧисло	
Вход:	Нет.
Предусловия:	Нет.
Процесс:	Возвращает значение числителя дроби в числовом формате.
Выход:	Вещественное значение.
Постусловия:	Нет.
ВзятьЗнаменательЧисло	
Вход:	Нет.
Предусловия:	Нет.
Процесс:	Возвращает значение знаменателя дроби в числовом формате.
Выход:	Вещественное значение.
Постусловия:	Нет.
ВзятьЧислительСтрока	
Вход:	Нет.
Предусловия:	Нет.
Процесс:	Возвращает значение числителя дроби в строковом формате.
Выход:	Строка.
Постусловия:	Нет.
ВзятьЗнаменательСтрока	
Вход:	Нет.
Предусловия:	Нет.

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

End TFracRatio

Спецификация типа данных «комплексное число».

ADT TComplex

Данные

Комплексное число TComplex - это неизменяемая пара вещественных чисел, представляющие действительную и мнимую части комплексного числа (a + i*b).

Операции

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

Конструктор	
Начальные значения:	Пара вещественных чисел a и b.
Процесс:	Устанавливает значения a, b в поля экземпляра класса комплексное число (тип TComplex) this.a - действительной частью и this.b мнимая часть. Например: КонструкторЧисло (6,3)=6 + i*3

	КонструкторЧисло (3,0)=3 + i*0 КонструкторЧисло (0,0)=0 + i*0
Конструктор	
Начальные значения:	Строка f, представляющая комплексное число.
Процесс:	Выделяет из строки f = ’a + i*b’, действительную частью (a) и комплексную часть (b) и преобразует их в число. Устанавливает значения a, b в поля экземпляра класса комплексное число (тип TComplex) this.a - действительной частью и this.b мнимая часть. Например: КонструкторСтрока(‘6+i*3’) = 6+i*3 КонструкторСтрока(‘0+i*3’) = 0+i*3
Копировать	
Вход:	Нет.
Предусловия:	Нет.
Процесс:	Создаёт и возвращает собственную копию - комплексное число (тип TComplex) с действительной и мнимой частями такими же, как у самого числа this.
Выход:	Комплексное число (тип TComplex). Например: c = 6+i3, Копировать(c) = 6+i3
Постусловия:	Нет.
Умножить	
Вход:	Комплексное число d (тип TComplex).
Предусловия:	Нет.
Процесс:	Создаёт и возвращает комплексное число, полученное умножением самого числа this = a1+i*b1 на число d = a2+i*b2: ((a1+i*b1)*(a2+i*b2)=(a1*a2 - b1*b2)+i*(a1*b2+ a2*b1)).
Выход:	Комплексное число d (тип TComplex).

Постусловия:	Нет.
Вычесть	
Вход:	Комплексное число d (тип TComplex).
Предусловия:	Нет.
Процесс:	Создаёт и возвращает комплексное число (тип TComplex), полученное вычитанием d = a2 + i b2 из самого числа this = (a1+i*b1): (a1+i*b1)-(a2+i*b2)=(a1-a2)+i*(b1-b2).
Выход:	Комплексное число d (тип TComplex).
Постусловия:	Нет.
Делить	
Вход:	Комплексное число d (тип TComplex).
Предусловия:	Числитель числа d не равно 0.
Процесс:	Создаёт и возвращает комплексное число (тип TComplex), полученное делением самого числа this на число (d) ((a1+i*b1)/(a2+i*b2)=(a1*a2 + b1*b2)/(a2**2 + b2**2)+i*(a2*b1 – a1*b2)/(a2**2 + b2**2)).
Выход:	Комплексное число d (тип TComplex).
Постусловия:	Нет.
Модуль	
Вход:	Нет.
Предусловия:	Нет.
Процесс:	Вычисляет и возвращает модуль самого комплексного числа this. Например: q = (2 +i*1), q. Модуль = (2*2+1*1). q = (i*17), q. Модуль = (0*0+17*17)
Выход:	Вещественное число.
Постусловия:	Нет.

УголРад	
Вход:	Нет.
Предусловия:	Нет.
Процесс:	Возвращает аргумент fi самого комплексного числа this (в радианах). fi = (arcTg(b/a), a>0; pi/2, a = 0, b > 0; arcTg(b/a) + pi, a < 0; -pi/2, a = 0, b <0). Например: q = (1 +i*1), q. УголРад = 0,79.
Выход:	Вещественное число.
Постусловия:	Нет.
УголГрад	
Вход:	Нет.
Предусловия:	Нет.
Процесс:	Возвращает аргумент самого комплексного числа this (в градусах). Например: q = (1 +i*1), q. Град = 45.
Выход:	Вещественное число.
Постусловия:	Нет.
Корень	
Вход:	Целое n, целое i.
Предусловия:	Нет.
Процесс:	Возвращает i-ый корень целой положительной степени n самого комплексного числа this. √ n (this) = √ n (r)* (cos ((fi + 2*k*pi)/n)+ i* sin ((fi + 2*k*pi)/n)). При этом коофициенту k придается последовательно n значений: k = 0,1,2…, n - 1 и получают n значений корня, т.е. ровно столько, каков показатель корня
Выход:	Комплексное число (тип TComplex).
Постусловия:	Нет.

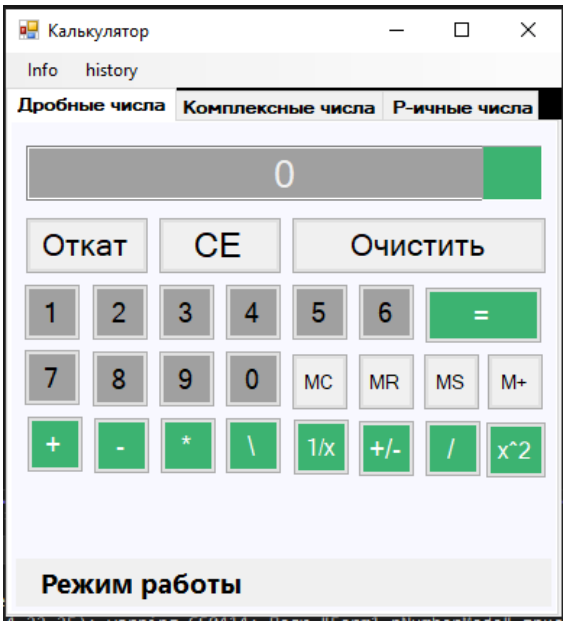
Равно	
Вход:	Комплексное число d.
Предусловия:	Нет.
Процесс:	Сравнивает само комплексное число this с числом d. Возвращает значение True, если они – тождественные комплексные числа, и значение False - в противном случае.
Выход:	Булевское значение.
Постусловия:	Нет.
ВзятьReЧисло	
Вход:	Нет.
Предусловия:	Нет.
Процесс:	Возвращает значение действительной части самого комплексного числа this в числовом формате.
Выход:	Вещественное значение.
Постусловия:	Нет.
ВзятьImЧисло	
Вход:	Нет.
Предусловия:	Нет.
Процесс:	Возвращает значение мнимой части самого комплексного числа this в числовом формате.
Выход:	Вещественное значение.
Постусловия:	Нет.
ВзятьReСтрока	
Вход:	Нет.
Предусловия:	Нет.
Процесс:	Возвращает значение вещественной части самого комплексного числа this в строковом формате.

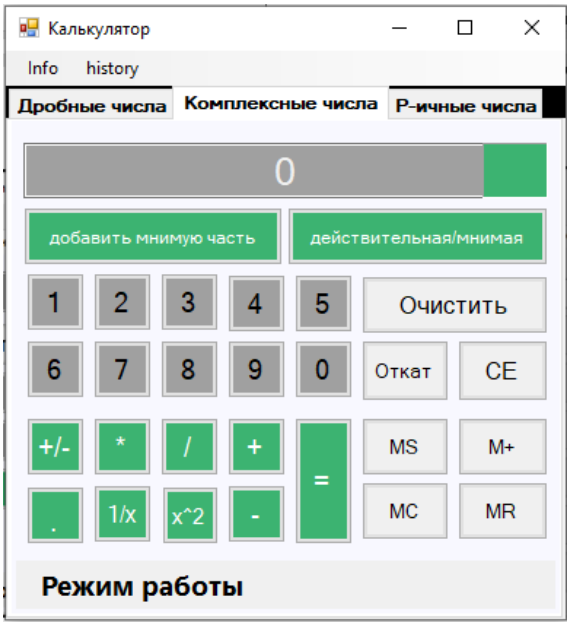
Выход:	Строка.
Постусловия:	Нет.
ВзятьImСтрока	
Вход:	Нет.
Предусловия:	Нет.
Процесс:	Возвращает значение мнимой части самого комплексного числа this в строковом формате.
Выход:	Строка.
Постусловия:	Нет.
ВзятьКомплексноеСтрока	
Вход:	Нет.
Предусловия:	Нет.
Процесс:	Возвращает значение самого комплексного числа this в строковом формате.
Выход:	Строка.
Постусловия:	Нет.

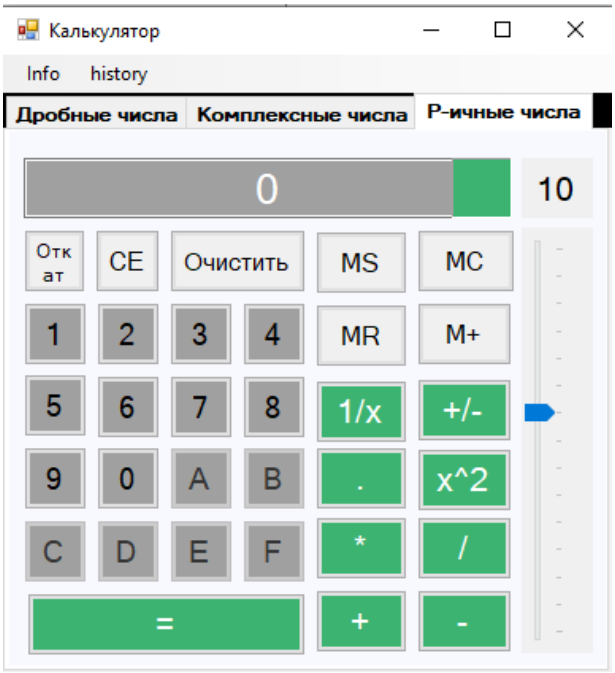
End TComplex

Результаты тестирования программы

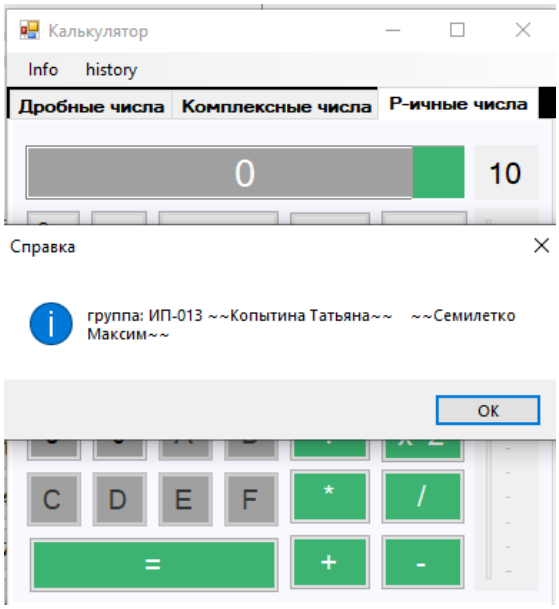
Начало работы:



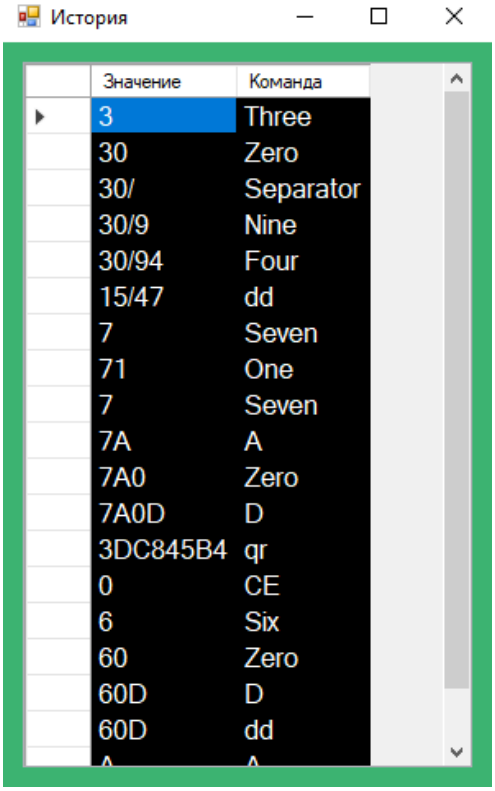




info:



history:



Листинг

ADT_Control_.cs:

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

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

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

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

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

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

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

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

        public string
ExecCommandEditor(AEditor.Command
command)
        {
            string toReturn;
            if (CurState ==
ADT_Control_State.cExpDone)
            {
                Proc.ResetProc();
                CurState =
ADT_Control_State.cStart;
            }
            if (CurState !=
ADT_Control_State.cStart)
                CurState =
ADT_Control_State.cEditing;
            toReturn =
Edit.Edit(command);
            T tmp = new T();
            if (tmp is TPNumber)
            {
                dynamic a = tmp;
```

```
                dynamic b = Edit;
                a.Notation =
b.Notation;
                tmp = a;
            }
            tmp.SetString(toReturn);
            proc.Right_operand =
tmp;

history.AddRecord(toReturn,
command.ToString());

            return toReturn;
        }

        public string
ExecOperation(ADT_Proc<T>.Operations
operation)
        {
            if (operation ==
ADT_Proc<T>.Operations.None)
                return Edit.Number;
            string toReturn;
            try
            {
                switch (CurState)
                {
                    case
ADT_Control_State.cStart:

Proc.Left_Result_operand =
Proc.Right_operand;

Proc.Operation = operation;
                        CurState =
ADT_Control_State.cOpDone;

Edit.Clear();
                        break;
                    case
ADT_Control_State.cEditing:

Proc.DoOperation();

Proc.Operation = operation;

Edit.Clear();
                        CurState =
ADT_Control_State.cOpDone;
                        break;
                    case
ADT_Control_State.FunDone:
                        if
(Proc.Operation == 0)

Proc.Left_Result_operand =
Proc.Right_operand;
                        else

Proc.DoOperation();

Proc.Operation = operation;

Edit.Clear();
                        CurState =
ADT_Control_State.cOpChange;

Proc.Right_operand =
Proc.Left_Result_operand;
                        break;
                    case
ADT_Control_State.cOpDone:
                        CurState =
ADT_Control_State.cOpChange;

Edit.Clear();
                        break;
                    case
ADT_Control_State.cValDone:
                        break;
                    case
ADT_Control_State.cExpDone:

Proc.Operation = operation;

Proc.Right_operand =
Proc.Left_Result_operand;
                        CurState =
ADT_Control_State.cOpChange;

Edit.Clear();
                        break;
                    case
ADT_Control_State.cOpChange:

Proc.Operation = operation;

Edit.Clear();
                        break;
                    case
ADT_Control_State.cError:

Proc.ResetProc();
                        return
"ERR";
                }
                toReturn =
Proc.Left_Result_operand.ToString();
            }
            catch
            {
                Reset();
                return "ERROR";
            }

history.AddRecord(toReturn,
operation.ToString());

            return toReturn;
        }

        public string
ExecFunction(ADT_Proc<T>.Functions
function)
        {
```

```
            string toReturn;
            try
            {
                if (CurState ==
ADT_Control_State.cExpDone)
                {

Proc.Right_operand =
Proc.Left_Result_operand;
                    Proc.Operation =
ADT_Proc<T>.Operations.None;
                }

Proc.DoFunction(function);
                CurState =
ADT_Control_State.FunDone;
                toReturn =
Proc.Right_operand.ToString();
            }
            catch
            {
                Reset();
                return "ERROR";
            }

history.AddRecord(toReturn,
function.ToString());

            return toReturn;
        }

        public string Calculate()
        {
            string ToReturn;
            try
            {
                if (CurState ==
ADT_Control_State.cStart)

Proc.Left_Result_operand =
Proc.Right_operand;
                Proc.DoOperation();
                CurState =
ADT_Control_State.cExpDone;
               //
Edit.SetEditor(Proc.Left_Result_oper
and);
                ToReturn =
Proc.Left_Result_operand.ToString();
            }
            catch
            {
                Reset();
                return "ERROR";
            }

            return ToReturn;
        }

        public (T, bool)
ExecCommandMemory(TMemory<T>.Command
s command, string str)
        {
            T TempObj = new T();
            TempObj.SetString(str);
            (T, bool) obj = (null,
false);
            try
            {
                obj =
Memory.Edit(command, TempObj);
            }
            catch
            {
                Reset();
                return obj;
            }
            if (command ==
TMemory<T>.Commands.Copy)
            {
                Edit.Number =
obj.Item1.ToString();
                Proc.Right_operand =
obj.Item1;
            }
            return obj;
        }
    }
}
```

ADT_Proc.cs:

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

namespace 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;
        }
    }
}
```

AEditor.cs:

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

namespace calculator
{
    public abstract class AEditor
    {
        public enum Command
        {
            cZero, cOne, cTwo, cThree, cFour, cFive, cSix, cSeven,
cEight, cNine,
            cA, cB, cC, cD, cE, cF,
            cSign, cSeparator, cNumbSeparator,
            cBS, cCE,
            cToggleComplexMode,
            cNone
        }
        public abstract string Number
        {
            get;
            set;
        }

        //public abstract bool SetEditor(ANumber number);

        public abstract string AddNumber(int num);
        public abstract string ToggleMinus();
        public abstract string AddSeparator();
        public abstract string AddZero();
        public abstract bool IsZero();
        public abstract string RemoveSymbol();
        public abstract string Clear();
        public abstract string Edit(Command command);
    }
}
```

Program.cs:

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

namespace calculator
{
    static class Program
    {
        /// <summary>
        /// Главная точка входа для приложения.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new Form1());
        }
    }
}
```

TComplex.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 TComplex : ANumber
    {
        private double real;
        private double imaginary;
        const string Separator = " +
i * ";

        public double Real
        {
            get
            {
                return real;
            }
            set
            {
                real = value;
            }
        }

        public double Imaginary
        {

            get
            {
                return imaginary;
            }
            set
            {
                imaginary = value;
            }
        }

        public TComplex()
        {
            real = 0;
            imaginary = 0;
        }

        public TComplex(int a, int
b)
        {
            real = a;
            imaginary = b;
        }

        public TComplex(double a,
double b)
        {
            real = a;
            imaginary = b;
        }

        public TComplex(TComplex
complex)
        {
            real = complex.real;
            imaginary =
complex.imaginary;
        }

        public TComplex(string str)
        {
            Regex FullNumber = new
Regex(@"^-
?(\d+.?\d*)\s+\+\s+i\s+\*\s+-
?(\d+.?\d*)$");
            Regex LeftPart = new
Regex(@"^-
?(\d+.?\d*)(\s+\+\s+i\s+\*\s+)?$");
            if
(FullNumber.IsMatch(str))
            {
                List<string> Parts =
str.Split(new string[] { Separator
},
StringSplitOptions.None).ToList();

                real =
Double.Parse(Parts[0]);
                imaginary =
Double.Parse(Parts[1]);
            }
            else if
(LeftPart.IsMatch(str))
            {
                if
(str.Contains(Separator))
                    str =
str.Replace(Separator,
string.Empty);
                real =
Double.Parse(str);
                imaginary = 0;
            }
            else
            {
                real = 0;
                imaginary = 0;
            }
        }

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

        public TComplex Add(TComplex
b)
        {
            TComplex res =
this.Copy();
            res.real += b.real;
            res.imaginary +=
b.imaginary;
```

```
            return res;
        }

        public TComplex Sub(TComplex
b)
        {
            TComplex res =
this.Copy();
            res.real -= b.real;
            res.imaginary -=
b.imaginary;
            return res;
        }

        public TComplex Mul(TComplex
b)
        {
            TComplex res =
this.Copy();
            res.real = this.real *
b.real - this.imaginary *
b.imaginary;
            res.imaginary =
this.real * b.imaginary +
this.imaginary * b.real;
            return res;
        }

        public TComplex Div(TComplex
b)
        {
            TComplex res =
this.Copy();
            res.real = (this.real *
b.real + this.imaginary *
b.imaginary) / (b.real * b.real +
b.imaginary * b.imaginary);
            res.imaginary = (b.real
* this.imaginary - this.real *
b.imaginary) / (b.real * b.real +
b.imaginary * b.imaginary);
            return res;
        }

        public TComplex Square()
        {
            TComplex res =
this.Copy();
            res.real = this.real *
this.real - this.imaginary *
this.imaginary;
            res.imaginary =
this.real * this.imaginary +
this.real * this.imaginary;
            return res;
        }

        public TComplex Reverse()
        {
            TComplex res =
this.Copy();
            res.real = this.real /
(this.real * this.real +
this.imaginary * this.imaginary);
            res.imaginary = -
this.imaginary / (this.real *
this.real + this.imaginary *
this.imaginary);
            return res;
        }

        public TComplex Minus()
        {
            TComplex res =
this.Copy();
            res.real = 0 - res.real;
            res.imaginary = 0 -
res.imaginary;
            return res;
        }

        public double Abs()
        {
            return
Math.Sqrt(this.real * this.real +
this.imaginary * this.imaginary);
        }

        public double Rad()
        {
            if (this.real > 0)
                return
Math.Atan(this.imaginary /
this.real);

            if (this.real == 0 &&
this.imaginary > 0)
                return (Math.PI /
2);

            if (this.real < 0)
                return
(Math.Atan(this.imaginary /
this.real) + Math.PI);

            if (this.real == 0 &&
this.imaginary < 0)
                return (-Math.PI /
2);

            return 0;
        }

        public double Degree()
        {
            return Rad() * 180 /
Math.PI;
        }

        public TComplex Pow(int n)
        {
            TComplex res =
this.Copy();
```

```
            res.real =
Math.Pow(Abs(), n) * Math.Cos(n *
Rad());
            res.imaginary =
Math.Pow(Abs(), n) * Math.Sin(n *
Rad());
            return res;
        }

        public TComplex Sqrt(int
powN, int rootI)
        {
            if (rootI >= powN ||
rootI < 0 || powN < 0)
                return new
TComplex();
            return new
TComplex(Math.Pow(Abs(), 1.0 / powN)
* Math.Cos((Degree() + 2 * Math.PI *
rootI) / powN), Math.Pow(Abs(), 1.0
/ powN) * Math.Sin((Degree() + 2 *
Math.PI * rootI) / powN));
        }

        public bool Equal(TComplex
anClass)
        {
            return (this.real ==
anClass.real && this.imaginary ==
anClass.imaginary);
        }

        public bool
NotEqual(TComplex anClass)
        {
            return (this.real !=
anClass.real || this.imaginary !=
anClass.imaginary);
        }

        public double
GetRealNumber()
        {
            return this.real;
        }

        public double
GetImaginaryNumber()
        {
            return this.imaginary;
        }

        public string
GetRealString()
        {
            return
this.real.ToString();
        }

        public string
GetImaginaryString()
        {
            return
this.imaginary.ToString();
        }

        public override void
SetString(string str) {
            TComplex temp = new
TComplex(str);
            Real = temp.Real;
            Imaginary =
temp.Imaginary;
        }

        public override string
ToString()
        {
            return GetRealString() +
" + i * " + GetImaginaryString();
        }
    }
}
```

TFrac.cs:

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

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

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

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

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

        public static long GCD(long
a, long b)
        {
            a = Math.Abs(a);
            b = Math.Abs(b);
            while (b > 0)
            {
                a %= b;
                Swap(ref a, ref b);
            }
            return a;
        }

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

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

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

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

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

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

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

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

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

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

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

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

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

        public bool Equal(TFrac b)
        {
            return numerator ==
b.numerator && denominator ==
b.denominator;
        }

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

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

        public static implicit
operator string(TFrac v)
        {
```

```
            throw new
NotImplementedException();
        }

        public long
getNumeratorNum()
        {
            return numerator;
        }

        public long
getDenominatorNum()
        {
            return denominator;
        }

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

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

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

TFracEditor.cs:

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

namespace calculator
{
    public class TFracEditor :
AEditor
    {

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

        public override string
Number
        {
            get
            {
                return fraction;
            }

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

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

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

            return fraction;
        }

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

            return fraction;
        }

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

            return fraction;
        }

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

            return fraction;
        }

        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.cSign:
                    ToggleMinus();
                    break;
                case
Command.cSeparator:
                    AddSeparator();
                    break;
                case Command.cBS:
                    RemoveSymbol();
                    break;
                case Command.cCE:
                    Clear();
                    break;
                default:
                    break;
            }

            return fraction;
        }

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

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

            return fraction;
        }

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

TComplexEditor.cs:

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

namespace calculator
{
    public class TComplexEditor :
AEditor
    {
        public enum PartToEdit
        {
            Real, Imag
        };

        string complex_num;
        PartToEdit mode;
        Regex ZeroComplex = new
Regex(@"^-
?(0+.?0*)(\s*\+\s*i\s*\*\s*-
?(0+.?0*)|(\s*\+\s*i\s*\*\s*-
?))?$");
        const string Separator = " +
i * ";


        public override string
Number
        {
            get
            {
                return complex_num;
            }

            set
            {
                complex_num = new
TComplex(value).ToString();
            }
        }

        public TComplexEditor()
        {
            complex_num = "0";
            mode = PartToEdit.Real;
        }

        public TComplexEditor(int a,
int b)
        {
            complex_num = new
TComplex(a, b).ToString();
            mode = PartToEdit.Real;
        }
        public TComplexEditor(string
str)
        {
            complex_num = new
TComplex(str).ToString();
            mode = PartToEdit.Real;
        }
        public override bool
IsZero()
        {
            /*string tmp = pNumber;
            if (tmp[0] == '-')
                tmp =
tmp.Substring(1);
            tmp = tmp.Replace('-',
'+');
            if (tmp == zero)
                return true;
            else
                return false;*/
            return
ZeroComplex.IsMatch(complex_num);
        }

        public override string
ToggleMinus()
        {
            if
(complex_num.Contains(Separator))
            {
                if (mode ==
PartToEdit.Real)
                {
                    if
(complex_num[0] == '-')
                        complex_num
= complex_num.Substring(1);
                    else
                        complex_num
= '-' + complex_num;
                }
                else
                {
                    complex_num =
complex_num.Substring(0,
complex_num.IndexOf(Separator)) +
Separator + "-" +

complex_num.Substring(complex_num.In
dexOf(Separator) +
Separator.Length);
                }
                return complex_num;
            }
            if (mode ==
PartToEdit.Imag)
                ToggleMode();
            if (complex_num[0] == '-
')
                complex_num =
complex_num.Substring(1);
            else
                complex_num = '-' +
complex_num;

            return complex_num;
        }
```

```
        public PartToEdit
ToggleMode()
        {
            if (mode ==
PartToEdit.Real)
                mode =
PartToEdit.Imag;
            else
                mode =
PartToEdit.Real;
            return mode;
        }

        public override string
AddNumber(int a)
        {
            if (a < 0 || a > 9)
                return complex_num;
            if (a == 0)
                AddZero();
            string left = "", right
= "";
            if
(complex_num.Contains(Separator))
            {
                left =
complex_num.Substring(0,
complex_num.IndexOf(Separator));
                right =
complex_num.Substring(complex_num.In
dexOf(Separator) +
+Separator.Length);
            }
            else
            {
                left = complex_num;
            }

            if (mode ==
PartToEdit.Real)
            {
                if (left == "0" ||
left == "-0")
                    left =
left.First() == '-' ? "-" +
a.ToString() : a.ToString();
                else left +=
a.ToString();
            }
            else
            {
                if (right == "0" ||
right == "-0")
                    right =
right.First() == '-' ? "-" +
a.ToString() : a.ToString();
                else right +=
a.ToString();
            }

            if (right == "")
                complex_num = left;
            else
                complex_num = left +
Separator + right;
            return complex_num;
        }

        public override string
AddZero()
        {
            if (complex_num == "0"
|| complex_num == "-0" ||
complex_num.EndsWith(" 0") ||
complex_num.EndsWith(" -0") ||
complex_num.EndsWith(Separator))
                return complex_num;
            complex_num += "0";
            return complex_num;
        }

        public string
AddNumberSeparator()
        {
            string left = "", right
= "";
            if
(complex_num.Contains(Separator))
            {
                left =
complex_num.Substring(0,
complex_num.IndexOf(Separator));
                right =
complex_num.Substring(complex_num.In
dexOf(Separator) +
Separator.Length);
            }
            else
            {
                left = complex_num;
            }

            if (mode ==
PartToEdit.Real)
            {
                if
(!left.Contains("."))
                    left += ".";
            }
            else
            {
                if
(!right.Contains(".") &&
right.Length > 0)
                    right += ".";
            }

            if (right == "")
                complex_num = left;
            else
                complex_num = left +
Separator + right;

            return complex_num;
        }
```

```
        public override string
AddSeparator()
        {
            if
(!complex_num.Contains(Separator))
            {
                complex_num =
complex_num + Separator + "0";
                mode =
PartToEdit.Imag;
            }
            return complex_num;
        }

        public override string
RemoveSymbol()
        {
            string left = "", right
= "";
            if
(complex_num.Contains(Separator))
            {
                left =
complex_num.Substring(0,
complex_num.IndexOf(Separator));
                right =
complex_num.Substring(complex_num.In
dexOf(Separator) +
Separator.Length);
            }
            else
            {
                left = complex_num;
            }

            if (mode ==
PartToEdit.Real)
            {
                if (left.Length == 1
|| (left.Length == 2 && left[0] ==
'-'))
                {
                    left = left[0]
== '-' ? "-0" : "0";
                }
                else
                {
                    left =
left.Remove(left.Length - 1);
                }
            }
            else
            {
                if (right.Length ==
1 || (right.Length == 2 && right[0]
== '-'))
                {
                    right = right[0]
== '-' ? "-0" : "0";
                }
                else
                {
                    right =
right.Remove(right.Length - 1);
                }
            }

            if (right == "")
                complex_num = left;
            else
                complex_num = left +
Separator + right;

            return complex_num;
        }

        public override string
Clear()
        {
            complex_num = "0";
            mode = PartToEdit.Real;
            return complex_num;
        }

        public override string
ToString()
        {
            return complex_num;
        }

        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.cSign:
                    ToggleMinus();
                    break;
                case
Command.cSeparator:
                    AddNumberSeparator();
                    break;
                case Command.cBS:
                    RemoveSymbol();
                    break;
                case Command.cCE:
                    Clear();
                    break;
                case
Command.cNumbSeparator:
                    AddSeparator();
                    break;
                case
Command.cToggleComplexMode:
                    ToggleMode();
                    break;
                default:
                    break;
            }
            return complex_num;
        }
    }
}
```

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(Cha
rToEditorCommand(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("Универсальный
калькулятор\n" +
                "Авторы:
Бурдуковский Илья и Стояк Юрий\n" +
                "Группа: ИП-813.",
"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(Par
sedEnum);
            }
            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(
ParsedEnum);
            }
            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(
ParsedEnum);
            }

        }

        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(ParsedE
num));
            }
            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(Pars
edEnum));
            }
            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(Pars
edEnum));
            }
        }

        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(ParsedEn
um));
            }
            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(Parse
dEnum));
            }
            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(Parse
dEnum));
            }
        }

        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(Par
sedEnum, 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(
ParsedEnum, 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(
ParsedEnum, 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<Butt
on>())
            {
                if
(AllowedEndings.Contains(i.Name.ToSt
ring().Last()) &&
i.Name.ToString().Substring(i.Name.T
oString().Length - 2, 1) == "_")
                {
                    int j =
AllowedEndings.IndexOf(i.Name.ToStri
ng().Last());
                    if (j <
trackBar_PNumber.Value)
                    {
```

```
                        i.Enabled =
true;
                    }
                    if ((j >=
trackBar_PNumber.Value) && (j <=
15))
                    {
                        i.Enabled =
false;
                    }
                }
            }

pNumberController.Proc.Left_Result_o
perand.Notation =
trackBar_PNumber.Value;

pNumberController.Proc.Right_operand
.Notation = 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(com
mand);
                    }
                    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(
command);
                    }
                    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(
command);
                    }
                    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(Cha
rToEditorCommand(e.KeyChar));
                    else if
(operation_signs.Contains(e.KeyChar)
)
                        tB_Frac.Text
= NumberBeautifier(TAG_FRAC,
fracController.ExecOperation(CharToO
perationsCommand<TFrac>(e.KeyChar)))
;
                    break;
                }
                case 1: {
                    if ((e.KeyChar
>= '0' && e.KeyChar <= '9') ||
e.KeyChar == '.')

tB_Complex.Text =
complexController.ExecCommandEditor(
CharToEditorCommand(e.KeyChar));
```

```
                    else if
(operation_signs.Contains(e.KeyChar)
)

tB_Complex.Text =
NumberBeautifier(TAG_COMPLEX,
complexController.ExecOperation(Char
ToOperationsCommand<TComplex>(e.KeyC
har)));
                    break;
                }
                case 2: {
                    if ((e.KeyChar
>= '0' && e.KeyChar <= '9') ||
(e.KeyChar >= 'A' && e.KeyChar <=
'F') || (e.KeyChar == '.'))

tB_PNumber.Text =
pNumberController.ExecCommandEditor(
CharToEditorCommand(e.KeyChar));
                    else if
(operation_signs.Contains(e.KeyChar)
)

tB_PNumber.Text =
NumberBeautifier(TAG_PNUMBER,
pNumberController.ExecOperation(Char
ToOperationsCommand<TPNumber>(e.KeyC
har)));
                    break;
                }
                default:
                    break;
            }
        }
        private void
HistoryToolStripMenuItem1_Click(obje
ct 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(curre
ntRecord[0], currentRecord[1]);
            }
            for (int i = 0; i <
complexController.history.Count();
i++)
            {
                List<string>
currentRecord =
complexController.history[i].ToList(
);

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

history.dataGridView1.Rows.Add(curre
ntRecord[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(obje
ct sender, EventArgs e)
        {

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

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);
        }
    }
}
```

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();
        }
    }
}
```

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;
        }
    }
}
```

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_cha
r(num).ToString() :
TPNumber.ADT_Convert_10_p.Int_to_cha
r(num).ToString();
            else
                pNumber +=
TPNumber.ADT_Convert_10_p.Int_to_cha
r(num).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;
        }
    }
}
```

Листинг тестов

UnitTest1.cs:

```
using
Microsoft.VisualStudio.TestTools
.UnitTesting;
using calculator;
using System;

namespace calculator_tests
{
    [TestClass]
    public class TFracTest
    {
        [TestMethod]
        public void
InitString1()
        {
            string fracString =
"1/2";
            TFrac fracClass =
new TFrac(fracString);

Assert.AreEqual(fracString,
fracClass.ToString());
        }

        [TestMethod]
        public void
InitString2()
        {
            string fracString =
"111/2";
            TFrac fracClass =
new TFrac(fracString);

Assert.AreEqual(fracString,
fracClass.ToString());
        }

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

Assert.AreEqual(Expect,
fracClass.ToString());
        }

        [TestMethod]
        public void
InitString4()
        {
            string fracString =
"00000003/000004";
            TFrac fracClass =
new TFrac(fracString);
            string Expect =
"3/4";

Assert.AreEqual(Expect,
fracClass.ToString());
        }

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

Assert.AreEqual(Expect,
fracClass.ToString());
        }

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

Assert.AreEqual(Expect,
fracClass.ToString());
        }

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

Assert.AreEqual(Expect,
fracClass.ToString());
        }

        [TestMethod]
```

```
        public void
InitNumber3()
        {
            TFrac fracClass =
new TFrac(-100, -99);
            string Expect =
"100/99";

Assert.AreEqual(Expect,
fracClass.ToString());
        }

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

Assert.AreEqual(Expect,
fracClass.ToString());
        }

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

Assert.AreEqual(fracCompar,
fracClass.ToString());
        }

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

        [TestMethod]
        public void Substract2()
        {
            TFrac fracClass1 =
new TFrac(5, 1);
            TFrac fracClass2 =
new TFrac(1, 1);
            fracClass2 =
fracClass1.Sub(fracClass2);
            string answer =
"4/1";

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

        [TestMethod]
        public void Divide5()
        {
```

```
            TFrac fracClass1 =
new TFrac(2, 3);
            TFrac fracClass2 =
new TFrac(2, 3);
            fracClass2 =
fracClass1.Div(fracClass2);
            string answer =
"1/1";

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

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

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

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

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

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

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

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

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

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

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

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

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

        [TestMethod]
        public void Equal1()
        {
            TFrac fracClass1 =
new TFrac(1, 3);
            TFrac fracClass2 =
new TFrac(1, 3);

Assert.IsTrue(fracClass1.Equal(f
racClass2));
        }

        [TestMethod]
        public void Equal2()
        {
            TFrac fracClass1 =
new TFrac(0, 6);
            TFrac fracClass2 =
new TFrac(1, 6);
```

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

        [TestMethod]
        public void Equal3()
        {
            TFrac fracClass1 =
new TFrac(-1, 6);
            TFrac fracClass2 =
new TFrac(-1, 6);

Assert.IsTrue(fracClass1.Equal(f
racClass2));
        }

        [TestMethod]
        public void Equal4()
        {
            TFrac fracClass1 =
new TFrac(-1, 7);
            TFrac fracClass2 =
new TFrac(1, 7);

Assert.IsFalse(fracClass1.Equal(
fracClass2));
        }

        [TestMethod]
        public void Equal5()
        {
            TFrac fracClass1 =
new TFrac(1, 6);
            TFrac fracClass2 =
new TFrac(0, 1);

Assert.IsFalse(fracClass1.Equal(
fracClass2));
        }

        [TestMethod]
        public void Greater1()
        {
            TFrac fracClass1 =
new TFrac(1, 6);
            TFrac fracClass2 =
new TFrac(0, 1);

Assert.IsTrue(fracClass1 >
fracClass2);
        }

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

Assert.IsFalse(fracClass1 >
fracClass2);
        }

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

Assert.IsFalse(fracClass1 >
fracClass2);
        }

        [TestMethod]
        public void Greater4()
        {
            TFrac fracClass1 =
new TFrac(17, 3);
            TFrac fracClass2 =
new TFrac(16, 3);

Assert.IsTrue(fracClass1 >
fracClass2);
        }

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

Assert.IsFalse(fracClass1 >
fracClass2);
        }
    }

    [TestClass]
    public class TFracEditorTest
    {
        [TestMethod]
        public void Init1()
        {
            TFracEditor
testClass = new TFracEditor();
            string input =
"3/4";
            testClass.Number =
input;
```

```
Assert.AreEqual(input,
testClass.Number);
        }
        [TestMethod]
        public void Init2()
        {
            TFracEditor
testClass = new TFracEditor();
            string input = "-
16/3";
            testClass.Number =
input;

Assert.AreEqual(input,
testClass.Number);
        }
        [TestMethod]
        public void Init3()
        {
            TFracEditor
testClass = new TFracEditor();
            string input =
"0/8";
            testClass.Number =
input;
            string result =
"0/1";

Assert.AreEqual(result,
testClass.Number);
        }
        [TestMethod]
        public void Init4()
        {
            TFracEditor
testClass = new TFracEditor();
            string input = "-
17/4";
            testClass.Number =
input;

Assert.AreEqual(input,
testClass.Number);
        }

        [TestMethod]
        public void Init5()
        {
            TFracEditor
testClass = new TFracEditor();
            string input =
"0/1";
            testClass.Number =
input;

Assert.AreEqual(input,
testClass.Number);
        }

        [TestMethod]
        public void Init6()
        {
            TFracEditor
testClass = new TFracEditor();
            string input =
"666/6666";
            testClass.Number =
input;
            string result =
"111/1111";

Assert.AreEqual(result,
testClass.Number);
        }

        [TestMethod]
        public void Init7()
        {
            TFracEditor
testClass = new TFracEditor();
            string input =
"aaaa";
            testClass.Number =
input;
            string result =
"0/1";

Assert.AreEqual(result,
testClass.Number);
        }

        [TestMethod]
        public void Init8()
        {
            TFracEditor
testClass = new TFracEditor();
            string input =
"0/1";
            testClass.Number =
input;

Assert.AreEqual(input,
testClass.Number);
        }

        [TestMethod]
        public void Init10()
        {
            TFracEditor
testClass = new TFracEditor();
            string input =
"16/000000";
            testClass.Number =
input;
```

```
            string result =
"0/1";

Assert.AreEqual(result,
testClass.Number);
        }

        [TestMethod]
        public void hasZero1()
        {
            TFracEditor
testClass = new
TFracEditor("14/3");

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

Assert.AreEqual(true,
testClass.IsZero());
        }

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

testClass.ToggleMinus();
            string result = "-
14/3";

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

testClass.ToggleMinus();
            string result =
"14/3";

Assert.AreEqual(result,
testClass.ToString());
        }
        [TestMethod]
        public void
AddDeleteTest1()
        {
            TFracEditor
testClass = new
TFracEditor("123/123");

testClass.AddNumber(0);

testClass.AddNumber(1);

testClass.AddNumber(3);

testClass.AddSeparator();

testClass.ToggleMinus();
            string result = "-
1/1013";

Assert.AreEqual(result,
testClass.ToString());
        }
        [TestMethod]
        public void
AddDeleteTest2()
        {
            TFracEditor
testClass = new TFracEditor(123,
123);

testClass.RemoveSymbol();

testClass.RemoveSymbol();

testClass.RemoveSymbol();

testClass.RemoveSymbol();

testClass.RemoveSymbol();

testClass.RemoveSymbol();

testClass.RemoveSymbol();

testClass.AddNumber(1);

testClass.AddNumber(2);

testClass.AddNumber(3);

testClass.AddNumber(4);

testClass.AddNumber(5);

testClass.AddSeparator();
```

```
testClass.AddNumber(1);

testClass.AddNumber(1);

testClass.AddNumber(1);

testClass.AddNumber(1);
            string result =
"12345/1111";

Assert.AreEqual(result,
testClass.ToString());
        }

        [TestMethod]
        public void
AddDeleteTest3()
        {
            TFracEditor
testClass = new
TFracEditor(1234567, 12345678);
            for (int i = 0; i <
100; ++i)

testClass.RemoveSymbol();
            for (int i = 0; i <
100; ++i)

testClass.AddSeparator();

testClass.AddNumber(1);

testClass.AddNumber(1);

testClass.AddNumber(1);

testClass.AddNumber(1);
            string result =
"1111";

Assert.AreEqual(result,
testClass.ToString());
        }
        [TestMethod]
        public void
AddDeleteTest4()
        {
            TFracEditor
testClass = new
TFracEditor("0/1");
            for (int i = 0; i <
100; ++i)

testClass.AddNumber(i);
            string result =
"123456789";

Assert.AreEqual(result,
testClass.ToString());
        }
        [TestMethod]
        public void Clear()
        {
            TFracEditor
testClass = new
TFracEditor("2345678/345678");
            testClass.Clear();
            string result = "0";

Assert.AreEqual(result,
testClass.ToString());
        }
    }

    [TestClass]
    public class TPNumberTest
    {
        [TestMethod]
        public void Init1()
        {
            TPNumber tPNumber =
new TPNumber(10, 10, 3);
            string answer =
"10";

Assert.AreEqual(answer,
tPNumber.ToString());
        }
        [TestMethod]
        public void Init2()
        {
            TPNumber tPNumber =
new TPNumber("-12111.112", 3,
8);
            string answer = "-
12111.112";

Assert.AreEqual(answer,
tPNumber.ToString());
        }
        [TestMethod]
        public void Init3()
        {
            TPNumber tPNumber =
new TPNumber(16, 16, 3);
            string answer =
"10";

Assert.AreEqual(answer,
tPNumber.ToString());
        }
        [TestMethod]
        public void Init4()
        {
```

```
            TPNumber tPNumber =
new TPNumber(255, 2, 3);
            string answer =
"11111111";

Assert.AreEqual(answer,
tPNumber.ToString());
        }
        [TestMethod]
        public void Init5()
        {
            TPNumber tPNumber =
new TPNumber(14.471, 7, 5);
            string answer =
"20.32036";

Assert.AreEqual(answer,
tPNumber.ToString());
        }
        [TestMethod]
        public void Init6()
        {
            TPNumber tPNumber =
new TPNumber(145.1742, 15, 8);
            string answer =
"9A.292DD1D1";

Assert.AreEqual(answer,
tPNumber.ToString());
        }
        [TestMethod]
        public void Init7()
        {
            TPNumber tPNumber =
new TPNumber(36.09, 16, 2);
            string answer =
"24.17";

Assert.AreEqual(answer,
tPNumber.ToString());
        }
        [TestMethod]
        public void Init8()
        {
            TPNumber tPNumber =
new TPNumber(88.88, 8, 9);
            string answer =
"130.70243656";

Assert.AreEqual(answer,
tPNumber.ToString());
        }
        [TestMethod]
        public void Init9()
        {
            TPNumber tPNumber =
new TPNumber(256.256, 2, 5);
            string answer =
"100000000.01";

Assert.AreEqual(answer,
tPNumber.ToString());
        }
        [TestMethod]
        public void Init10()
        {
            TPNumber tPNumber =
new TPNumber(-356.22, 10, 5);
            string answer = "-
356.22";

Assert.AreEqual(answer,
tPNumber.ToString());
        }
        [TestMethod]
        public void Init11()
        {
            TPNumber tPNumber =
new TPNumber(0, 2, 1);
            string answer = "0";

Assert.AreEqual(answer,
tPNumber.ToString());
        }
        [TestMethod]
        public void Init12()
        {
            TPNumber tPNumber =
new TPNumber("A3.158", 12, 3);
            string answer =
"A3.158";

Assert.AreEqual(answer,
tPNumber.ToString());
        }
        [TestMethod]
        public void Add1()
        {
            TPNumber class1 =
new TPNumber(15, 7, 5);
            TPNumber class2 =
new TPNumber(66.66, 7, 5);
            class1 =
class1.Add(class2);
            double answer =
81.66;

Assert.AreEqual(answer,
class1.Number);
        }
        [TestMethod]
        public void Add2()
        {
            TPNumber class1 =
new TPNumber(12, 16, 10);
```

```
            TPNumber class2 =
new TPNumber(-13, 3, 6);
            class1 =
class1.Add(class2);
            double answer = 0;

Assert.AreEqual(answer,
class1.Number);
        }

        [TestMethod]
        public void Multiply1()
        {
            TPNumber class1 =
new TPNumber(12.23, 7, 5);
            TPNumber class2 =
new TPNumber(-6.66, 7, 5);
            class1 =
class1.Mul(class2);
            double answer = -
81.4518;

Assert.AreEqual(answer,
class1.Number);
        }
        [TestMethod]
        public void Multiply2()
        {
            TPNumber class1 =
new TPNumber(15.6, 16, 11);
            TPNumber class2 =
new TPNumber(0, 7, 5);
            class1 =
class1.Mul(class2);
            double answer = 0;

Assert.AreEqual(answer,
class1.Number);
        }

        [TestMethod]
        public void Substract1()
        {
            TPNumber class1 =
new TPNumber(12.23, 7, 5);
            TPNumber class2 =
new TPNumber(6.66, 7, 5);
            class1 =
class1.Sub(class2);
            double answer =
5.57;

Assert.AreEqual(answer,
class1.Number);
        }
        [TestMethod]
        public void Substract2()
        {
            TPNumber class1 =
new TPNumber(12.22, 16, 10);
            TPNumber class2 =
new TPNumber(-6.67, 7, 5);
            class1 =
class1.Sub(class2);
            double answer = 0;

Assert.AreEqual(answer,
class1.Number);
        }

        [TestMethod]
        public void Divide1()
        {
            TPNumber class1 =
new TPNumber(3, 7, 5);
            TPNumber class2 =
new TPNumber(8, 7, 5);
            class1 =
class1.Div(class2);
            double answer =
0.375;

Assert.AreEqual(answer,
class1.Number);
        }
        [TestMethod]
        public void Divide2()
        {
            TPNumber class1 =
new TPNumber(666, 16, 5);
            TPNumber class2 =
new TPNumber(-333, 2, 5);
            class1 =
class1.Div(class2);
            double answer = 0;

Assert.AreEqual(answer,
class1.Number);
        }

        [TestMethod]
        public void Square1()
        {
            TPNumber testClass =
new TPNumber(-1.57, 7, 5);
            testClass =
testClass.Square() as TPNumber;
            double answer =
2.4649;

Assert.AreEqual(answer,
testClass.Number);
        }
    }

    [TestClass]
```

```
    public class
TPNumberEditorTest
    {
        [TestMethod]
        public void Init1()
        {
            TPNumberEditor edit
= new TPNumberEditor(0.0f, 16,
5);
            string Input =
"2.3F";
            edit.Number = Input;
            string Output =
Input;

Assert.AreEqual(Output,
edit.Number);
        }
        [TestMethod]
        public void Init2()
        {
            TPNumberEditor edit
= new TPNumberEditor(0.0f, 2,
5);
            string Input =
"11101.01011";
            edit.Number = Input;
            string Output =
Input;

Assert.AreEqual(Output,
edit.Number);
        }
        [TestMethod]
        public void Init3()
        {
            TPNumberEditor edit
= new TPNumberEditor(0.0f, 16,
5);
            string Input = "-
FF.FF";
            edit.Number = Input;
            string Output =
Input;

Assert.AreEqual(Output,
edit.Number);
        }

[ExpectedException(typeof(Overfl
owException))]
        [TestMethod]
        public void Init4()
        {
            TPNumberEditor edit
= new TPNumberEditor(0.0f, 16,
5);
            string Input =
"aaaaaaaaaaaa";
            edit.Number = Input;
        }

[ExpectedException(typeof(Overfl
owException))]
        [TestMethod]
        public void Init5()
        {
            TPNumberEditor edit
= new TPNumberEditor(0.0f, 16,
5);
            string Input =
"16.ppppp";
            edit.Number = Input;
        }

[ExpectedException(typeof(Overfl
owException))]
        [TestMethod]
        public void Init6()
        {
            TPNumberEditor edit
= new TPNumberEditor(0.0f, 16,
5);
            string Input =
"FFZZZ";
            edit.Number = Input;
        }
        [TestMethod]
        public void Init7()
        {
            TPNumberEditor edit
= new TPNumberEditor(0.0f, 16,
5);
            string Input =
"888.8888";
            edit.Number = Input;
            string Output =
Input;

Assert.AreEqual(Output,
edit.Number);
        }
        [TestMethod]
        public void Init8()
        {
            TPNumberEditor edit
= new TPNumberEditor(0.0f, 16,
5);
            string Input = "-
01.03";
            edit.Number = Input;
            string Output = "-
1.03";
```

```
Assert.AreEqual(Output,
edit.Number);
        }
        [TestMethod]
        public void Init9()
        {
            TPNumberEditor edit
= new TPNumberEditor(0.0f, 16,
5);
            string Input = "-
9992D.DDDD";
            edit.Number = Input;
            string Output =
Input;

Assert.AreEqual(Output,
edit.Number);
        }
        [TestMethod]
        public void Init10()
        {
            TPNumberEditor edit
= new TPNumberEditor(0.0f, 16,
5);
            string Input =
"7777.7777";
            edit.Number = Input;
            string Output =
"7777.7777";

Assert.AreEqual(Output,
edit.Number);
        }

        [TestMethod]
        public void
AddDeleteTest1()
        {
            TPNumberEditor
testClass = new
TPNumberEditor();

testClass.AddNumber(0);

testClass.AddNumber(1);

testClass.AddNumber(3);

testClass.AddSeparator();

testClass.ToggleMinus();
            string result = "-
13.";

Assert.AreEqual(result,
testClass.ToString());
        }
        [TestMethod]
        public void
AddDeleteTest2()
        {
            TPNumberEditor
testClass = new
TPNumberEditor(0.0f, 16, 5);
            for (int i = 0; i <
16; ++i)

testClass.AddNumber(i);
            string result =
"123456789ABCDEF";

Assert.AreEqual(result,
testClass.ToString());
        }
        [TestMethod]
        public void
AddDeleteTest3()
        {
            TPNumberEditor
testClass = new
TPNumberEditor(0.0f, 16, 5);
            for (int i = 0; i <
100; ++i)

testClass.RemoveSymbol();

testClass.AddSeparator();

testClass.AddSeparator();

testClass.AddSeparator();

testClass.AddNumber(15);
            string result =
"0.F";

Assert.AreEqual(result,
testClass.ToString());
        }
        [TestMethod]
        public void
AddDeleteTest4()
        {
            TPNumberEditor
testClass = new
TPNumberEditor(0.0f, 2, 5);
            for (int i = 2; i <
100; ++i)

testClass.AddNumber(i);
            string result = "0";

Assert.AreEqual(result,
testClass.ToString());
        }
```

```
    }

    [TestClass]
    public class TComplexTest
    {
        [TestMethod]
        public void Init1()
        {
            TComplex testClass =
new TComplex(6, 3);
            string output = "6 +
i * 3";

Assert.AreEqual(output,
testClass.ToString());
        }
        [TestMethod]
        public void Init2()
        {
            TComplex testClass =
new TComplex(0, 0);
            string output = "0 +
i * 0";

Assert.AreEqual(output,
testClass.ToString());
        }
        [TestMethod]
        public void Init3()
        {
            TComplex testClass =
new TComplex(3, -2);
            string output = "3 +
i * -2";

Assert.AreEqual(output,
testClass.ToString());
        }
        [TestMethod]
        public void Init4()
        {
            TComplex testClass =
new TComplex(4.01, 6);
            string output =
"4,01 + i * 6";

Assert.AreEqual(output,
testClass.ToString());
        }
        [TestMethod]
        public void Init5()
        {
            TComplex testClass =
new TComplex(-3.02, 7);
            string output = "-
3,02 + i * 7";

Assert.AreEqual(output,
testClass.ToString());
        }
        [TestMethod]
        public void Init6()
        {
            TComplex testClass =
new TComplex(0, -22.22);
            string output = "0 +
i * -22,22";

Assert.AreEqual(output,
testClass.ToString());
        }

        [TestMethod]
        public void Reverse1()
        {
            TComplex testClass =
new TComplex(0, -3);
            testClass =
testClass.Reverse() as TComplex;
            string output = "0 +
i * 0,3333333333333333";

Assert.AreEqual(output,
testClass.ToString());
        }
        [TestMethod]
        public void Reverse2()
        {
            TComplex testClass =
new TComplex(3, 4);
            testClass =
testClass.Reverse() as TComplex;
            string output =
"0,12 + i * -0,16";

Assert.AreEqual(output,
testClass.ToString());
        }
        [TestMethod]
        public void Square1()
        {
            TComplex testClass =
new TComplex(3, 4);
            testClass =
testClass.Square() as TComplex;
            string output = "-7
+ i * 24";

Assert.AreEqual(output,
testClass.ToString());
        }
        [TestMethod]
        public void Square2()
        {
```

```
            TComplex testClass =
new TComplex(0, -3);
            testClass =
testClass.Square() as TComplex;
            string output = "-9
+ i * -0";

Assert.AreEqual(output,
testClass.ToString());
        }
        [TestMethod]
        public void Abs1()
        {
            TComplex testClass =
new TComplex(3, 4);
            double answer = 5;

Assert.AreEqual(answer,
testClass.Abs(), 4);
        }
        [TestMethod]
        public void Abs2()
        {
            TComplex testClass =
new TComplex(0, -3);
            double answer = 3;

Assert.AreEqual(answer,
testClass.Abs(), 4);
        }
        [TestMethod]
        public void GetRad1()
        {
            TComplex testClass =
new TComplex(3, 4);
            double answer =
0.927295;

Assert.AreEqual(answer,
testClass.Rad(), 4);
        }
        [TestMethod]
        public void GetRad2()
        {
            TComplex testClass =
new TComplex(0, -3);
            double answer = -
1.570796;

Assert.AreEqual(answer,
testClass.Rad(), 4);
        }
        [TestMethod]
        public void GetDegree1()
        {
            TComplex testClass =
new TComplex(3, 4);
            double answer =
53.1301;

Assert.AreEqual(answer,
testClass.Degree(), 4);
        }
        [TestMethod]
        public void GetDegree2()
        {
            TComplex testClass =
new TComplex(0, -3);
            double answer = -90;

Assert.AreEqual(answer,
testClass.Degree(), 4);
        }
        [TestMethod]
        public void Pow1()
        {
            TComplex testClass =
new TComplex(3, 4);
            testClass =
testClass.Pow(5);
            string output = "-
237 + i * -3116";

Assert.AreEqual(output,
testClass.ToString());
        }
        [TestMethod]
        public void Pow2()
        {
            TComplex testClass =
new TComplex(0, -3);
            testClass =
testClass.Pow(5);
            string output = "0 +
i * -243";

Assert.AreEqual(output,
testClass.ToString());
        }
        [TestMethod]
        public void Root1()
        {
            TComplex testClass =
new TComplex(3, 4);
            testClass =
testClass.Sqrt(5, 3);
            string output = "-
0.353 + i * 1.334";

Assert.AreEqual(output,
testClass.ToString());
        }
        [TestMethod]
        public void Root2()
        {
```

```
            TComplex testClass =
new TComplex(16, -2);
            testClass =
testClass.Sqrt(10, 4);
            string output = "-
0.301 + i * 1.286";

Assert.AreEqual(output,
testClass.ToString());
        }
    }

    [TestClass]
    public class
TComplexEditorTest
    {
        [TestMethod]
        public void Init1()
        {
            TComplexEditor
testClass = new
TComplexEditor();
            string input =
"10,3";
            testClass.Number =
input;
            string output =
"10,3 + i * 0";

Assert.AreEqual(output,
testClass.ToString());
        }
        [TestMethod]
        public void Init2()
        {
            TComplexEditor
testClass = new
TComplexEditor();
            string input = "10,
+ i * -12,";
            testClass.Number =
input;
            string output = "10
+ i * -12";

Assert.AreEqual(output,
testClass.ToString());
        }
        [TestMethod]
        public void Init3()
        {
            TComplexEditor
testClass = new
TComplexEditor();
            string input = "-0 +
i * -0";
            testClass.Number =
input;
            string output = "-0
+ i * -0";

Assert.AreEqual(output,
testClass.ToString());
        }
        [TestMethod]
        public void Init4()
        {
            TComplexEditor
testClass = new
TComplexEditor();
            string input =
"66,66 + i * 66,66";
            testClass.Number =
input;
            string output =
"66,66 + i * 66,66";

Assert.AreEqual(output,
testClass.ToString());
        }
        [TestMethod]
        public void Init5()
        {
            TComplexEditor
testClass = new
TComplexEditor();
            string input = "10.3
+ ";
            testClass.Number =
input;
            string output = "0 +
i * 0";

Assert.AreEqual(output,
testClass.ToString());
        }

        [TestMethod]
        public void
AddDeleteTest1()
        {
            TComplexEditor
testClass = new
TComplexEditor();

testClass.AddNumber(0);

testClass.ToggleMinus();

testClass.AddNumber(1);

testClass.AddNumber(3);

testClass.AddSeparator();
            string result = "-13
+ i * 0";
```

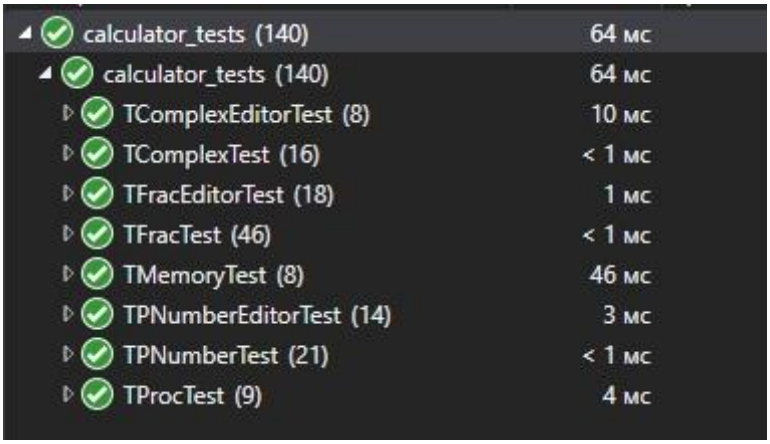
```
Assert.AreEqual(result,
testClass.ToString());
        }
        [TestMethod]
        public void
AddDeleteTest2()
        {
            TComplexEditor
testClass = new
TComplexEditor();

testClass.AddNumber(0);

testClass.AddNumber(1);

testClass.AddNumber(3);

testClass.AddSeparator();

testClass.ToggleMinus();

testClass.RemoveSymbol();
            string result = "13
+ i * -0";

Assert.AreEqual(result,
testClass.ToString());
        }
        [TestMethod]
        public void
AddDeleteTest3()
        {
            TComplexEditor
testClass = new
TComplexEditor();

testClass.AddNumber(0);

testClass.AddNumber(1);

testClass.AddNumber(3);

testClass.AddNumberSeparator();

testClass.AddSeparator();

testClass.ToggleMinus();

testClass.AddNumber(0);

testClass.AddNumber(1);

testClass.AddNumber(3);

testClass.AddNumberSeparator();
            string result = "13.
+ i * -13.";

Assert.AreEqual(result,
testClass.ToString());
        }
    }

    [TestClass]
    public class TMemoryTest
    {
        [TestMethod]
        public void
InitAndOutput1()
        {
            TFrac frac = new
TFrac(22, 33);
            TMemory<TFrac>
memory = new
TMemory<TFrac>(frac);
            string answer =
"2/3";

Assert.AreEqual(answer,
memory.FNumber.ToString());
        }
        [TestMethod]
        public void
InitAndOutput2()
        {
            TFrac frac = new
TFrac();
            TMemory<TFrac>
memory = new
TMemory<TFrac>(frac);
            string answer =
"0/1";

Assert.AreEqual(answer,
memory.FNumber.ToString());
        }
        [TestMethod]
        public void
InitAndOutput3()
        {
            TFrac frac = new
TFrac(-1, 5);
            TMemory<TFrac>
memory = new
TMemory<TFrac>(frac);
            string answer = "-
1/5";

Assert.AreEqual(answer,
memory.FNumber.ToString());
        }

        [TestMethod]
        public void Sum1()
        {
```

```
            TFrac frac = new
TFrac(-1, 5);
            TMemory<TFrac>
memory = new
TMemory<TFrac>(frac);
            TFrac summator = new
TFrac(1, 2);

memory.Add(summator);
            string answer =
"3/10";

Assert.AreEqual(answer,
memory.FNumber.ToString());
        }

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

memory.Add(summator);
            string answer = "-
40/9";

Assert.AreEqual(answer,
memory.FNumber.ToString());
        }

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

Assert.AreEqual(expected,
memory.FState);
        }

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

Assert.AreEqual(expected,
memory.FState);
        }

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

Assert.AreEqual(expected,
memory.FState);
        }
    }

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

Assert.AreEqual(answer,
proc.Left_Result_operand.ToStrin
g());

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

        [TestMethod]
        public void Init2()
        {
            TFrac leftFrac = new
TFrac(11, 3);
```

```
            TFrac rightFrac =
new TFrac();
            ADT_Proc<TFrac> proc
= new ADT_Proc<TFrac>(leftFrac,
rightFrac);
            string answer =
"11/3";

Assert.AreEqual(answer,
proc.Left_Result_operand.ToStrin
g());
        }

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

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

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

Assert.AreEqual(answer,
proc.Left_Result_operand.ToStrin
g());
        }

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

Assert.AreEqual(answer,
proc.Left_Result_operand.ToStrin
g());
        }

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

Assert.AreEqual(answer,
proc.Left_Result_operand.ToStrin
g());
        }

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

Assert.AreEqual(answer,
proc.Left_Result_operand.ToStrin
g());
        }
        [TestMethod]
        public void
TestFState1()
```

```
        {
            TFrac leftFrac = new
TFrac(56, 7);
            TFrac rightFrac =
new TFrac(-22, 3);
            ADT_Proc<TFrac> proc
= new ADT_Proc<TFrac>(leftFrac,
rightFrac);

proc.DoFunction(ADT_Proc<TFrac>.
Functions.Rev);
            string answer = "-
3/22";

Assert.AreEqual(answer,
proc.Right_operand.ToString());
        }
        [TestMethod]
        public void
TestFState2()
        {
            TFrac leftFrac = new
TFrac(56, 7);
            TFrac rightFrac =
new TFrac(-22, 3);
            ADT_Proc<TFrac> proc
= new ADT_Proc<TFrac>(leftFrac,
rightFrac);

proc.DoFunction(ADT_Proc<TFrac>.
Functions.Sqr);
            string answer =
"484/9";

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

Результат работы тестов



Вывод

По итогам данного проекта было проведено проектирование программы в технологии «абстрактных типов данных», и «объектно-ориентированного программирования». Произведена реализация абстрактных типов данных с помощью классов C#: простые дроби, р-ичные числа и комплесные числа. 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.