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.

# Содержание

Цель	3
Задание	4
Диаграмма классов для прецедентов	10
Спецификации к типам данных	11
Результаты тестирования программы	26
Листинг	28
Листинг тестов	64
Результат работы тестов	84
Вывод	85

# 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 С#, С++;
- 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 С#, С++ 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	6/1	36/1	36/1	2/1	4/1	40/1	10/1	4/1	6/1	10/1
результат										

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 [-][< р ичная дробь без знака >] Необходимо обеспечить возможность работы в режимах:
  - «целые» (вводятся только р-ичные целые числа),
  - «действительные» (вводятся р-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 (
$$p = 10$$
)

б. 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>

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

<pазделитель>::= '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: Рwr 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:

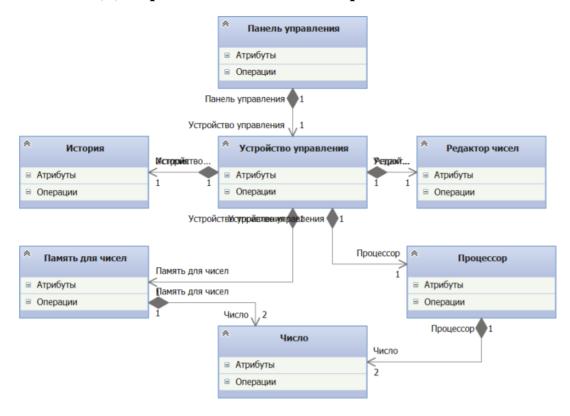
«операнд1» «функция1» «операция1» «операнд2» «функция2» «операция2» ...«операндN» «операцияN» «=» «результат»

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: р-ичное число, простая дробь, комплексное число.

Мой вариант: р-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 – ( $c \ge 0$ ). 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 (а), основание системы счисления (b), точность представления числа (c)
Предусловия:	Основание системы счисления (b) должно принадлежать интервалу [216], точность представления числа $c \ge 0$ .
Процесс:	Инициализирует поля объекта this p-ичное число: система счисления (b), точность представления (c).
	В поле (n) числа заносится (a).
	Например:
	TPNumber(a,3,3) = число а в системе счисления 3 с тремя разрядами после троичной точки.
	TPNumber (a,3,2) = число а в системе счисления 3 с двумя разрядами после троичной точки.
Постусловия:	Объект инициализирован начальными значениями.
Выход:	Нет.
КонструкторСтрока	

Вход:	Строковые представления: р—ичного числа (а), основания системы счисления (b), точности представления числа (с).
Предусловия:	Основание системы счисления (b) должно принадлежать интервалу [216], точность представления числа с $>= 0$
Процесс:	Инициализирует поля объекта this р-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:  ТРNumber ("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:	р-ичное число.
Постусловия:	Нет.
Умножить	
Вход:	Р-ичное число d с основанием и точностью такими же, как у самого числа this.
Предусловия:	Нет.
Процесс:	Создаёт и возвращает р-ичное число (тип TPNumber), полученное умножением полей (n) самого числа this и числа d.
Выход:	Р-ичное число (тип TPNumber).
Постусловия:	Нет.
Вычесть	

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

Выход:	Р-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 а, 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).
	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.
Взять Числитель Число	
Вход:	Нет.
Предусловия:	Нет.
Процесс:	Возвращает значение числителя дроби в числовом формате.
Выход:	Вещественное значение.
Постусловия:	Нет.
Взять Знаменатель Число	
Вход:	Нет.
Предусловия:	Нет.
Процесс:	Возвращает значение знаменателя дроби в числовом формате.
Выход:	Вещественное значение.
Постусловия:	Нет.
Взять Числитель Строка	
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 а 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 (а) 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.
16.)	
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.

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:	УголРад	
Процесс:  Возвращает аргумент fi самого комплексного числа this (в радманах). fi = (агсТg(b/a), a>0; pi/2, a = 0, b>0; агсТg(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.  Предусловия: Нет.  Процесс: Возвращает аргумент самого комплексного числа this (в градусах). Например: q = (1 + i*1), q. Град = 45. Выход: Вещественное число. Постусловия: Нет.  Карень Вход: Нет.  Процесс: Везиращает аргумент самого комплексного числа this (в градусах). Например: q = (1 + i*1), q. Град = 45. Выход: Нет.  Карень Вход: Нет.  Карень Вход: Нет.  Процесс: Возвращает i-ый корень целой положительной степени п самого комплексного числа this. ∨ n (this) = ∨ n (r)* (cos (fi + 2*k*pi)/n)+ i*s in (fi + 2*k*pi)/n). При этом коофиценту к придается последовательно п значений: k = 0,1,2, n - 1 и получают п значений корня, т.е. ровно столько, каков показатель корня Комплексное число (тип ТСотрleх).	Вход:	Нет.
радианах). fi = (агсТg(b/a), a>0; pi/2, a = 0, b > 0; aгсТg(b/a) + pi, a < 0; -pi/2, a = 0, b < 0).  Например:	Предусловия:	Нет.
Например:	Процесс:	Возвращает аргумент fi самого комплексного числа this (в радианах). fi = (arcTg(b/a), a>0; pi/2, a = 0, b > 0;
q = (1 + i*1), q. УголРад = 0,79.         Выход:         Нет.         Иредусловия:         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         Возвращает і-ый корень целой положительной степени п самого комплексного числа this. √ n (this) = √ n (r)* (сох ((fi + 2*k*pi)/m)-i* sin ((fi + 2*k*pi)/m). При этом коофициенту к придвется последовательно п значений: k = 0,1,2, n - 1 и получают п значений корня, т.е. ровно столько, каков показатель корня         Выход:         Комплексное число (тип ТСотрlex).		arcTg(b/a) + pi, a < 0; -pi/2, a = 0, b < 0).
Выход:         Вещественное число.           Постусловия:         Нет.           Вход:         Нет.           Предусловня:         Нет.           Процесе:         Возвращает аргумент самого комплексного числа this (в градусах).           Например:         q = (1 + i*1), q. Град = 45.           Выход:         Вещественное число.           Постусловия:         Нет.           Корепь         Возвращает і-ый корень целой положительной степени п самого комплексного числа this. √ n (this) = √ n (r)* (cos ((fi + 2*k*pi)/n)). При этом коофициенту к придается последовательно п значений: k = 0,1,2, n - 1 и получают п значений корня, т.е. ровно столько, каков показатель корня           Выход:         Комплексное число (тип ТСотрlex).		Например:
УголГраф         Нет.           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         Возвращает і-ый корень целой положительной степени п самого комплексного числа this. √ n (this) = √ n (r)* (cos ((fi + 2*k*pi)/n)). При этом коофициенту к придается последовательно п значений: k = 0,1,2, n - 1 и получают п значений корня, т.е. ровно столько, каков показатель корня           Выход:         Комплексное число (тип ТСотрlex).		q = (1 + i*1), q. Угол $P$ ад $= 0,79.$
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         Целое п, целое і.           Предусловия:         Нет.           Процесс:         Возвращает і-ый корень целой положительной степени п самого комплексного числа this. √ п (this) = √ п (r)* (cos ((fi + 2*k*pi)/m). При этом коофициенту к придается последовательно п значений: k = 0,1,2, n - 1 и получают п значений корня, т.е. ровно столько, каков показатель корня           Выход:         Комплексное число (тип ТСотрlex).	Выход:	Вещественное число.
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       Целое п, целое і.         Предусловия:       Нет.         Процесс:       Возвращает і-ый корень целой положительной степени п самого комплексного числа this. √ n (this) = √ n (r)* (соя ((fi + 2*k*pi)/n))+i* sin ((fi + 2*k*pi)/n)). При этом коофициенту к придается последовательно п значений: k = 0,1,2, n - 1 и получают п значений кория, т.е. ровно столько, каков показатель корня         Выход:       Комплексное число (тип ТСотрlex).	Постусловия:	Нет.
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       Целое п, целое і.         Предусловия:       Нет.         Процесс:       Возвращает і-ый корень целой положительной степени п самого комплексного числа this. √ n (this) = √ n (r)* (соя ((fi + 2*k*pi)/n))+i* sin ((fi + 2*k*pi)/n)). При этом коофициенту к придается последовательно п значений: k = 0,1,2, n - 1 и получают п значений кория, т.е. ровно столько, каков показатель корня         Выход:       Комплексное число (тип ТСотрlex).		
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 п, целое і.         Предусловия:       Нет.         Процесс:       Возвращает і-ый корень целой положительной степени п самого комплексного числа this. √ п (this) = √ п (r)* (соз ((fi + 2*k*pi)/n)+i* sin ((fi + 2*k*pi)/n)). При этом коофициенту к придается последовательно п значений: k = 0,1,2, n - 1 и получают п значений корня, т.е. ровно столько, каков показатель корня         Выход:       Комплексное число (тип ТСотрlex).	УголГрад	
Процесс:     Возвращает аргумент самого комплексного числа this (в градусах).     Например:	Вход:	Нет.
градусах).  Например: $q = (1 + i*1), q. \Gamma pad = 45.$ Выход: Вещественное число.  Постусловия: Нет.  Корень  Вход: Целое п, целое i.  Предусловия: Нет.  Процесс: Возвращает i-ый корень целой положительной степени п самого комплексного числа this. $\sqrt{n}$ (this) = $\sqrt{n}$ (r)* (cos ((fi + 2*k*pi)/n)+ i* sin ((fi + 2*k*pi)/n)). При этом коофициенту к придается последовательно п значений: $k = 0,1,2, n-1$ и получают п значений корня, т.е. ровно столько, каков показатель корня  Выход: Комплексное число (тип TComplex).	Предусловия:	Нет.
$q=(1+i*1), q. \Gamma pag=45.$ Выход: Вещественное число.  Постусловия: Нет.  Корень  Вход: Целое п, целое і.  Предусловия: Нет.  Процесс: Возвращает і-ый корень целой положительной степени п самого комплексного числа this. $\sqrt{n}$ (this) = $\sqrt{n}$ (r)* (cos ((fi + 2*k*pi)/n)+ i* sin ((fi + 2*k*pi)/n)). При этом коофициенту к придается последовательно п значений: $k=0,1,2,n-1$ и получают п значений корня, т.е. ровно столько, каков показатель корня  Выход: Комплексное число (тип TComplex).	Процесс:	Возвращает аргумент самого комплексного числа this (в градусах).
Выход:         Вещественное число.           Постусловия:         Нет.           Вход:         Целое п, целое і.           Предусловия:         Нет.           Процесс:         Возвращает і-ый корень целой положительной степени п самого комплексного числа this. √ n (this) = √ n (r)* (соѕ ((fi + 2*k*pi)/n)+ i* sin ((fi + 2*k*pi)/n)). При этом коофициенту к придается последовательно п значений: k = 0,1,2, n - 1 и получают п значений корня, т.е. ровно столько, каков показатель корня           Выход:         Комплексное число (тип ТСотрlex).		Например:
Постусловия: Нет.  Вход: Целое п, целое і.  Предусловия: Нет.  Процесс: Возвращает і-ый корень целой положительной степени п самого комплексного числа this. $\sqrt{n}$ (this) = $\sqrt{n}$ (r)* (cos ((fi + 2*k*pi)/n)+ і* sin ((fi + 2*k*pi)/n)). При этом коофициенту к придается последовательно п значений: $k = 0,1,2, n-1$ и получают п значений корня, т.е. ровно столько, каков показатель корня  Выход: Комплексное число (тип TComplex).		$q = (1 + i*1), q. \Gamma pa \pi = 45.$
Корень       Вход:       Целое п, целое і.         Предусловия:       Нет.         Процесс:       Возвращает і-ый корень целой положительной степени п самого комплексного числа this. $\sqrt{n}$ (this) = $\sqrt{n}$ ( $r$ )* (cos ((fi + 2*k*pi)/n)+ i* sin ((fi + 2*k*pi)/n)). При этом коофициенту к придается последовательно п значений: $k = 0,1,2, n-1$ и получают п значений корня, т.е. ровно столько, каков показатель корня         Выход:       Комплексное число (тип ТСотрlex).	Выход:	Вещественное число.
Вход: Целое п, целое і.  Предусловия: Нет.  Возвращает і-ый корень целой положительной степени п самого комплексного числа this. $\sqrt{n}$ (this) = $\sqrt{n}$ (r)* (cos ((fi + 2*k*pi)/n)+ і* sin ((fi + 2*k*pi)/n)). При этом коофициенту k придается последовательно п значений: k = 0,1,2, n - 1 и получают п значений корня, т.е. ровно столько, каков показатель корня  Выход: Комплексное число (тип TComplex).	Постусловия:	Нет.
Вход: Целое п, целое і.  Предусловия: Нет.  Возвращает і-ый корень целой положительной степени п самого комплексного числа this. $\sqrt{n}$ (this) = $\sqrt{n}$ (r)* (cos ((fi + 2*k*pi)/n)+ і* sin ((fi + 2*k*pi)/n)). При этом коофициенту k придается последовательно п значений: k = 0,1,2, n - 1 и получают п значений корня, т.е. ровно столько, каков показатель корня  Выход: Комплексное число (тип TComplex).		
Процесс: Возвращает і-ый корень целой положительной степени п самого комплексного числа this. $\sqrt{n}$ (this) = $\sqrt{n}$ (r)* (cos ((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).	Корень	
Процесс: Возвращает і-ый корень целой положительной степени п самого комплексного числа this. $\sqrt{n}$ (this) = $\sqrt{n}$ (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).	Вход:	Целое n, целое i.
самого комплексного числа this. $\sqrt{n}$ (this) = $\sqrt{n}$ (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).	Предусловия:	Нет.
	Процесс:	и получают n значений корня, т.е. ровно столько, каков
Постусловия: Нет.	Выход:	Комплексное число (тип TComplex).
	Постусловия:	Нет.

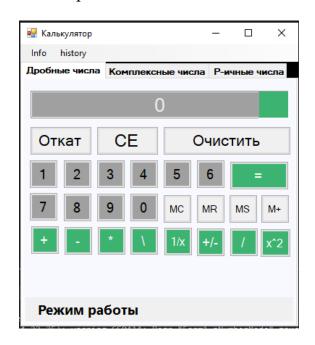
Равно	
Вход:	Комплексное число d.
Предусловия:	Нет.
Процесс:	Сравнивает само комплексное число this с числом d. Возвращает значение True, если они — тождественные комплексные числа, и значение False - в противном случае.
Выход:	Булевское значение.
Постусловия:	Нет.
Взять Re Число	
Вход:	Нет.
Предусловия: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

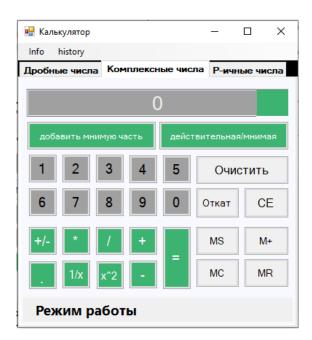
Выход:	Строка.
Постусловия:	Нет.
ВзятьІтСтрока	
Вход:	Нет.
Предусловия:	Нет.
Процесс:	Возвращает значение мнимой части самого комплексного числа this в строковом формате.
Выход:	Строка.
Постусловия:	Нет.
ВзятьКомплексноеСтрока	
Вход:	Нет.
Предусловия:	Нет.
Процесс:	Возвращает значение самого комплексного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

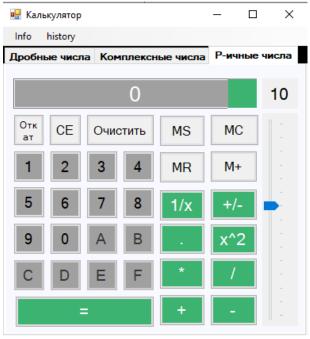
End TComplex

## Результаты тестирования программы

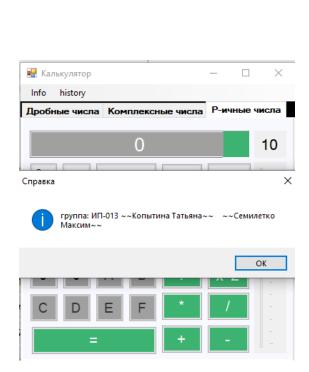
## Начало работы:

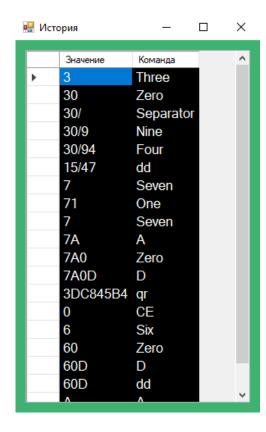






info: history:





#### Листинг

```
ADT_Control_.cs:
                                                             memory = value;
using System;
                                                     }
using System.Collections.Generic;
using System.Linq;
                                                    public TEditor Edit
using System. Text;
using System. Threading. Tasks;
                                                         get
                                                         {
namespace calculator
                                                             return editor;
    public class ADT Control<T,</pre>
                                                         set
                                                         {
        where T : ANumber, new()
                                                             editor = value;
        where TEditor: AEditor,
new()
                                                     }
        public enum
                                                    public ADT Control()
ADT Control State { cStart,
cEditing, FunDone, cValDone,
                                                         Edit = new TEditor();
cExpDone, cOpDone, cOpChange, cError
                                                        Proc = new
                                            ADT Proc<T>();
                                                        Memory = new
        ADT Control State calcState;
                                            TMemory<T>();
        TEditor editor;
                                                         CurState =
        ADT_Proc<T> proc;
                                            ADT Control State.cStart;
        TMemory<T> memory;
        public THistory history =
new THistory();
                                                    public string Reset()
        public ADT Control State
                                                         Edit.Clear();
CurState
                                                         Proc.ResetProc();
                                                        Memory.Clear();
            get
                                                         CurState =
                                            ADT_Control_State.cStart;
                return calcState;
                                                         return Edit.ToString();
            }
            set
             {
                                                    public string
                calcState = value;
                                            ExecCommandEditor(AEditor.Command
                                            command)
        public ADT Proc<T> Proc
                                                        string toReturn;
                                                         if (CurState ==
            get
                                            ADT_Control_State.cExpDone)
                return proc;
                                                             Proc.ResetProc();
            }
                                                            CurState =
            set
                                            ADT Control State.cStart;
                                                         }
                proc = value;
                                                         if (CurState !=
                                            ADT Control State.cStart)
        }
                                                            CurState =
                                            ADT Control State.cEditing;
        public TMemory<T> Memory
                                                        toReturn =
                                            Edit.Edit(command);
            get
                                                         T \text{ tmp} = \text{new } T();
                                                         if (tmp is TPNumber)
                return memory;
            }
                                                             dynamic a = tmp;
            set
```

```
dynamic b = Edit;
                                          Edit.Clear();
                a.Notation =
b.Notation;
                                                                   CurState =
                                          ADT Control State.cOpChange;
                tmp = a;
            tmp.SetString(toReturn);
                                          Proc.Right operand =
                                          Proc.Left Result operand;
            proc.Right operand =
tmp;
                                                                   break;
                                                               case
history.AddRecord(toReturn,
                                          ADT Control State.cOpDone:
command.ToString());
                                                                   CurState =
                                          ADT Control State.cOpChange;
            return toReturn;
        }
                                          Edit.Clear();
                                                                   break;
        public string
                                                               case
ExecOperation(ADT Proc<T>.Operations
                                          ADT Control State.cValDone:
operation)
            if (operation ==
                                          ADT Control State.cExpDone:
ADT Proc<T>.Operations.None)
                return Edit. Number;
                                          Proc.Operation = operation;
            string toReturn;
            try
                                           Proc.Right operand =
                                          Proc.Left Result_operand;
                switch (CurState)
                                                                   CurState =
                                          ADT Control State.cOpChange;
                    case
ADT Control State.cStart:
                                          Edit.Clear();
                                                                   break;
Proc.Left Result operand =
Proc.Right operand;
                                          ADT_Control_State.cOpChange:
                                          Proc.Operation = operation;
Proc.Operation = operation;
                        CurState =
ADT Control State.cOpDone;
                                          Edit.Clear();
                                                                   break:
Edit.Clear();
                                                               case
                                          ADT Control State.cError:
                        break;
                    case
ADT Control State.cEditing:
                                          Proc.ResetProc();
                                                                   return
                                           "ERR";
Proc.DoOperation();
                                                           }
Proc.Operation = operation;
                                                           toReturn =
                                           Proc.Left Result operand.ToString();
Edit.Clear();
                                                       }
                        CurState =
                                                       catch
ADT Control State.cOpDone;
                                                       {
                                                           Reset();
                                                           return "ERROR";
                    case
ADT Control State.FunDone:
(Proc.Operation == 0)
                                          history.AddRecord(toReturn,
                                           operation.ToString());
Proc.Left Result operand =
Proc.Right operand;
                                                       return toReturn;
                        else
Proc.DoOperation();
                                                   public string
                                          ExecFunction(ADT Proc<T>.Functions
Proc.Operation = operation;
                                          function)
```

```
string toReturn;
                                                            ToReturn =
            try
                                            Proc.Left Result operand.ToString();
                                                        }
                if (CurState ==
                                                        catch
ADT Control State.cExpDone)
                                                            Reset();
                                                            return "ERROR";
Proc.Right_operand =
Proc.Left_Result_operand;
                     Proc.Operation =
                                                        return ToReturn;
ADT Proc<T>.Operations.None;
                }
                                                    public (T, bool)
                                            {\tt ExecCommandMemory(TMemory<T>.Command}
Proc.DoFunction(function);
                CurState =
                                            s command, string str)
ADT_Control_State.FunDone;
                                                        T \text{ TempObj} = \text{new } T();
                toReturn =
                                                        TempObj.SetString(str);
Proc.Right_operand.ToString();
            }
                                                        (T, bool) obj = (null,
            catch
                                            false);
                                                        try
            {
                Reset();
                                                        {
                return "ERROR";
                                                            obj =
                                            Memory.Edit(command, TempObj);
                                                        }
history.AddRecord(toReturn,
                                                        catch
function.ToString());
                                                            Reset();
            return toReturn;
                                                            return obj;
        }
                                                        if (command ==
        public string Calculate()
                                            TMemory<T>.Commands.Copy)
                                                        {
            string ToReturn;
                                                            Edit.Number =
                                            obj.Item1.ToString();
            try
                                                            Proc.Right operand =
                if (CurState ==
                                            obj.Item1;
ADT_Control_State.cStart)
                                                        return obj;
Proc.Left Result operand =
Proc.Right operand;
                                                }
                Proc.DoOperation();
                CurState =
ADT Control_State.cExpDone;
Edit.SetEditor(Proc.Left Result oper
and);
```

```
ADT_Proc.cs:
                                                             operation = value;
using System;
                                                     }
using System.Collections.Generic;
using System.Linq;
                                                     public ADT Proc()
using System. Text;
using System. Threading. Tasks;
                                                         operation =
                                            Operations.None;
namespace calculator
                                                         left result operand =
                                            new T();
    public class ADT Proc<T> where T
                                                         right operand = new T();
: ANumber, new()
        public enum Operations
                                                    public ADT Proc(T leftObj, T
                                            rightObj)
            None, Add, Sub, Mul, Div
                                                         operation =
        public enum Functions
                                            Operations.None;
                                                         left result operand =
            Rev, Sqr
                                            leftObj;
                                                         right operand =
                                            rightObj;
        T left result operand;
        T right operand;
        Operations operation;
                                                    public void ResetProc()
        public T Left Result operand
                                                         operation =
                                            Operations.None;
            get
                                                         T \text{ newObj} = \text{new } T();
            {
                                                         left result operand =
                return
                                            right operand = newObj;
left result operand;
            }
                                                    public void DoOperation()
            set
                                                         try
                left_result_operand
= value;
                                                             dynamic a =
             }
                                            left result operand;
                                                             dynamic b =
        public T Right operand
                                            right operand;
                                                             switch (operation)
            get
                                                                 case
                 return
                                            Operations.Add:
right operand;
                                            left result operand = a.Add(b);
                                                                     break;
            set
                                                                 case
                                            Operations.Sub:
                 right operand =
value;
                                            left result operand = a.Sub(b);
             }
                                                                     break;
                                                                 case
                                            Operations.Mul:
        public Operations Operation
                                            left result operand = a.Mul(b);
            get
                                                                     break;
             {
                                                                 case
                return operation;
                                            Operations.Div:
                                            left result operand = a.Div(b);
```

set

```
break;
                                                       switch (function)
                    default:
                                                           case Functions.Rev:
left_result_operand = right_operand;
                                                              a = a.Reverse();
                        break;
                                                               right_operand =
                                           (T)a;
                }
            }
                                                               break;
                                                           case Functions.Sqr:
            catch
                                                               a = a.Square();
                throw new
                                                               right_operand =
System.OverflowException();
                                          (T)a;
                                                               break;
                                                           default:
                                                               break;
        public void
                                                       }
DoFunction (Functions function)
                                                   }
                                              }
            dynamic a =
                                           }
right operand;
```

#### AEditor.cs:

```
using System;
using System.Collections.Generic;
using System.Ling;
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:
```

```
complex)
using System;
                                                        real = complex.real;
using System.Collections.Generic;
                                                        imaginary =
using System.Linq;
                                            complex.imaginary;
using System. Text;
using
System.Text.RegularExpressions;
                                                    public TComplex(string str)
using System. Threading. Tasks;
                                                        Regex FullNumber = new
namespace calculator
                                            Regex (@"^-
                                            ?(\d+.?\d*)\s+\+\s+i\s+\*\s+-
    public class TComplex : ANumber
                                            ?(\d+.?\d*)$");
                                                        Regex LeftPart = new
        private double real;
                                            Regex (@"^-
        private double imaginary;
                                            ?(\d+.?\d*)(\s+\+\s+i\s+\*\s+)?");
        const string Separator = " +
                                            (FullNumber.IsMatch(str))
        public double Real
                                                            List<string> Parts =
                                            str.Split(new string[] { Separator
            get
                                            StringSplitOptions.None).ToList();
                return real;
            }
                                                            real =
            set
                                            Double.Parse(Parts[0]);
            {
                                                            imaginary =
                real = value;
                                            Double.Parse(Parts[1]);
        }
                                                        else if
                                            (LeftPart.IsMatch(str))
        public double Imaginary
        {
                                                             i f
                                            (str.Contains(Separator))
            get
            {
                                            str.Replace (Separator,
                return imaginary;
                                            string.Empty);
            }
                                                             real =
            set
                                            Double.Parse(str);
                                                            imaginary = 0;
                imaginary = value;
                                                        else
        }
                                                             real = 0;
        public TComplex()
                                                             imaginary = 0;
            real = 0;
            imaginary = 0;
        }
                                                    public TComplex Copy()
        public TComplex(int a, int
                                                        return
b)
                                            (TComplex) this. MemberwiseClone();
        {
            real = a;
            imaginary = b;
                                                    public TComplex Add(TComplex
                                            b)
        public TComplex (double a,
                                                        TComplex res =
double b)
                                            this.Copy();
                                                        res.real += b.real;
            real = a;
                                                        res.imaginary +=
            imaginary = b;
                                            b.imaginary;
        }
```

public TComplex (TComplex

```
res.real = this.real /
           return res;
                                          (this.real * this.real +
        }
                                          this.imaginary * this.imaginary);
        public TComplex Sub(TComplex
                                                      res.imaginary = -
b)
                                          this.imaginary / (this.real *
                                          this.real + this.imaginary *
            TComplex res =
                                          this.imaginary);
                                                     return res;
this.Copy();
            res.real -= b.real;
            res.imaginary -=
b.imaginary;
                                                  public TComplex Minus()
           return res;
                                                      TComplex res =
                                          this.Copy();
        public TComplex Mul(TComplex
                                                      res.real = 0 - res.real;
b)
                                                      res.imaginary = 0 -
                                          res.imaginary;
           TComplex res =
                                                     return res;
this.Copy();
           res.real = this.real *
b.real - this.imaginary *
                                                  public double Abs()
b.imaginary;
           res.imaginary =
                                                      return
this.real * b.imaginary +
                                         Math.Sqrt(this.real * this.real +
this.imaginary * b.real;
                                         this.imaginary * this.imaginary);
           return res;
                                                  public double Rad()
       public TComplex Div(TComplex
                                                      if (this.real > 0)
b)
                                                          return
           TComplex res =
                                          Math.Atan(this.imaginary /
this.Copy();
                                          this.real);
            res.real = (this.real *
b.real + this.imaginary *
                                                     if (this.real == 0 &&
b.imaginary) / (b.real * b.real +
                                         this.imaginary > 0)
b.imaginary * b.imaginary);
                                                          return (Math.PI /
           res.imaginary = (b.real
                                          2);
* this.imaginary - this.real *
b.imaginary) / (b.real * b.real +
                                                      if (this.real < 0)</pre>
b.imaginary * b.imaginary);
                                                          return
           return res;
                                          (Math.Atan(this.imaginary /
                                          this.real) + Math.PI);
        public TComplex Square()
                                                     if (this.real == 0 &&
                                          this.imaginary < 0)</pre>
            TComplex res =
                                                          return (-Math.PI /
this.Copy();
                                          2);
           res.real = this.real *
this.real - this.imaginary *
                                                     return 0;
this.imaginary;
           res.imaginary =
this.real * this.imaginary +
                                                  public double Degree()
this.real * this.imaginary;
           return res;
                                                     return Rad() * 180 /
                                          Math.PI;
        public TComplex Reverse()
                                                  public TComplex Pow(int n)
            TComplex res =
                                                     TComplex res =
this.Copy();
                                         this.Copy();
```

```
res.real =
                                                   return this.real;
Math.Pow(Abs(), n) * Math.Cos(n *
Rad());
                                                public double
           res.imaginary =
Math.Pow(Abs(), n) * Math.Sin(n * GetImaginaryNumber()
Rad());
                                                    return this.imaginary;
           return res;
       }
       public TComplex Sqrt(int
                                                public string
powN, int rootI)
                                         GetRealString()
           if (rootI >= powN ||
                                                    return
rootI < 0 || powN < 0)
                                         this.real.ToString();
              return new
                                               }
TComplex();
                                                public string
           return new
TComplex(Math.Pow(Abs(), 1.0 / powN)
                                         GetImaginaryString()
* Math.Cos((Degree() + 2 * Math.PI *
rootI) / powN), Math.Pow(Abs(), 1.0
                                                    return
/ powN) * Math.Sin((Degree() + 2 *
                                         this.imaginary.ToString();
Math.PI * rootI) / powN));
                                                public override void
       public bool Equal(TComplex
                                         SetString(string str) {
anClass)
                                                   TComplex temp = new
                                         TComplex(str);
           return (this.real ==
                                                    Real = temp.Real;
anClass.real && this.imaginary ==
                                                    Imaginary =
anClass.imaginary);
                                         temp.Imaginary;
       }
       public bool
                                                public override string
NotEqual (TComplex anClass)
                                         ToString()
                                                {
           return (this.real !=
                                                    return GetRealString() +
anClass.real || this.imaginary !=
                                         " + i * " + GetImaginaryString();
anClass.imaginary);
                                             }
       public double
GetRealNumber()
```

{

```
public TFrac()
TFrac.cs:
                                                         numerator = 0;
using System;
                                                         denominator = 1;
using System.Collections.Generic;
using System.Linq;
using
                                                    public TFrac(long a, long b)
System.Text.RegularExpressions;
                                                         if (a < 0 \&\& b < 0)
namespace rgz
                                                             a *= -1;
    public class TFrac
                                                             b *= -1;
        private long numerator;
                                                         else if (b < 0 \&\& a > 0)
        private long denominator;
                                                             b *= -1;
        /// Числитель
                                                             a *= -1;
        public long Numerator
                                                         else if (a == 0 \&\& b ==
            get
                                            0 \mid \mid b == 0 \mid \mid a == 0 \&\& b == 1)
            {
                return numerator;
                                                             numerator = 0;
                                                             denominator = 1;
            set
                                                             return;
            {
                numerator = value;
                                                         numerator = a;
                                                         denominator = b;
        }
                                                         long gcdRes = GCD(a, b);
                                                         if (gcdRes > 1)
        /// Знаменатель
        public long Denominator
                                                             numerator /= gcdRes;
        {
                                                             denominator /=
            get
                                            qcdRes;
             {
                return denominator;
            }
            set
                                                    public TFrac(string frac)
                denominator = value;
                                                         Regex FracRegex = new
            }
                                            Regex(@"^-?(\d+)/(\d+)$");
        }
                                                         Regex NumberRegex = new
                                            Regex(@"^-?\d+/?$");
        static void Swap<T>(ref T
                                                         if
lhs, ref T rhs)
                                            (FracRegex.IsMatch(frac))
                                                         {
            T temp;
                                                             List<string>
            temp = lhs;
                                            FracSplited =
            lhs = rhs;
                                            frac.Split('/').ToList();
            rhs = temp;
                                                             numerator =
        }
                                            Convert.ToInt64(FracSplited[0]);
                                                             denominator =
        public static long GCD(long
                                            Convert.ToInt64(FracSplited[1]);
a, long b)
                                                             if (denominator ==
                                            0)
            a = Math.Abs(a);
                                                             {
            b = Math.Abs(b);
                                                                 numerator = 0;
            while (b > 0)
                                                                 denominator = 1;
             {
                                                                 return;
                a %= b;
```

long gcd =

if  $(\gcd > 1)$ 

GCD (numerator, denominator);

Swap(ref a, ref b);

return a;

}

```
numerator /=
                                                   return new
                                        TFrac(numerator * b.denominator -
qcd;
                                        denominator * b.numerator,
                   denominator /=
                                        denominator * b.denominator);
qcd;
               }
               return;
                                                public TFrac Div(TFrac b)
           }
           else if
(NumberRegex.IsMatch(frac))
                                                   return new
                                        TFrac(numerator * b.denominator,
           {
               if
                                        denominator * b.numerator);
(long.TryParse(frac, out long
NewNumber))
                  numerator =
                                                public TFrac Square()
NewNumber;
               else
                                                   return new
                  numerator = 0;
                                       TFrac(numerator * numerator,
               denominator = 1;
                                        denominator * denominator);
               return;
                                                }
           }
           else
                                                public TFrac Reverse()
               numerator = 0;
                                                   return new
               denominator = 1;
                                      TFrac(denominator, numerator);
               return;
                                                public TFrac Minus()
       public TFrac Copy()
                                                   return new TFrac(-
                                        numerator, denominator);
           return
(TFrac) this. MemberwiseClone();
                                                public bool Equal(TFrac b)
       public void SetString(string
                                                   return numerator ==
                                        b.numerator && denominator ==
str)
                                        b.denominator;
           TFrac TempFrac = new
TFrac(str);
                                                public static bool operator
           numerator =
                                        >(TFrac a, TFrac b)
TempFrac.numerator;
           denominator =
TempFrac.denominator;
                                                   return
                                        (Convert.ToDouble(a.numerator) /
                                        Convert.ToDouble(a.denominator)) >
       public TFrac Add(TFrac a)
                                        (Convert.ToDouble(b.numerator) /
                                        Convert.ToDouble(b.denominator));
           return new
TFrac(numerator * a.denominator +
denominator * a.numerator,
                                               public static bool operator
denominator * a.denominator);
                                        <(TFrac a, TFrac b)
                                                   return
       public TFrac Mul(TFrac b)
                                        (Convert.ToDouble(a.numerator) /
                                        Convert.ToDouble(a.denominator)) <</pre>
                                        (Convert.ToDouble(b.numerator) /
           return new
TFrac(numerator * b.numerator,
                                        Convert.ToDouble(b.denominator));
denominator * b.denominator);
       }
                                               public static implicit
```

```
throw new
NotImplementedException();
                                                 public string
                                         getDenominatorString()
       public long
getNumeratorNum()
                                                     return
                                         denominator.ToString();
           return numerator;
                                                public override string
       public long
                                         ToString()
getDenominatorNum()
                                                     return
           return denominator;
                                         getNumeratorString() + "/" +
                                         getDenominatorString();
       public string
getNumeratorString()
           return
numerator.ToString();
```

### TFracEditor.cs:

```
ZeroFraction) || fraction == "0" ||
                                          fraction == "-0";
using System;
using System.Collections.Generic;
using System.Linq;
                                                  public override string
using System. Text;
                                          ToggleMinus()
using System. Threading. Tasks;
                                                      if (fraction[0] == '-')
namespace calculator
                                                          fraction =
                                          fraction.Remove(0, 1);
   public class TFracEditor :
                                                     else
AEditor
                                                         fraction = '-' +
                                          fraction;
       const string Separator =
                                                     return fraction;
       const string ZeroFraction =
"0/";
                                                  public override string
       const int
                                         AddNumber(int a)
max numerator length = 14;
        const int
                                                      if
max denominator length = 22;
                                          (!fraction.Contains(Separator) &&
       private string fraction;
                                          fraction.Length >
                                          max numerator length)
        public override string
                                                         return fraction;
Number
                                                      else if (fraction.Length
        {
                                          > max denominator length)
            get
                                                         return fraction;
            {
                                                      if (a < 0 | | a > 9)
               return fraction;
                                                         return fraction;
                                                      if (a == 0)
                                                          AddZero();
            set
                                                      else if (IsZero())
                                                         fraction =
               fraction = new
                                          fraction.First() == '-' ? "-" +
TFrac(value).ToString();
                                          a.ToString() : a.ToString();
           }
                                                     else
        }
                                                          fraction +=
                                          a.ToString();
        public TFracEditor()
                                                     return fraction;
           fraction = "0";
                                                  public override string
        public TFracEditor(long a,
                                          AddZero()
long b)
                                                  {
                                                      if (IsZero())
            fraction = new TFrac(a,
                                                          return fraction;
b).ToString();
                                                      if
                                          (fraction.Last().ToString() ==
                                          Separator)
        public TFracEditor(string
                                                          return fraction;
frac)
                                                      fraction += "0";
            fraction = new
                                                      return fraction;
TFrac(frac).ToString();
                                                  }
        }
                                                  public override string
       public override bool
                                          RemoveSymbol()
IsZero()
                                                 {
                                                     if (fraction.Length ==
            return
                                          1)
fraction.StartsWith(ZeroFraction) ||
```

fraction.StartsWith("-" +

```
fraction = "0";
                                                             AddNumber(8);
           else if (fraction.Length
                                                             break;
== 2 && fraction.First() == '-')
                                                         case Command.cNine:
               fraction = "-0";
                                                             AddNumber(9);
           else
                                                             break;
               fraction =
                                                         case Command.cSign:
fraction.Remove(fraction.Length -
                                                             ToggleMinus();
1);
                                                             break;
                                                         case
           return fraction;
                                         Command.cSeparator:
        }
                                                             AddSeparator();
                                                             break;
                                                         case Command.cBS:
        public override string
Clear()
                                                             RemoveSymbol();
                                                             break;
           fraction = "0";
                                                         case Command.cCE:
                                                             Clear();
           return fraction;
                                                             break;
                                                         default:
                                                             break;
       public override string
Edit(Command command)
                                                     return fraction;
            switch (command)
                                                 }
                case Command.cZero:
                                                /* public void
                   AddZero();
                                         SetEditor(TFrac frac)
                   break;
                                                {
               case Command.cOne:
                                                     fraction =
                   AddNumber(1);
                                         frac.ToString();
                   break;
                                                 } * /
               case Command.cTwo:
                   AddNumber(2);
                                                 public override string
                   break:
                                         AddSeparator()
               case Command.cThree:
                                                 {
                   AddNumber(3);
                                                     if
                                         (!fraction.Contains(Separator))
                   break;
                case Command.cFour:
                                                         fraction +=
                   AddNumber(4);
                                         Separator;
                   break;
                case Command.cFive:
                                                     return fraction;
                   AddNumber(5);
                   break;
                case Command.cSix:
                                                 public override string
                   AddNumber(6);
                                        ToString()
                   break;
                                                 {
                case Command.cSeven:
                                                     return Number;
                   AddNumber(7);
                                                 }
                   break;
                                            }
                case Command.cEight:
                                        }
```

# TComplexEditor.cs:

```
IsZero()
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 = " +
        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
            /*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
                                                           complex num = left +
ToggleMode()
                                           Separator + right;
                                                       return complex num;
            if (mode ==
PartToEdit.Real)
                                                   public override string
               mode =
PartToEdit.Imag;
                                           AddZero()
            else
               mode =
                                                       if (complex num == "0"
                                           || complex num == "-0" ||
PartToEdit.Real;
                                           complex num.EndsWith(" 0") ||
            return mode;
                                           complex num.EndsWith(" -0") ||
        }
                                           complex num.EndsWith(Separator))
        public override string
                                                          return complex num;
                                                       complex num += "0";
AddNumber(int a)
                                                       return complex num;
            if (a < 0 | | a > 9)
               return complex num;
            if (a == 0)
                                                   public string
                AddZero();
                                           AddNumberSeparator()
            string left = "", right
= "";
                                                       string left = "", right
                                           = "";
(complex_num.Contains(Separator))
                                           (complex num.Contains(Separator))
            {
                left =
                                                       {
complex num.Substring(0,
                                                           left =
complex num.IndexOf(Separator));
                                           complex num.Substring(0,
                                           complex_num.IndexOf(Separator));
               right =
complex num.Substring(complex num.In
                                                           right =
dexOf(Separator) +
                                           complex num.Substring(complex num.In
+Separator.Length);
                                           dexOf(Separator) +
                                           Separator.Length);
            else
                                                       else
                left = complex num;
                                                           left = complex num;
            if (mode ==
PartToEdit.Real)
                                                       if (mode ==
                                           PartToEdit.Real)
                if (left == "0" ||
left == "-0")
                                                           if
                                           (!left.Contains("."))
                    left =
left.First() == '-' ? "-" +
                                                               left += ".";
a.ToString() : a.ToString();
               else left +=
                                                       else
a.ToString();
                                                       {
                                                           if
                                           (!right.Contains(".") &&
            else
                                           right.Length > 0)
                if (right == "0" ||
                                                               right += ".";
right == "-0")
                    right =
right.First() == '-' ? "-" +
                                                       if (right == "")
a.ToString() : a.ToString();
                                                           complex num = left;
               else right +=
a.ToString();
                                                           complex num = left +
                                           Separator + right;
            }
            if (right == "")
                                                      return complex num;
               complex num = left;
            else
```

```
public override string
                                                              right =
AddSeparator()
                                          right.Remove(right.Length - 1);
                                                         }
            if
(!complex num.Contains(Separator))
                                                      if (right == "")
                complex num =
                                                          complex num = left;
complex num + Separator + "0";
                                                      else
               mode =
                                                          complex num = left +
PartToEdit.Imag;
                                          Separator + right;
            return complex num;
                                                      return complex num;
        public override string
                                                  public override string
RemoveSymbol()
                                          Clear()
            string left = "", right
                                                      complex num = "0";
                                                      mode = PartToEdit.Real;
           if
                                                      return complex num;
(complex num.Contains(Separator))
           {
               left =
                                                  public override string
complex num.Substring(0,
                                          ToString()
complex_num.IndexOf(Separator));
                                                  {
                                                      return complex num;
               right =
complex num.Substring(complex num.In
dexOf(Separator) +
Separator.Length);
                                                  public override string
                                          Edit(Command command)
           }
            else
                                                      switch (command)
               left = complex num;
                                                          case Command.cZero:
                                                              AddZero();
            if (mode ==
                                                              break;
PartToEdit.Real)
                                                          case Command.cOne:
                                                              AddNumber(1);
                if (left.Length == 1
                                                              break;
|| (left.Length == 2 && left[0] ==
                                                          case Command.cTwo:
'-'))
                                                              AddNumber(2);
                                                              break;
                                                          case Command.cThree:
                   left = left[0]
== '-' ? "-0" : "0";
                                                              AddNumber(3);
                                                              break;
                                                          case Command.cFour:
                else
                                                              AddNumber(4);
                    left =
                                                              break;
left.Remove(left.Length - 1);
                                                          case Command.cFive:
                                                              AddNumber(5);
               }
                                                              break;
            }
            else
                                                          case Command.cSix:
                                                              AddNumber(6);
                if (right.Length ==
                                                              break;
1 || (right.Length == 2 && right[0]
                                                          case Command.cSeven:
                                                              AddNumber(7);
                                                              break;
                   right = right[0]
                                                          case Command.cEight:
== '-' ? "-0" : "0";
                                                              AddNumber(8);
                                                              break:
                }
                else
                                                          case Command.cNine:
                                                              AddNumber(9);
                                                              break;
```

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

# Form1.cs: else if (new TFrac(v).getDenominatorNum() == 1) using System; using System.Collections.Generic; using System.ComponentModel; using System.Data; using System.Drawing; else if (new TFrac(v).getDenominatorNum() == 1) toReturn = new TFrac(v).getNumeratorString();

using System.DlawIng,
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

true)

case TAG\_PNUMBER:
 break;
case TAG\_FRAC:
 if (fracMode == toReturn = toR

toReturn =

ADT\_Control<TFrac,

TFracEditor> fracController;
 ADT Control<TComplex,

toReturn =

new TFrac(v).Numerator.ToString();
break;

case TAG\_COMPLEX:</pre>

bool pNumberMode = true;
string memmory\_buffer =
string.Empty;

return toReturn;
}

"PNUMBER\_"; memmory\_buffer =
tB\_Frac.Text;
public Form1()

fracController = new
ADT\_Control<TFrac, TFracEditor>();
 complexController = new
ADT\_Control<TFrac, TFracEditor>();
 complexController = new
ADT\_Control<TFracEditor>();
 control<TFracEditor>();
 control<TFracE

ADT\_Control<TComplex,

TComplexEditor>();

pNumberController = new

ADT\_Control<TPNumber,

TPNumberEditor>();

sender, Eventargs e)

{

sender, Eventargs e)

{

string.Empty)

{

string.Empty)

InitializeComponent();

MessageBox.Show("Буфер обмена пуст.\n" +

private string

вставить.",

NumberBeautifier(string Tag, string v)

{
 if (v == "ERROR")

HETABNIE.,

"Ошибка",

MessageBoxButtons.OK,

```
foreach (char i in
                                                            case 'C':
memmory buffer)
                                                                command =
                tB Frac.Text =
                                           AEditor.Command.cC;
fracController.ExecCommandEditor(Cha
                                                                break;
rToEditorCommand(i));
                                                            case 'D':
                                                                command =
        }
                                           AEditor.Command.cD;
        private static
                                                                break;
                                                            case 'E':
AEditor.Command
CharToEditorCommand(char ch)
                                                                command =
                                           AEditor.Command.cE;
            AEditor.Command command
                                                                break;
                                                            case 'F':
= AEditor.Command.cNone;
            switch (ch)
                                                                 command =
                                           AEditor.Command.cF;
                case '0':
                                                                break;
                                                            case '.':
                     command =
AEditor.Command.cZero;
                                                                command =
                    break;
                                           AEditor.Command.cSeparator;
                case '1':
                     command =
                                                            case '-':
AEditor.Command.cOne;
                                                                command =
                    break;
                                           AEditor.Command.cSign;
                case '2':
                                                                break;
                                                            case 'i':
                     command =
                                                                command =
AEditor.Command.cTwo;
                                           AEditor.Command.cToggleComplexMode;
                    break;
                case '3':
                                                                break;
                    command =
                                                        }
AEditor.Command.cThree;
                    break;
                                                        return command:
                case '4':
                                                    }
                                                    private static
                    command =
AEditor.Command.cFour;
                                           ADT Proc<T>.Operations
                                           CharToOperationsCommand<T>(char ch)
                    break;
                case '5':
                                           where T : ANumber, new()
                    command =
                                                        ADT Proc<T>.Operations
AEditor.Command.cFive;
                                            command =
                    break;
                case '6':
                                           ADT Proc<T>.Operations.None;
                    command =
                                                        switch (ch)
AEditor.Command.cSix;
                                                            case '+':
                    break;
                case '7':
                                                                command =
                    command =
                                           ADT Proc<T>.Operations.Add;
AEditor.Command.cSeven;
                                                                break;
                                                            case '-':
                    break;
                case '8':
                                                                command =
                     command =
                                           ADT Proc<T>.Operations.Sub;
AEditor.Command.cEight;
                                                                break;
                                                            case '*':
                    break;
                case '9':
                                                                command =
                     command =
                                           ADT Proc<T>.Operations.Mul;
AEditor.Command.cNine;
                                                                break;
                                                            case '/':
                    break;
                case 'A':
                                                                command =
                                           ADT Proc<T>.Operations.Div;
                     command =
AEditor.Command.cA;
                                                                break;
                    break;
                                                        }
                case 'B':
                     command =
                                                        return command;
AEditor.Command.cB;
                                                    }
                    break;
```

```
private static
                                                           string tag command =
AEditor.Command
                                           Tag.Replace (TAG COMPLEX,
KeyCodeToEditorCommand(Keys ch)
                                           string.Empty);
            AEditor.Command command
                                          Enum.TryParse(tag command, out
= AEditor.Command.cNone;
                                           AEditor.Command ParsedEnum);
            switch (ch)
                                                           tB Complex.Text =
                                           complexController. ExecCommandEditor(
                                           ParsedEnum);
                case Keys.Back:
                    command =
AEditor.Command.cBS;
                                                       else if
                                           (Tag.StartsWith(TAG PNUMBER))
                    break;
                case Keys.Delete:
                case Keys.Escape:
                                                           string tag command =
                    command =
                                           Tag.Replace (TAG PNUMBER,
                                           string.Empty);
AEditor.Command.cCE;
                    break;
                                           Enum. TryParse (tag command, out
            }
                                          AEditor.Command ParsedEnum);
            return command;
                                                         tB PNumber.Text =
                                           pNumberController. ExecCommandEditor(
                                           ParsedEnum);
        private void
AboutToolStripMenuItem1 Click(object
sender, EventArgs e)
                                                  private void
                                           Button Number Operation (object
MessageBox.Show("Универсальный
                                           sender, EventArgs e)
калькулятор\n" +
                "Авторы:
Бурдуковский Илья и Стояк Юрий\n" +
                                                       Button button =
                "Группа: ИП-813.",
                                           (Button) sender;
"Справка", MessageBoxButtons.OK,
                                                       string Tag =
MessageBoxIcon.Information);
                                          button.Tag.ToString();
                                                       if
        private void
                                           (Tag.StartsWith(TAG FRAC))
Button Number_Edit(object sender,
                                                           string tag command =
EventArgs e)
                                           Tag.Replace(TAG FRAC, string.Empty);
            Button button =
                                           Enum. TryParse (tag command, out
(Button) sender;
                                           ADT Proc<TFrac>.Operations
            string Tag =
button.Tag.ToString();
                                           ParsedEnum);
                                                          tB Frac.Text =
            if
                                           NumberBeautifier (TAG FRAC,
(Tag.StartsWith(TAG FRAC))
                                           fracController.ExecOperation(ParsedE
                                           num));
                string tag command =
Tag.Replace(TAG FRAC, string.Empty);
                                                       else if
                                           (Tag.StartsWith(TAG_COMPLEX))
Enum.TryParse(tag command, out
AEditor.Command ParsedEnum);
                                                           string tag command =
               tB Frac.Text =
                                           Tag.Replace(TAG COMPLEX,
fracController.ExecCommandEditor(Par
                                           string.Empty);
sedEnum);
                                          Enum.TryParse(tag command, out
                                          ADT Proc<TComplex>.Operations
            else if
                                          ParsedEnum);
(Tag.StartsWith(TAG COMPLEX))
                                                           tB Complex.Text =
                                          NumberBeautifier (TAG COMPLEX,
```

```
complexController.ExecOperation(Pars
                                                           string tag command =
                                           Tag.Replace (TAG PNUMBER,
edEnum));
                                           string.Empty);
            else if
(Tag.StartsWith(TAG PNUMBER))
                                           Enum.TryParse(tag command, out
                                           ADT Proc<TPNumber>.Functions
                                           ParsedEnum);
                string tag command =
Tag.Replace(TAG PNUMBER,
                                                           tB PNumber.Text =
                                           NumberBeautifier (TAG PNUMBER,
string.Empty);
                                           pNumberController.ExecFunction(Parse
Enum.TryParse(tag_command, out
                                           dEnum));
ADT Proc<TPNumber>.Operations
ParsedEnum);
                tB PNumber.Text =
NumberBeautifier (TAG PNUMBER,
                                                   private void
pNumberController.ExecOperation(Pars
                                           Button Memory (object sender,
edEnum));
                                           EventArgs e)
                                                       Button button =
                                           (Button) sender;
        private void
                                                       string Tag =
Button Number Function(object
                                           button.Tag.ToString();
sender, EventArgs e)
                                                       if
                                           (Tag.StartsWith(TAG FRAC))
            Button button =
(Button) sender;
                                                           string tag command =
            string Tag =
                                           Tag.Replace(TAG FRAC, string.Empty);
button.Tag.ToString();
                                           Enum.TryParse(tag command, out
                                           TMemory<TFrac>.Commands ParsedEnum);
(Tag.StartsWith(TAG FRAC))
                                                           dynamic exec =
                                           fracController.ExecCommandMemory(Par
                string tag command =
                                           sedEnum, tB Frac.Text);
                                                           if (ParsedEnum ==
Tag.Replace(TAG_FRAC, string.Empty);
                                           TMemory<TFrac>.Commands.Copy)
Enum.TryParse(tag_command, out
                                                                tB Frac.Text =
ADT Proc<TFrac>.Functions
                                           exec.Item1.ToString();
ParsedEnum);
                tB Frac.Text =
                                           label Frac Memory.Text = exec.Item2
                                           == true ? "M" : string.Empty;
NumberBeautifier (TAG FRAC,
fracController.ExecFunction(ParsedEn
um));
                                                       else if
                                           (Tag.StartsWith(TAG_COMPLEX))
            else if
(Tag.StartsWith(TAG_COMPLEX))
                                                           string tag command =
                                           Tag.Replace (TAG COMPLEX,
                string tag command =
                                           string.Empty);
Tag.Replace(TAG COMPLEX,
string.Empty);
                                           Enum.TryParse(tag command, out
                                           TMemory<TComplex>.Commands
Enum.TryParse(tag command, out
                                           ParsedEnum);
ADT Proc<TComplex>.Functions
                                                           dynamic exec =
ParsedEnum);
                                           complexController.ExecCommandMemory(
                tB Complex.Text =
                                           ParsedEnum, tB Complex.Text);
NumberBeautifier (TAG COMPLEX,
                                                           if (ParsedEnum ==
complexController.ExecFunction(Parse
                                           TMemory<TComplex>.Commands.Copy)
                                                                tB_Complex.Text
dEnum));
                                           = exec.Item1.ToString();
            else if
(Tag.StartsWith(TAG PNUMBER))
                                           label Complex Memory.Text =
                                           exec.Item2 == true ? "M" :
                                           string.Empty;
```

```
else if
                                                  private void
(Tag.StartsWith(TAG PNUMBER))
                                          Button Calculate (object sender,
                                          EventArgs e)
                string tag command =
Tag.Replace (TAG PNUMBER,
                                                      Button button =
string.Empty);
                                          (Button) sender;
                                                      string Tag =
Enum.TryParse(tag command, out
                                          button.Tag.ToString();
TMemory<TPNumber>.Commands
                                                      if
ParsedEnum);
                                          (Tag.StartsWith(TAG FRAC))
                dynamic exec =
pNumberController.ExecCommandMemory(
                                                          tB Frac.Text =
ParsedEnum, tB_Complex.Text);
                                          NumberBeautifier (TAG FRAC,
                if (ParsedEnum ==
                                          fracController.Calculate());
TMemory<TPNumber>.Commands.Copy)
                    tB PNumber.Text
                                                      else if
                                           (Tag.StartsWith(TAG COMPLEX))
= exec.Item1.ToString();
                                                         tB Complex.Text =
label PNumber Memory.Text =
exec.Item2 == true ? "M" :
                                          NumberBeautifier (TAG COMPLEX,
string.Empty;
                                          complexController.Calculate());
                                                      else if
                                          (Tag.StartsWith(TAG PNUMBER))
       private void
Button Reset (object sender,
                                                          tB PNumber.Text =
                                          pNumberController.Calculate();
EventArgs e)
       {
           Button button =
(Button) sender;
            string Tag =
                                                  private void
                                          TrackBar PNumber ValueChanged(object
button.Tag.ToString();
                                          sender, EventArgs e)
           i f
(Tag.StartsWith(TAG FRAC))
                                                      label PNumber P.Text =
                                          trackBar PNumber.Value.ToString();
                tB Frac.Text =
fracController.Reset();
                                          pNumberController.Edit.Notation =
label Frac Memory.Text =
                                          trackBar PNumber.Value;
                                                     tB PNumber.Text =
string.Empty;
                                          pNumberController.Reset();
            else if
(Tag.StartsWith(TAG COMPLEX))
                                          label PNumber Memory.Text =
                                          string.Empty;
                tB Complex.Text =
                                                      string AllowedEndings =
complexController.Reset();
                                          "0123456789ABCDEF";
                                                     foreach (Control i in
label Complex Memory.Text =
                                          tabPage PNumber.Controls.OfType<Butt
string.Empty;
                                          on>())
           else if
                                                          if
(Tag.StartsWith(TAG PNUMBER))
                                          (AllowedEndings.Contains(i.Name.ToSt
                                          ring().Last()) &&
               tB PNumber.Text =
                                          i.Name.ToString().Substring(i.Name.T
                                          oString().Length - 2, 1) == "")
pNumberController.Reset();
                                                               int j =
label PNumber Memory.Text =
                                          AllowedEndings.IndexOf(i.Name.ToStri
string.Empty;
                                          ng().Last());
                                                               if (j <
        }
                                          trackBar PNumber.Value)
```

```
i.Enabled =
                                           complexController.ExecCommandEditor(
true;
                                           command);
                    if ((j >=
                                                                break;
trackBar PNumber.Value) && (j <=
15))
                                                            case 2: {
                                                                if (e.KeyCode ==
                         i.Enabled =
                                           Keys.Enter)
false;
                     }
                                           b Complex Eval.PerformClick();
                }
                                                                else
pNumberController.Proc.Left Result o
                                           AEditor.Command command =
perand.Notation =
                                           KeyCodeToEditorCommand(e.KeyCode);
trackBar_PNumber.Value;
                                                                    if (command
                                           != AEditor.Command.cNone)
pNumberController.Proc.Right operand
.Notation = trackBar PNumber.Value;
                                           tB Complex.Text =
                                           pNumberController.ExecCommandEditor(
                                           command);
        private void
Form1 KeyDown (object sender,
                                                                break;
KeyEventArgs e)
                                                            default:
        {
                                                                break;
                                                        }
            switch
(tabControl.SelectedIndex)
                                                   private void
                case 0: {
                                           Form1 KeyPress(object sender,
                    if (e.KeyCode ==
                                           KeyPressEventArgs e)
Keys.Enter)
                                                        switch
b Frac Eval.PerformClick();
                                           (tabControl.SelectedIndex)
                    else
                                                            case 0: {
                                                                if (e.KeyChar >=
                                           '0' && e.KeyChar <= '9' || e.KeyChar
AEditor.Command command =
                                           == '.')
KeyCodeToEditorCommand(e.KeyCode);
                        if (command
                                                                    tB Frac.Text
!= AEditor.Command.cNone)
                                           fracController.ExecCommandEditor(Cha
tB Frac.Text =
                                           rToEditorCommand(e.KeyChar));
fracController.ExecCommandEditor(com
                                                                else if
mand);
                                           (operation signs.Contains(e.KeyChar)
                    break;
                                                                    tB Frac.Text
                                           = NumberBeautifier (TAG FRAC,
                case 1: {
                                           fracController.ExecOperation(CharToO
                                           perationsCommand<TFrac>(e.KeyChar)))
                    if (e.KeyCode ==
Keys.Enter)
                                                                break;
b_Complex_Eval.PerformClick();
                    else
                                                            case 1: {
                                                                if ((e.KeyChar
                                           >= '0' && e.KeyChar <= '9') ||
AEditor.Command command =
                                           e.KeyChar == '.')
KeyCodeToEditorCommand(e.KeyCode);
                        if (command
                                           tB Complex.Text =
!= AEditor.Command.cNone)
                                           complexController.ExecCommandEditor(
                                           CharToEditorCommand(e.KeyChar));
tB Complex.Text =
```

```
else if
                                                        for (int i = 0; i <
(operation signs.Contains(e.KeyChar)
                                           complexController.history.Count();
                                           i++)
tB Complex.Text =
                                                            List<string>
NumberBeautifier (TAG COMPLEX,
                                           currentRecord =
complexController.ExecOperation(Char
                                           complexController.history[i].ToList(
ToOperationsCommand<TComplex>(e.KeyC
                                           );
har)));
                                           history.dataGridView1.Rows.Add(curre
                    break;
                }
                                           ntRecord[0], currentRecord[1]);
                case 2: {
                                                       for (int i = 0; i <
                    if ((e.KeyChar
>= '0' && e.KeyChar <= '9') ||
                                           pNumberController.history.Count();
(e.KeyChar >= 'A' && e.KeyChar <=</pre>
                                           i++)
'F') || (e.KeyChar == '.'))
                                                            List<string>
tB PNumber.Text =
                                           currentRecord =
pNumberController.ExecCommandEditor(
                                           pNumberController.history[i].ToList(
CharToEditorCommand(e.KeyChar));
                                           );
                    else if
(operation signs.Contains(e.KeyChar)
                                           history.dataGridView1.Rows.Add(curre
                                           ntRecord[0], currentRecord[1]);
tB PNumber.Text =
NumberBeautifier (TAG PNUMBER,
pNumberController.ExecOperation(Char
                                                   private void
ToOperationsCommand<TPNumber>(e.KeyC
                                           дробьFracTSMI Click(object sender,
har)));
                                           EventArgs e)
                    break;
                                                   {
                }
                                                       дробьFracTSMI.Checked =
                default:
                                           true:
                    break;
                                                       числоFracTSMI.Checked =
                                           false;
                                                        fracMode = true;
        private void
HistoryToolStripMenuItem1 Click(obje
ct sender, EventArgs e)
                                                   private void
                                           числоFracTSMI_Click(object sender,
            Form2 history = new
                                           EventArgs e)
Form2();
            history.Show();
                                                       дробьFracTSMI.Checked =
                                           false;
(fracController.history.Count() ==
                                                       числоFracTSMI.Checked =
0)
                                           true;
            {
                                                       fracMode = false;
MessageBox.Show("История пуста",
"Внимание", MessageBoxButtons.OK,
                                                   private void
MessageBoxIcon.Warning);
                                           комплексноеComplexTSMI Click(object
                                           sender, EventArgs e)
                return;
            for (int i = 0; i <
fracController.history.Count(); i++)
                                           комплексноеComplexTSMI.Checked =
                                           true;
                List<string>
currentRecord =
                                           действительноеComplexTSMI.Checked =
fracController.history[i].ToList();
                                           false;
                                                       complexMode = true;
history.dataGridView1.Rows.Add(curre
ntRecord[0], currentRecord[1]);
```

# TMemory.cs:

```
}
using System;
                                                   public TMemory(T num)
using System.Collections.Generic;
using System.Linq;
                                                       number = num;
using System. Text;
                                                       state = false;
using System. Threading. Tasks;
namespace calculator
                                                   public T Add(T num)
    public class TMemory<T> where T
                                                       state = true;
: ANumber, new()
                                                       dynamic a = number;
                                                       dynamic b = num;
        public enum Commands {
                                                       number = a.Add(b);
            Store, Add, Clear, Copy
                                                       return number;
        T number;
                                                   public void Clear()
        bool state;
        public T FNumber
                                                       number = new T();
        {
                                                       state = false;
            get
            {
                state = true;
                                                   public (T, bool)
               return number;
                                           Edit(Commands command, T newNumber)
            set
                                                       switch (command) {
            {
                                                            case Commands.Store:
                number = value;
                                                               state = true;
                state = true;
                                                               number =
            }
                                           newNumber;
                                                               break;
        public bool FState
                                                            case Commands.Add:
                                                                dynamic a =
            get
                                           number;
            {
                                                                dynamic b =
               return state;
                                           newNumber;
                                                                number =
                                           a.Add(b);
            set
                                                               break;
                                                            case Commands.Clear:
                state = value;
                                                                Clear();
                                                                break;
        }
                                                       return (number, state);
        public TMemory()
                                                   }
                                               }
            number = new T();
```

state = false;

## THistory.cs:

```
o, string record string)
using System;
using System.Collections.Generic;
                                                       Record record = new
using System.Linq;
                                          Record(o, record string);
using System.Text;
                                                      L.Add(record);
using System. Threading. Tasks;
namespace calculator
                                                   public Record this[int i]
   public class THistory
                                                       get
        public struct Record
                                                           if (i < 0 || i >=
                                          L.Count)
            private string oper;
                                                               throw new
            private string str res;
                                          IndexOutOfRangeException();
            public Record(string
                                                          return L[i];
operation, string str res)
                                                       }
                                                       set
                str res =
                                                       {
str res.Remove(0, 1);
                                                           if (i < 0 || i >=
                this.oper =
                                          L.Count)
operation;
                                                               throw new
                this.str res =
                                           IndexOutOfRangeException();
str res;
                                                          L[i] = value;
            public List<string>
                                                   }
ToList()
                                                   public void Clear()
               return new
List<string> { oper, str res };
                                                       L.Clear();
           }
        }
                                                   public int Count()
        List<Record> L;
        public THistory()
                                                       return L.Count();
        {
            L = new List<Record>();
                                               }
```

public void AddRecord(string

#### if $(p < 2 \mid | p > 16)$ TPNumber.cs: throw new IndexOutOfRangeException(); using System; if (n == 0)using System.Collections.Generic; return "0"; using System.Linq; if (p == 10)using System. Text; return using System. Threading. Tasks; n.ToString(); namespace calculator bool isNegative = false; public class TPNumber : ANumber if (n < 0)public class isNegative = ADT Convert 10 p true; n \*= -1;public static string Do (double n, int p, int c) string buf = ""; if (p < 2 | | p > 16)while (n > 0)throw new IndexOutOfRangeException(); buf += if (c < 0 | | c > 10)Int to char((int)n % p); throw new n /= p;IndexOutOfRangeException(); long leftSide = if (isNegative) (long)n; buf += "-"; double rightSide = n char[] chs = - leftSide; buf.ToCharArray(); if (rightSide < 0)</pre> Array.Reverse(chs); rightSide \*=-1;return new string(chs); string leftSideString = Int\_to\_p(leftSide, p); public static string string Flt to p(double n, int p, int c) rightSideString = Flt to p(rightSide, p, c); if $(p < 2 \mid | p > 16)$ throw new return IndexOutOfRangeException(); leftSideString + (rightSideString == if (c < 0 | | c > 10)String.Empty ? "" : ".") + throw new rightSideString; IndexOutOfRangeException(); string pNumber = public static char String.Empty; Int to char(int d) for (int i = 0; i <c; i++) if $(d > 15 \mid | d < 0)$ pNumber += throw new Int\_to\_char((int)(n \* p)); IndexOutOfRangeException(); n = n \* p -}

(int)(n \* p);

pNumber.TrimEnd('0');

}

}

pNumber =

return pNumber;

string allSymbols =

return

public static string

"0123456789ABCDEF";

allSymbols.ElementAt(d);

Int\_to\_p(long n, int p)

}

```
public class
                                                                 for (int i = 0;
ADT Convert p 10
                                            i < chs.Length; i++)</pre>
            public static double
                                                                     if
Dval(string p num, int p)
                                            (Char to num(chs[i]) > p)
                                                                         throw
                if (p < 2 | | p > 16)
                                            new Exception();
                     throw new
                                                                     buf +=
IndexOutOfRangeException();
                                            Char to num(chs[i]) * Math.Pow(p,
                                            i);
                bool minus = false;
                if
                                                            }
(p_num.Contains("-"))
                                                            if (minus)
                                                                buf = buf * -1;
                    minus = true;
                    p num =
p_num.Replace("-",
                                                            return buf;
                }
                double buf = 0d;
                                                        public static double
(p num.Contains("."))
                                            Char to num(char ch)
                {
                     string[] lr =
                                                            string allNums =
p num.Split('.');
                                            "0123456789ABCDEF";
                     if (lr[0].Length
                                                            if
== 0)
                                            (!allNums.Contains(ch))
                        throw new
                                                                throw new
                                            IndexOutOfRangeException();
Exception();
                     char[] chs =
                                                            return
lr[0].ToCharArray();
                                            allNums.IndexOf(ch);
Array.Reverse(chs);
                                                        }
                     for (int i = 0;
i < chs.Length; i++)</pre>
                                                        public static double
                                            Convert(string p_num, int p, double
                         if
                                            weight)
(Char to num(chs[i]) > p)
                                                            return 0d;
                             throw
new Exception();
                         buf +=
Char to num(chs[i]) * Math.Pow(p,
                                                    public double Number;
                                                    public int Notation;
                                                    public int Precision;
                     char[] chsr =
lr[1].ToCharArray();
                     for (int i = 0;
                                                    public TPNumber()
i < chsr.Length; i++)</pre>
                                                        Number = 0f;
                                                        Notation = 10;
(Char to num(chsr[i]) > p)
                                                        Precision = 5;
                             throw
                                                    public TPNumber (double num,
new Exception();
                        buf +=
                                            int not, int pre)
Char_to_num(chsr[i]) * Math.Pow(p, -
(i + 1);
                                                        if (not < 2 || not > 16
                                            || pre < 0 || pre > 10)
                }
                                                        {
                else
                                                            Number = 0f;
                                                            Notation = 10;
                     char[] chs =
                                                            Precision = 5;
p num.ToCharArray();
                                                        }
                                                        else
Array.Reverse(chs);
                                                        {
```

```
Number = num;
                                                  public TPNumber Div(TPNumber
                Notation = not;
                Precision = pre;
                                          a)
            }
                                                      TPNumber tmp = a.Copy();
        }
        public TPNumber(TPNumber
                                                      if (a.Notation !=
                                          Notation || a.Precision !=
anotherTPNumber)
                                          Precision)
       {
           Number =
                                                          tmp.Number = 0.0;
anotherTPNumber.Number;
           Notation =
                                                          return tmp;
anotherTPNumber.Notation;
           Precision =
                                                      tmp.Number = Number /
anotherTPNumber.Precision;
                                          a.Number;
                                                      return tmp;
       public TPNumber(string str,
int not, int pre)
                                                  public TPNumber Sub(TPNumber
            try
                                          a)
            {
               Number =
                                                      TPNumber tmp = a.Copy();
ADT_Convert_p_10.Dval(str, not);
                                                      if (a.Notation !=
               Notation = not;
                                          Notation || a.Precision !=
                Precision = pre;
                                          Precision)
            }
                                                          tmp.Number = 0.0;
            catch
                                                          return tmp;
                throw new
System.OverflowException();
                                                      tmp.Number = Number -
                                          a.Number:
        }
                                                      return tmp;
        public TPNumber Add (TPNumber
                                                  public object Square()
a)
                                                      return new
            TPNumber tmp = a.Copy();
                                         TPNumber (Number * Number, Notation,
            if (a.Notation !=
                                          Precision);
this.Notation || a.Precision !=
Precision)
                                                  public object Reverse()
                tmp.Number = 0.0;
                                                      return new TPNumber(1 /
                return tmp;
                                          Number, Notation, Precision);
            tmp.Number = Number +
                                                  public bool IsZero()
a.Number;
            return tmp;
                                                      return Number == 0;
        }
        public TPNumber Mul(TPNumber
                                                  public TPNumber Copy()
a)
                                                      return
            TPNumber tmp = a.Copy(); (TPNumber)this.MemberwiseClone();
            if (a.Notation !=
this.Notation || a.Precision !=
this.Precision)
                                                  public override string
                                          ToString()
                tmp.Number = 0.0;
               return tmp;
                                                      string str;
                                                      try
            tmp.Number = Number *
a.Number;
                                                          str =
            return tmp;
                                          ADT_Convert_10_p.Do(Number,
        }
                                          Notation, Precision);
```

```
if (!checkOnSymbol(a))
            }
            catch
                                                          return false;
               throw new
                                                      return true;
System.OverflowException();
                                                   }
           }
                                                  private bool
            return str;
                                          checkOnBase(string a, int b)
        public override void
                                                       foreach (char iter in a)
SetString(string str)
                                                          int move =
                                          Math.Abs('A' - iter);
            try
                                                          int iter int = iter
                Number =
                                          - '0';
ADT Convert_p_10.Dval(str,
                                                          if (iter >= 'A' &&
Notation);
                                          iter <= 'Z')
           catch
                                                               iter int = 10 +
                                          move;
                throw new
System.OverflowException();
                                                          if (iter == ',')
                                                              continue;
       private bool check(double a,
                                                          if (iter int >= b)
int b, int c)
                                                              return false;
            string a str =
a.ToString();
            if (!checkOnBase(a str,
                                                      return true;
b))
               return false;
                                                  private bool checkOnC(string
                                          a, int c)
            if (!checkOnC(a str, c))
                                                  {
                                                       if (checkPoint(a) && c >
               return false;
                                          0)
                                                          string[] spliter =
            if
(!checkOnSymbol(a str))
                                          a.Split(',');
                                                          if
                return false;
                                          (spliter[1].Length == c)
            return true;
                                                               return true;
       private bool check(string a,
string b, string c)
                                                      return false;
           int b int =
                                                  private bool
Convert. ToInt32 (b);
                                          checkPoint(string n)
           int c int =
                                                  {
Convert. ToInt32(c);
                                                       int i;
           if (!checkOnBase(a,
                                                      for (i = 0; i < n.Length)
                                          && n[i] != ','; i++) { }
b int))
                                                      if (i < n.Length)</pre>
               return false;
                                                          return true;
            if (!checkOnC(a, c int))
                                                      return false;
               return false;
                                                   }
```

### TPNumberEditor.cs:

```
TPNumber.ADT Convert 10 p.Do(num,
                                           not, pre);
using System;
using System.Collections.Generic;
using System.Linq;
using System. Text;
                                                   public override bool
using
                                           IsZero()
System.Text.RegularExpressions;
using System. Threading. Tasks;
                                                       return
                                           ZeroPNumber.IsMatch(pNumber);
namespace calculator
                                                   public override string
    public class TPNumberEditor :
                                           ToggleMinus()
AEditor
   {
                                                       if (pNumber.ElementAt(0)
        private string pNumber;
                                           == '-')
                                                           pNumber =
        public int Notation;
                                           pNumber.Remove(0, 1);
        public int Precision;
                                                       else
                                                           pNumber = "-" +
        Regex ZeroPNumber = new
                                          pNumber;
Regex ("^-? (0+|.?0+|0+.(0+)?)$");
                                                       return pNumber;
        const string Separator =
                                                   public override string
        public override string
                                           AddNumber(int num)
Number
                                                  {
                                                       if (num < 0 || num >=
            get
                                           Notation)
            {
                                                          return pNumber;
                return pNumber;
                                                       if (num == 0)
            }
                                                           AddZero();
                                                       else if (pNumber == "0"
            set
                                           | | pNumber == "-0")
                                                           pNumber =
                pNumber = new
                                           pNumber.First() == '-' ? "-" +
TPNumber (value, Notation,
                                           TPNumber.ADT Convert 10 p.Int to cha
Precision).ToString();
                                           r(num).ToString() :
            }
                                           TPNumber.ADT Convert_10_p.Int_to_cha
                                           r(num).ToString();
        public TPNumberEditor()
                                                      else
                                                          pNumber +=
            pNumber = "0";
                                           TPNumber.ADT Convert 10 p.Int to cha
            Notation = 10;
                                           r(num).ToString();
            Precision = 5;
                                                       return pNumber;
        public TPNumberEditor(double
                                                   public override string
num, int not, int pre)
                                           RemoveSymbol()
                                                   {
            if (not < 2 || not > 16
                                                       if (pNumber.Length == 1)
|| pre < 0 || pre > 10)
                                                           pNumber = "0";
                                                       else if (pNumber.Length
                pNumber = "0";
                                          == 2 && pNumber.First() == '-')
                Notation = 10;
                                                           pNumber = "-0";
                Precision = 5;
                                                       else
            }
                                                           pNumber =
            else
                                           pNumber.Remove(pNumber.Length - 1);
            {
                                                      return pNumber;
                Notation = not;
                Precision = pre;
```

pNumber =

```
public override string
                                                                break;
AddSeparator()
                                                            case Command.cFive:
                                                                AddNumber(5);
            if
                                                                break;
(!pNumber.Contains(Separator))
                                                            case Command.cSix:
                pNumber +=
                                                                AddNumber(6);
                                                                break;
Separator;
                                                            case Command.cSeven:
            return pNumber;
                                                                AddNumber(7);
                                                                break;
                                                            case Command.cEight:
        public override string
                                                                AddNumber(8);
AddZero()
                                                                break;
                                                            case Command.cNine:
            if
                                                                AddNumber(9);
(pNumber.Contains(Separator) &&
pNumber.Last().ToString() ==
                                                                break;
                                                            case Command.cA:
Separator)
                                                                AddNumber(10);
                return pNumber;
            if (pNumber == "0" ||
                                                                break;
pNumber == "0.")
                                                            case Command.cB:
                return pNumber;
                                                                AddNumber(11);
            pNumber += "0";
                                                                break;
            return pNumber;
                                                            case Command.cC:
                                                                AddNumber(12);
        public override string
                                                                break;
ToString()
                                                            case Command.cD:
                                                                AddNumber (13);
        {
            return pNumber;
                                                                break;
                                                            case Command.cE:
        public override string
                                                                AddNumber(14);
Clear()
                                                                break;
                                                            case Command.cF:
            pNumber = "0";
                                                                AddNumber(15);
            return pNumber;
                                                                break;
                                                            case Command.cSign:
                                                                ToggleMinus();
        public override string
                                                                break;
Edit(Command command)
                                                            case
                                           Command.cSeparator:
            switch (command)
                                                                AddSeparator();
                                                                break;
                case Command.cZero:
                                                            case Command.cBS:
                    AddZero();
                                                                RemoveSymbol();
                    break;
                                                                break;
                case Command.cOne:
                                                            case Command.cCE:
                    AddNumber(1);
                                                                Clear();
                    break;
                                                                break;
                case Command.cTwo:
                                                            default:
                    AddNumber(2);
                                                                break;
                    break;
                case Command.cThree:
                                                        return Number;
                    AddNumber(3);
                                                    }
                    break;
                                               }
                case Command.cFour:
                    AddNumber(4);
```

# Листинг тестов

#### public void UnitTest1.cs: InitString4() using string fracString = Microsoft. Visual Studio. Test Tools "00000003/000004"; .UnitTesting; TFrac fracClass = using calculator; new TFrac(fracString); using System; string Expect = "3/4"; namespace calculator tests Assert.AreEqual(Expect, [TestClass] fracClass.ToString()); public class TFracTest } [TestMethod] [TestMethod] public void public void InitString1() InitString5() string fracString = string fracString = "1/2"; "-00000003/000004"; TFrac fracClass = TFrac fracClass = new TFrac(fracString); new TFrac(fracString); string Expect = "-Assert.AreEqual(fracString, 3/4"; fracClass.ToString()); } Assert.AreEqual (Expect, fracClass.ToString()); [TestMethod] } public void InitString2() [TestMethod] { public void string fracString = InitNumber1() "111/2"; TFrac fracClass = TFrac fracClass = new TFrac(fracString); new TFrac(1, 2); string Expect = Assert.AreEqual(fracString, "1/2"; fracClass.ToString()); Assert.AreEqual (Expect, fracClass.ToString()); [TestMethod] } public void InitString3() [TestMethod] { public void string fracString = InitNumber2() "-100/60"; TFrac fracClass = TFrac fracClass = new TFrac(fracString); new TFrac(100, 100); string Expect = "string Expect = 5/3"; "1/1"; Assert.AreEqual (Expect, Assert.AreEqual(Expect, fracClass.ToString()); fracClass.ToString()); } } [TestMethod] [TestMethod]

```
TFrac fracClass1 =
        public void
InitNumber3()
                                      new TFrac(-1, 2);
                                                   TFrac fracClass2 =
        {
            TFrac fracClass =
                                      new TFrac(-1, 2);
new TFrac(-100, -99);
                                                   fracClass2 =
            string Expect =
                                      fracClass1.Add(fracClass2);
"100/99";
                                                  string answer = "-
                                       1/1";
Assert.AreEqual(Expect,
fracClass.ToString());
                                      Assert.AreEqual(answer,
                                       fracClass2.ToString());
        }
                                               }
        [TestMethod]
        public void
                                               [TestMethod]
InitNumber4()
                                               public void Add3()
            TFrac fracClass =
                                                   TFrac fracClass1 =
new TFrac(0, 0);
                                      new TFrac(-6, 2);
                                                   TFrac fracClass2 =
            string Expect =
"0/1";
                                       new TFrac(6, 2);
                                                   fracClass2 =
                                      fracClass1.Add(fracClass2);
Assert.AreEqual(Expect,
fracClass.ToString());
                                                  string answer =
                                       "0/1";
        [TestMethod]
                                      Assert.AreEqual(answer,
        public void
                                       fracClass2.ToString());
InitNumber5()
                                               }
       {
            TFrac fracClass =
                                               [TestMethod]
new TFrac(50, -5);
                                               public void Add4()
            string fracCompar =
"-10/1";
                                                   TFrac fracClass1 =
                                      new TFrac(50, 3);
Assert.AreEqual(fracCompar,
                                                   TFrac fracClass2 =
fracClass.ToString());
                                      new TFrac(0, 1);
                                                   fracClass2 =
        }
                                      fracClass1.Add(fracClass2);
        [TestMethod]
                                                  string answer =
                                      "50/3";
        public void Add1()
            TFrac fracClass1 =
                                      Assert.AreEqual(answer,
                                      fracClass2.ToString());
new TFrac(1, 4);
            TFrac fracClass2 =
                                               }
new TFrac(-3, 4);
            fracClass2 =
                                               [TestMethod]
fracClass1.Add(fracClass2);
                                               public void Add5()
           string answer = "-
1/2";
                                                   TFrac fracClass1 =
                                      new TFrac(0, 1);
                                                   TFrac fracClass2 =
Assert.AreEqual(answer,
fracClass2.ToString());
                                      new TFrac(0, 1);
                                                  fracClass2 =
        }
                                      fracClass1.Add(fracClass2);
        [TestMethod]
                                                  string answer =
        public void Add2()
                                      "0/1";
        {
```

```
TFrac fracClass2 =
Assert.AreEqual(answer,
                                       new TFrac(12, 1);
fracClass2.ToString());
                                                   fracClass2 =
                                       fracClass1.Mul(fracClass2);
                                                   string answer = "-
        [TestMethod]
                                       2/1";
        public void Multiply1()
                                       Assert.AreEqual (answer,
            TFrac fracClass1 =
                                       fracClass2.ToString());
new TFrac(-1, 2);
                                               }
            TFrac fracClass2 =
new TFrac(-1, 2);
                                                [TestMethod]
            fracClass2 =
                                               public void Multiply5()
fracClass1.Mul(fracClass2);
            string answer =
                                                   TFrac fracClass1 =
"1/4";
                                       new TFrac(-1, 6);
                                                   TFrac fracClass2 =
Assert.AreEqual(answer,
                                       new TFrac(12, 1);
fracClass2.ToString());
                                                   fracClass2 =
        }
                                       fracClass1.Mul(fracClass2);
                                                   string answer = "-
                                       2/1";
        [TestMethod]
        public void Multiply2()
                                       Assert.AreEqual (answer,
            TFrac fracClass1 =
                                       fracClass2.ToString());
new TFrac(1, 6);
            TFrac fracClass2 =
new TFrac(0, 1);
                                                [TestMethod]
            fracClass2 =
                                               public void Substract1()
fracClass1.Mul(fracClass2);
                                                   TFrac fracClass1 =
            string answer =
"0/1";
                                       new TFrac(0, 1);
                                                   TFrac fracClass2 =
Assert.AreEqual(answer,
                                       new TFrac(1, 1);
                                                   fracClass2 =
fracClass2.ToString());
                                       fracClass1.Sub(fracClass2);
        }
                                                   string answer = "-
                                       1/1";
        [TestMethod]
        public void Multiply3()
                                       Assert.AreEqual (answer,
            TFrac fracClass1 =
                                       fracClass2.ToString());
new TFrac(1, 6);
                                               }
            TFrac fracClass2 =
new TFrac(1, 6);
                                                [TestMethod]
            fracClass2 =
                                               public void Substract2()
fracClass1.Mul(fracClass2);
           string answer =
                                                   TFrac fracClass1 =
"1/36";
                                       new TFrac(5, 1);
                                                   TFrac fracClass2 =
                                       new TFrac(1, 1);
Assert.AreEqual(answer,
fracClass2.ToString());
                                                   fracClass2 =
                                       fracClass1.Sub(fracClass2);
        }
                                                   string answer =
                                       "4/1";
        [TestMethod]
        public void Multiply4()
                                       Assert.AreEqual(answer,
            TFrac fracClass1 =
                                       fracClass2.ToString());
new TFrac(-1, 6);
```

```
string answer =
        [TestMethod]
                                       "5/6";
        public void Substract3()
                                       Assert.AreEqual(answer,
            TFrac fracClass1 =
                                       fracClass2.ToString());
new TFrac(1, 2);
            TFrac fracClass2 =
new TFrac(1, 2);
                                                [TestMethod]
            fracClass2 =
                                               public void Divide2()
fracClass1.Sub(fracClass2);
            string answer =
                                                   TFrac fracClass1 =
"0/1";
                                       new TFrac(1, 1);
                                                   TFrac fracClass2 =
Assert.AreEqual(answer,
                                       new TFrac(5, 6);
fracClass2.ToString());
                                                   fracClass2 =
                                       fracClass1.Div(fracClass2);
        }
                                                   string answer =
                                       "6/5";
        [TestMethod]
        public void Substract4()
                                       Assert.AreEqual(answer,
            TFrac fracClass1 =
                                       fracClass2.ToString());
new TFrac(-1, 6);
            TFrac fracClass2 =
new TFrac(-1, 6);
                                               [TestMethod]
                                               public void Divide3()
            fracClass2 =
fracClass1.Sub(fracClass2);
                                                   TFrac fracClass1 =
            string answer =
"0/1";
                                       new TFrac(0, 1);
                                                   TFrac fracClass2 =
Assert.AreEqual(answer,
                                       new TFrac(5, 6);
                                                   fracClass2 =
fracClass2.ToString());
        }
                                       fracClass1.Div(fracClass2);
                                                   string answer =
                                       "0/1";
        [TestMethod]
        public void Substract5()
                                       Assert.AreEqual(answer,
            TFrac fracClass1 =
                                       fracClass2.ToString());
new TFrac(-1, 6);
            TFrac fracClass2 =
new TFrac(2, 6);
                                               [TestMethod]
            fracClass2 =
                                               public void Divide4()
fracClass1.Sub(fracClass2);
            string answer = "-
                                                   TFrac fracClass1 =
1/2";
                                       new TFrac(2, 3);
                                                   TFrac fracClass2 =
Assert.AreEqual(answer,
                                       new TFrac(7, 4);
fracClass2.ToString());
                                                    fracClass2 =
                                       fracClass1.Div(fracClass2);
        }
                                                   string answer =
                                       "8/21";
        [TestMethod]
        public void Divide1()
                                       Assert.AreEqual(answer,
            TFrac fracClass1 =
                                       fracClass2.ToString());
new TFrac(5, 6);
            TFrac fracClass2 =
                                               [TestMethod]
new TFrac(1, 1);
            fracClass2 =
                                               public void Divide5()
fracClass1.Div(fracClass2);
```

```
TFrac fracClass1 =
                                                   TFrac fracClass =
new TFrac(2, 3);
                                       new TFrac(2, 3);
            TFrac fracClass2 =
                                                   fracClass =
new TFrac(2, 3);
                                       fracClass.Square() as TFrac;
            fracClass2 =
                                                   string answer =
fracClass1.Div(fracClass2);
                                       "4/9";
            string answer =
"1/1";
                                       Assert.AreEqual (answer,
                                       fracClass.ToString());
Assert.AreEqual(answer,
                                               }
fracClass2.ToString());
        }
                                                [TestMethod]
                                               public void Square2()
        [TestMethod]
        public void Reverse1()
                                                   TFrac fracClass =
                                       new TFrac(0, 1);
            TFrac fracClass =
                                                   fracClass =
new TFrac(-2, 3);
                                       fracClass.Square() as TFrac;
            fracClass =
                                                   string answer =
fracClass.Reverse() as TFrac;
                                       "0/1";
            string answer = "-
3/2";
                                       Assert.AreEqual (answer,
                                       fracClass.ToString());
Assert.AreEqual (answer,
fracClass.ToString());
                                                [TestMethod]
                                               public void Square3()
        [TestMethod]
                                                   TFrac fracClass =
        public void Reverse2()
                                       new TFrac(-2, 3);
            TFrac fracClass =
                                                   fracClass =
new TFrac(0, 1);
                                       fracClass.Square() as TFrac;
            fracClass =
                                                   string answer =
                                       "4/9";
fracClass.Reverse() as TFrac;
            string answer =
"0/1";
                                       Assert.AreEqual(answer,
                                       fracClass.ToString());
Assert.AreEqual(answer,
                                               }
fracClass.ToString());
                                               [TestMethod]
                                               public void Equal1()
        [TestMethod]
        public void Reverse3()
                                                   TFrac fracClass1 =
                                       new TFrac(1, 3);
            TFrac fracClass =
                                                   TFrac fracClass2 =
new TFrac(5, 6);
                                       new TFrac(1, 3);
            fracClass =
fracClass.Reverse() as TFrac;
                                       Assert.IsTrue(fracClass1.Equal(f
            string answer =
                                       racClass2));
"6/5";
                                               }
Assert.AreEqual(answer,
                                                [TestMethod]
fracClass.ToString());
                                               public void Equal2()
                                                   TFrac fracClass1 =
        [TestMethod]
                                       new TFrac(0, 6);
                                                   TFrac fracClass2 =
        public void Square1()
                                       new TFrac(1, 6);
```

```
TFrac fracClass2 =
Assert.IsFalse(fracClass1.Equal(
                                     new TFrac(0, 1);
fracClass2));
                                      Assert.IsFalse(fracClass1 >
       }
                                      fracClass2);
        [TestMethod]
       public void Equal3()
                                              [TestMethod]
            TFrac fracClass1 =
                                              public void Greater3()
new TFrac(-1, 6);
           TFrac fracClass2 =
                                                 TFrac fracClass1 =
                                      new TFrac(-1, 6);
new TFrac(-1, 6);
                                                 TFrac fracClass2 =
Assert.IsTrue(fracClass1.Equal(f
                                     new TFrac(0, 1);
racClass2));
                                      Assert.IsFalse(fracClass1 >
       }
                                      fracClass2);
        [TestMethod]
                                             }
       public void Equal4()
                                              [TestMethod]
           TFrac fracClass1 =
                                              public void Greater4()
new TFrac(-1, 7);
           TFrac fracClass2 =
                                                 TFrac fracClass1 =
                                     new TFrac(17, 3);
new TFrac(1, 7);
                                                 TFrac fracClass2 =
Assert.IsFalse(fracClass1.Equal( new TFrac(16, 3);
fracClass2));
                                      Assert.IsTrue(fracClass1 >
       }
                                      fracClass2);
        [TestMethod]
                                             }
       public void Equal5()
                                              [TestMethod]
           TFrac fracClass1 =
                                              public void Greater5()
new TFrac(1, 6);
           TFrac fracClass2 =
                                                 TFrac fracClass1 =
new TFrac(0, 1);
                                     new TFrac(-2, 3);
                                                  TFrac fracClass2 =
Assert.IsFalse(fracClass1.Equal(
                                     new TFrac(-1, 3);
fracClass2));
                                     Assert.IsFalse(fracClass1 >
       }
                                     fracClass2);
        [TestMethod]
                                             }
       public void Greater1()
           TFrac fracClass1 =
                                         [TestClass]
                                          public class TFracEditorTest
new TFrac(1, 6);
            TFrac fracClass2 =
new TFrac(0, 1);
                                              [TestMethod]
                                              public void Init1()
Assert.IsTrue(fracClass1 >
fracClass2);
                                                  TFracEditor
                                     testClass = new TFracEditor();
       }
                                                 string input =
                                      "3/4";
        [TestMethod]
       public void Greater2()
                                                 testClass.Number =
                                      input;
           TFrac fracClass1 =
new TFrac(0, 1);
```

```
}
Assert.AreEqual(input,
testClass.Number);
                                                [TestMethod]
                                                public void Init6()
        [TestMethod]
        public void Init2()
                                                    TFracEditor
                                       testClass = new TFracEditor();
            TFracEditor
                                                    string input =
testClass = new TFracEditor();
                                       "666/6666";
            string input = "-
                                                    testClass.Number =
16/3";
                                       input;
            testClass.Number =
                                                    string result =
                                       "111/1111";
input;
                                       Assert.AreEqual(result,
Assert.AreEqual(input,
testClass.Number);
                                       testClass.Number);
        [TestMethod]
        public void Init3()
                                                [TestMethod]
                                                public void Init7()
            TFracEditor
testClass = new TFracEditor();
                                                    TFracEditor
            string input =
                                       testClass = new TFracEditor();
"0/8";
                                                    string input =
            testClass.Number =
                                       "aaaa";
                                                    testClass.Number =
input;
            string result =
                                       input;
"0/1";
                                                    string result =
                                       "0/1";
Assert.AreEqual(result,
testClass.Number);
                                       Assert.AreEqual(result,
                                       testClass.Number);
        [TestMethod]
        public void Init4()
                                                [TestMethod]
            TFracEditor
                                                public void Init8()
testClass = new TFracEditor();
            string input = "-
                                                    TFracEditor
17/4";
                                       testClass = new TFracEditor();
            testClass.Number =
                                                    string input =
                                        "0/1";
input;
                                                    testClass.Number =
Assert.AreEqual(input,
                                       input;
testClass.Number);
        }
                                       Assert.AreEqual(input,
                                       testClass.Number);
        [TestMethod]
                                               }
        public void Init5()
                                                [TestMethod]
            TFracEditor
                                                public void Init10()
testClass = new TFracEditor();
            string input =
                                                    TFracEditor
                                       testClass = new TFracEditor();
"0/1";
            testClass.Number =
                                                    string input =
                                        "16/000000";
input;
                                                    testClass.Number =
Assert.AreEqual(input,
                                       input;
testClass.Number);
```

```
string result =
"0/1";
                                              [TestMethod]
                                              public void
Assert.AreEqual(result,
                                     AddDeleteTest1()
testClass.Number);
                                                  TFracEditor
                                      testClass = new
        [TestMethod]
                                      TFracEditor("123/123");
        public void hasZero1()
                                     testClass.AddNumber(0);
            TFracEditor
                                      testClass.AddNumber(1);
testClass = new
TFracEditor("14/3");
                                      testClass.AddNumber(3);
Assert.AreEqual(false,
testClass.IsZero());
                                      testClass.AddSeparator();
        [TestMethod]
                                      testClass.ToggleMinus();
                                                  string result = "-
        public void hasZero2()
                                      1/1013";
            TFracEditor
testClass = new
                                      Assert.AreEqual(result,
TFracEditor("16/00000");
                                      testClass.ToString());
Assert.AreEqual(true,
                                              [TestMethod]
testClass.IsZero());
                                              public void
       }
                                      AddDeleteTest2()
        [TestMethod]
                                                  TFracEditor
       public void
                                      testClass = new TFracEditor(123,
ToogleMinus1()
                                      123);
            TFracEditor
                                      testClass.RemoveSymbol();
testClass = new
TFracEditor("14/3");
                                      testClass.RemoveSymbol();
testClass.ToggleMinus();
                                      testClass.RemoveSymbol();
           string result = "-
14/3";
                                      testClass.RemoveSymbol();
Assert.AreEqual(result,
                                      testClass.RemoveSymbol();
testClass.ToString());
                                      testClass.RemoveSymbol();
       [TestMethod]
        public void
                                      testClass.RemoveSymbol();
ToogleMinus2()
                                      testClass.AddNumber(1);
            TFracEditor
testClass = new TFracEditor("-
                                     testClass.AddNumber(2);
14/3");
                                     testClass.AddNumber(3);
testClass.ToggleMinus();
           string result =
                                     testClass.AddNumber(4);
"14/3";
                                     testClass.AddNumber(5);
Assert.AreEqual(result,
testClass.ToString());
                                     testClass.AddSeparator();
```

```
testClass.AddNumber(1);
                                      Assert.AreEqual(result,
                                       testClass.ToString());
testClass.AddNumber(1);
                                               [TestMethod]
testClass.AddNumber(1);
                                               public void Clear()
testClass.AddNumber(1);
                                                   TFracEditor
            string result =
                                      testClass = new
"12345/1111";
                                       TFracEditor("2345678/345678");
                                                  testClass.Clear();
Assert.AreEqual(result,
                                                   string result = "0";
testClass.ToString());
        }
                                      Assert.AreEqual(result,
                                      testClass.ToString());
        [TestMethod]
                                              }
        public void
                                           }
AddDeleteTest3()
                                           [TestClass]
            TFracEditor
                                           public class TPNumberTest
testClass = new
TFracEditor(1234567, 12345678);
                                               [TestMethod]
            for (int i = 0; i <
                                               public void Init1()
100; ++i)
                                                   TPNumber tPNumber =
testClass.RemoveSymbol();
                                      new TPNumber(10, 10, 3);
            for (int i = 0; i <
                                                   string answer =
100; ++i)
                                       "10";
testClass.AddSeparator();
                                      Assert.AreEqual(answer,
                                       tPNumber.ToString());
testClass.AddNumber(1);
                                               [TestMethod]
                                               public void Init2()
testClass.AddNumber(1);
testClass.AddNumber(1);
                                                   TPNumber tPNumber =
                                       new TPNumber("-12111.112", 3,
testClass.AddNumber(1);
                                       8);
            string result =
                                                  string answer = "-
                                       12111.112";
"1111";
Assert.AreEqual(result,
                                      Assert.AreEqual(answer,
testClass.ToString());
                                      tPNumber.ToString());
        [TestMethod]
                                               [TestMethod]
                                               public void Init3()
        public void
AddDeleteTest4()
                                                   TPNumber tPNumber =
        {
            TFracEditor
                                      new TPNumber(16, 16, 3);
testClass = new
                                                  string answer =
                                       "10":
TFracEditor("0/1");
            for (int i = 0; i <
100; ++i)
                                      Assert.AreEqual(answer,
                                      tPNumber.ToString());
testClass.AddNumber(i);
           string result =
                                               [TestMethod]
"123456789";
                                               public void Init4()
```

```
TPNumber tPNumber =
                                                   string answer =
new TPNumber(255, 2, 3);
                                      "100000000.01";
            string answer =
"11111111";
                                      Assert.AreEqual(answer,
                                      tPNumber.ToString());
Assert.AreEqual(answer,
tPNumber.ToString());
                                               [TestMethod]
                                              public void Init10()
        [TestMethod]
                                                  TPNumber tPNumber =
        public void Init5()
                                      new TPNumber (-356.22, 10, 5);
            TPNumber tPNumber =
                                                  string answer = "-
new TPNumber(14.471, 7, 5);
                                      356.22";
            string answer =
"20.32036";
                                      Assert.AreEqual(answer,
                                      tPNumber.ToString());
Assert.AreEqual(answer,
tPNumber.ToString());
                                               [TestMethod]
                                              public void Init11()
        [TestMethod]
        public void Init6()
                                                  TPNumber tPNumber =
                                      new TPNumber(0, 2, 1);
            TPNumber tPNumber =
                                                  string answer = "0";
new TPNumber(145.1742, 15, 8);
            string answer =
                                      Assert.AreEqual(answer,
"9A.292DD1D1";
                                      tPNumber.ToString());
                                              }
Assert.AreEqual(answer,
                                              [TestMethod]
tPNumber.ToString());
                                              public void Init12()
                                                  TPNumber tPNumber =
        [TestMethod]
        public void Init7()
                                      new TPNumber("A3.158", 12, 3);
                                                  string answer =
            TPNumber tPNumber =
                                      "A3.158";
new TPNumber(36.09, 16, 2);
            string answer =
                                      Assert.AreEqual(answer,
"24.17";
                                      tPNumber.ToString());
Assert.AreEqual(answer,
                                              [TestMethod]
tPNumber.ToString());
                                              public void Add1()
        [TestMethod]
                                                  TPNumber class1 =
        public void Init8()
                                     new TPNumber (15, 7, 5);
                                                  TPNumber class2 =
            TPNumber tPNumber =
                                      new TPNumber(66.66, 7, 5);
new TPNumber(88.88, 8, 9);
                                                  class1 =
                                      class1.Add(class2);
            string answer =
"130.70243656";
                                                  double answer =
                                      81.66;
Assert.AreEqual(answer,
tPNumber.ToString());
                                      Assert.AreEqual(answer,
                                      class1.Number);
        [TestMethod]
        public void Init9()
                                               [TestMethod]
                                              public void Add2()
            TPNumber tPNumber =
new TPNumber (256.256, 2, 5);
                                                  TPNumber class1 =
                                      new TPNumber(12, 16, 10);
```

```
TPNumber class2 =
                                                   TPNumber class1 =
new TPNumber (-13, 3, 6);
                                       new TPNumber(12.22, 16, 10);
            class1 =
                                                   TPNumber class2 =
                                       new TPNumber (-6.67, 7, 5);
class1.Add(class2);
            double answer = 0;
                                                   class1 =
                                       class1.Sub(class2);
Assert.AreEqual(answer,
                                                   double answer = 0;
class1.Number);
                                       Assert.AreEqual(answer,
                                       class1.Number);
        [TestMethod]
                                               }
        public void Multiply1()
                                               [TestMethod]
            TPNumber class1 =
                                               public void Divide1()
new TPNumber(12.23, 7, 5);
            TPNumber class2 =
                                                   TPNumber class1 =
new TPNumber (-6.66, 7, 5);
                                       new TPNumber(3, 7, 5);
            class1 =
                                                   TPNumber class2 =
class1.Mul(class2);
                                       new TPNumber(8, 7, 5);
            double answer = -
                                                   class1 =
81.4518;
                                       class1.Div(class2);
                                                   double answer =
Assert.AreEqual(answer,
                                       0.375;
class1.Number);
                                       Assert.AreEqual(answer,
        [TestMethod]
                                       class1.Number);
        public void Multiply2()
                                               }
                                               [TestMethod]
            TPNumber class1 =
                                               public void Divide2()
new TPNumber(15.6, 16, 11);
            TPNumber class2 =
                                                   TPNumber class1 =
new TPNumber(0, 7, 5);
                                       new TPNumber (666, 16, 5);
            class1 =
                                                   TPNumber class2 =
class1.Mul(class2);
                                       new TPNumber (-333, 2, 5);
            double answer = 0;
                                                   class1 =
                                       class1.Div(class2);
Assert.AreEqual(answer,
                                                   double answer = 0;
class1.Number);
                                       Assert.AreEqual(answer,
        }
                                       class1.Number);
        [TestMethod]
                                               }
        public void Substract1()
                                               [TestMethod]
            TPNumber class1 =
                                               public void Square1()
new TPNumber(12.23, 7, 5);
            TPNumber class2 =
                                                   TPNumber testClass =
new TPNumber(6.66, 7, 5);
                                       new TPNumber (-1.57, 7, 5);
            class1 =
                                                   testClass =
                                       testClass.Square() as TPNumber;
class1.Sub(class2);
            double answer =
                                                   double answer =
5.57;
                                       2.4649;
Assert.AreEqual(answer,
                                       Assert.AreEqual(answer,
class1.Number);
                                       testClass.Number);
                                               }
        [TestMethod]
        public void Substract2()
                                           [TestClass]
```

```
public class
                                                   string Input =
TPNumberEditorTest
                                       "aaaaaaaaaaaa";
                                                   edit.Number = Input;
        [TestMethod]
        public void Init1()
                                       [ExpectedException(typeof(Overfl
                                       owException))]
            TPNumberEditor edit
= new TPNumberEditor(0.0f, 16,
                                               [TestMethod]
                                               public void Init5()
            string Input =
"2.3F";
                                                   TPNumberEditor edit
            edit.Number = Input;
                                       = new TPNumberEditor(0.0f, 16,
            string Output =
                                       5);
Input;
                                                   string Input =
                                       "16.ppppp";
Assert.AreEqual(Output,
                                                   edit.Number = Input;
edit.Number);
                                               }
                                       [ExpectedException(typeof(Overfl
        [TestMethod]
                                       owException))]
        public void Init2()
                                               [TestMethod]
            TPNumberEditor edit
                                               public void Init6()
= new TPNumberEditor(0.0f, 2,
                                                   TPNumberEditor edit
5);
            string Input =
                                       = new TPNumberEditor(0.0f, 16,
                                       5);
"11101.01011";
            edit.Number = Input;
                                                   string Input =
            string Output =
                                       "FFZZZ";
Input;
                                                   edit.Number = Input;
Assert.AreEqual(Output,
                                               [TestMethod]
                                               public void Init7()
edit.Number);
                                                   TPNumberEditor edit
        [TestMethod]
        public void Init3()
                                       = new TPNumberEditor(0.0f, 16,
                                       5);
            TPNumberEditor edit
                                                   string Input =
= new TPNumberEditor(0.0f, 16,
                                       "888.8888";
                                                   edit.Number = Input;
5);
            string Input = "-
                                                   string Output =
FF.FF";
                                       Input;
            edit.Number = Input;
            string Output =
                                       Assert.AreEqual (Output,
                                       edit.Number);
Input;
Assert.AreEqual(Output,
                                               [TestMethod]
edit.Number);
                                               public void Init8()
        }
                                                   TPNumberEditor edit
[ExpectedException(typeof(Overfl
                                       = new TPNumberEditor(0.0f, 16,
owException))]
                                       5);
        [TestMethod]
                                                   string Input = "-
        public void Init4()
                                       01.03";
                                                   edit.Number = Input;
            TPNumberEditor edit
                                                   string Output = "-
= new TPNumberEditor(0.0f, 16,
                                       1.03";
5);
```

```
[TestMethod]
Assert.AreEqual(Output,
                                             public void
                                     AddDeleteTest2()
edit.Number);
                                             {
       [TestMethod]
                                                 TPNumberEditor
       public void Init9()
                                    testClass = new
                                     TPNumberEditor(0.0f, 16, 5);
                                                 for (int i = 0; i <
            TPNumberEditor edit
= new TPNumberEditor(0.0f, 16,
                                    16; ++i)
5);
           string Input = "- testClass.AddNumber(i);
9992D.DDDD";
                                                 string result =
           edit.Number = Input; "123456789ABCDEF";
           string Output =
Input;
                                     Assert.AreEqual(result,
                                     testClass.ToString());
Assert.AreEqual(Output,
edit.Number);
                                             [TestMethod]
                                             public void
        [TestMethod]
                                    AddDeleteTest3()
       public void Init10()
                                             {
                                                 TPNumberEditor
           TPNumberEditor edit
                                     testClass = new
= new TPNumberEditor(0.0f, 16,
                                     TPNumberEditor(0.0f, 16, 5);
                                                 for (int i = 0; i <
5);
           string Input =
                                    100; ++i)
"7777.7777";
           edit.Number = Input;
                                    testClass.RemoveSymbol();
           string Output =
"7777.7777";
                                     testClass.AddSeparator();
Assert.AreEqual(Output,
                                     testClass.AddSeparator();
edit.Number);
       }
                                     testClass.AddSeparator();
        [TestMethod]
                                    testClass.AddNumber(15);
       public void
                                                 string result =
                                     "0.F";
AddDeleteTest1()
        {
           TPNumberEditor
                                     Assert.AreEqual(result,
testClass = new
                                     testClass.ToString());
TPNumberEditor();
                                             [TestMethod]
testClass.AddNumber(0);
                                             public void
                                     AddDeleteTest4()
testClass.AddNumber(1);
                                                 TPNumberEditor
testClass.AddNumber(3);
                                     testClass = new
                                     TPNumberEditor(0.0f, 2, 5);
                                                for (int i = 2; i <
testClass.AddSeparator();
                                     100; ++i)
testClass.ToggleMinus();
           string result = "-
                                    testClass.AddNumber(i);
13.";
                                                 string result = "0";
Assert.AreEqual(result,
                                     Assert.AreEqual(result,
testClass.ToString());
                                     testClass.ToString());
       }
                                             }
```

```
}
                                      Assert.AreEqual(output,
    [TestClass]
                                      testClass.ToString());
    public class TComplexTest
                                              [TestMethod]
        [TestMethod]
                                              public void Init6()
        public void Init1()
                                                   TComplex testClass =
            TComplex testClass =
                                     new TComplex(0, -22.22);
new TComplex(6, 3);
                                                  string output = "0 +
            string output = "6 +
                                      i * -22,22";
i * 3";
                                      Assert.AreEqual(output,
Assert.AreEqual(output,
                                      testClass.ToString());
testClass.ToString());
        [TestMethod]
                                               [TestMethod]
        public void Init2()
                                              public void Reverse1()
            TComplex testClass =
                                                   TComplex testClass =
new TComplex(0, 0);
                                      new TComplex(0, -3);
            string output = "0 +
                                                  testClass =
i * 0";
                                      testClass.Reverse() as TComplex;
                                                  string output = "0 +
Assert.AreEqual(output,
                                      i * 0,33333333333333";
testClass.ToString());
                                      Assert.AreEqual(output,
                                      testClass.ToString());
        [TestMethod]
        public void Init3()
                                              [TestMethod]
            TComplex testClass =
                                              public void Reverse2()
new TComplex(3, -2);
            string output = "3 +
                                                  TComplex testClass =
i * -2";
                                      new TComplex(3, 4);
                                                  testClass =
Assert.AreEqual(output,
                                      testClass.Reverse() as TComplex;
testClass.ToString());
                                                  string output =
                                      "0,12 + i * -0,16";
        [TestMethod]
        public void Init4()
                                      Assert.AreEqual (output,
                                      testClass.ToString());
            TComplex testClass =
new TComplex(4.01, 6);
                                               [TestMethod]
            string output =
                                              public void Square1()
"4,01 + i * 6";
                                                  TComplex testClass =
                                      new TComplex(3, 4);
Assert.AreEqual(output,
testClass.ToString());
                                                  testClass =
                                      testClass.Square() as TComplex;
        [TestMethod]
                                                  string output = "-7
        public void Init5()
                                      + i * 24";
            TComplex testClass =
                                      Assert.AreEqual(output,
new TComplex(-3.02, 7);
                                      testClass.ToString());
            string output = "-
3,02 + i * 7";
                                               [TestMethod]
                                              public void Square2()
```

```
TComplex testClass =
                                                  double answer =
new TComplex(0, -3);
                                      53.1301;
            testClass =
testClass.Square() as TComplex;
                                      Assert.AreEqual(answer,
           string output = "-9
                                      testClass.Degree(), 4);
+ i * -0";
                                              [TestMethod]
Assert.AreEqual (output,
                                              public void GetDegree2()
testClass.ToString());
                                                  TComplex testClass =
        [TestMethod]
                                      new TComplex(0, -3);
        public void Abs1()
                                                  double answer = -90;
            TComplex testClass =
                                      Assert.AreEqual(answer,
                                      testClass.Degree(), 4);
new TComplex(3, 4);
            double answer = 5;
                                              [TestMethod]
Assert.AreEqual(answer,
                                              public void Pow1()
testClass.Abs(), 4);
       }
                                                   TComplex testClass =
        [TestMethod]
                                      new TComplex(3, 4);
        public void Abs2()
                                                  testClass =
                                      testClass.Pow(5);
            TComplex testClass =
                                                  string output = "-
new TComplex(0, -3);
                                      237 + i * -3116";
            double answer = 3;
                                      Assert.AreEqual(output,
                                      testClass.ToString());
Assert.AreEqual(answer,
testClass.Abs(), 4);
                                              [TestMethod]
        [TestMethod]
                                              public void Pow2()
        public void GetRad1()
                                                  TComplex testClass =
            TComplex testClass = new TComplex(0, -3);
                                                  testClass =
new TComplex(3, 4);
            double answer =
                                      testClass.Pow(5);
0.927295;
                                                  string output = "0 +
                                      i * -243";
Assert.AreEqual(answer,
testClass.Rad(), 4);
                                      Assert.AreEqual (output,
                                      testClass.ToString());
        [TestMethod]
        public void GetRad2()
                                              [TestMethod]
                                              public void Root1()
            TComplex testClass =
new TComplex(0, -3);
                                                  TComplex testClass =
            double answer = -
                                     new TComplex(3, 4);
1.570796;
                                                  testClass =
                                      testClass.Sqrt(5, 3);
Assert.AreEqual(answer,
                                                  string output = "-
testClass.Rad(), 4);
                                      0.353 + i * 1.334";
        [TestMethod]
                                      Assert.AreEqual(output,
        public void GetDegree1()
                                      testClass.ToString());
            TComplex testClass =
                                              [TestMethod]
new TComplex(3, 4);
                                              public void Root2()
```

```
TComplex testClass =
                                                   string output = "-0
new TComplex(16, -2);
                                      + i * -0";
            testClass =
testClass.Sqrt(10, 4);
                                      Assert.AreEqual (output,
            string output = "-
                                      testClass.ToString());
0.301 + i * 1.286";
                                               [TestMethod]
Assert.AreEqual(output,
                                               public void Init4()
testClass.ToString());
                                                   TComplexEditor
                                      testClass = new
                                      TComplexEditor();
    [TestClass]
                                                  string input =
                                       "66,66 + i * 66,66";
    public class
                                                  testClass.Number =
TComplexEditorTest
                                      input;
    {
        [TestMethod]
                                                   string output =
                                      "66,66 + i * 66,66";
        public void Init1()
            TComplexEditor
                                      Assert.AreEqual(output,
testClass = new
                                      testClass.ToString());
TComplexEditor();
            string input =
                                               [TestMethod]
"10,3";
                                               public void Init5()
            testClass.Number =
                                                   TComplexEditor
input;
                                      testClass = new
            string output =
"10,3 + i * 0";
                                      TComplexEditor();
                                                   string input = "10.3"
Assert.AreEqual(output,
testClass.ToString());
                                                   testClass.Number =
                                      input;
        [TestMethod]
                                                   string output = "0 +
        public void Init2()
                                      i * 0";
            TComplexEditor
                                      Assert.AreEqual(output,
testClass = new
                                      testClass.ToString());
TComplexEditor();
                                              }
            string input = "10,
+ i * -12,";
                                              [TestMethod]
            testClass.Number =
                                              public void
                                      AddDeleteTest1()
input;
            string output = "10
+ i * -12";
                                                   TComplexEditor
                                      testClass = new
Assert.AreEqual(output,
                                      TComplexEditor();
testClass.ToString());
                                      testClass.AddNumber(0);
        [TestMethod]
        public void Init3()
                                      testClass.ToggleMinus();
            TComplexEditor
                                      testClass.AddNumber(1);
testClass = new
TComplexEditor();
                                      testClass.AddNumber(3);
            string input = "-0 +
i * -0";
                                      testClass.AddSeparator();
            testClass.Number =
                                                   string result = "-13
                                      + i * 0";
input;
```

```
Assert.AreEqual(result,
                                      Assert.AreEqual(result,
testClass.ToString());
                                      testClass.ToString());
        [TestMethod]
        public void
AddDeleteTest2()
                                           [TestClass]
                                          public class TMemoryTest
            TComplexEditor
testClass = new
                                               [TestMethod]
TComplexEditor();
                                              public void
                                      InitAndOutput1()
testClass.AddNumber(0);
                                                   TFrac frac = new
testClass.AddNumber(1);
                                      TFrac(22, 33);
                                                  TMemory<TFrac>
testClass.AddNumber(3);
                                      memory = new
                                      TMemory<TFrac>(frac);
testClass.AddSeparator();
                                                  string answer =
                                       "2/3";
testClass.ToggleMinus();
                                      Assert.AreEqual (answer,
testClass.RemoveSymbol();
                                      memory.FNumber.ToString());
           string result = "13
+ i * -0";
                                              [TestMethod]
                                              public void
Assert.AreEqual(result,
                                      InitAndOutput2()
testClass.ToString());
                                              {
                                                  TFrac frac = new
       }
       [TestMethod]
                                      TFrac();
        public void
                                                  TMemory<TFrac>
AddDeleteTest3()
                                      memory = new
                                      TMemory<TFrac>(frac);
            TComplexEditor
                                                  string answer =
                                       "0/1";
testClass = new
TComplexEditor();
                                      Assert.AreEqual(answer,
testClass.AddNumber(0);
                                      memory.FNumber.ToString());
testClass.AddNumber(1);
                                               [TestMethod]
                                              public void
testClass.AddNumber(3);
                                      InitAndOutput3()
                                                   TFrac frac = new
testClass.AddNumberSeparator();
                                      TFrac(-1, 5);
testClass.AddSeparator();
                                                  TMemory<TFrac>
                                      memory = new
testClass.ToggleMinus();
                                      TMemory<TFrac>(frac);
                                                  string answer = "-
                                      1/5";
testClass.AddNumber(0);
testClass.AddNumber(1);
                                      Assert.AreEqual(answer,
                                      memory.FNumber.ToString());
testClass.AddNumber(3);
testClass.AddNumberSeparator();
                                              [TestMethod]
          string result = "13.
                                              public void Sum1()
+ i * -13.";
```

```
TFrac frac = new
                                                   TMemory<TFrac>
                                      memory = new
TFrac(-1, 5);
                                       TMemory<TFrac>(frac);
            TMemory<TFrac>
                                                  bool expected =
memory = new
TMemory<TFrac>(frac);
                                       false;
            TFrac summator = new
TFrac(1, 2);
                                      Assert.AreEqual (expected,
                                      memory.FState);
memory.Add(summator);
           string answer =
"3/10";
                                              [TestMethod]
                                              public void
Assert.AreEqual(answer,
                                      TestFState3()
memory.FNumber.ToString());
                                                  TFrac frac = new
                                      TFrac(8, 9);
        [TestMethod]
                                                  TMemory<TFrac>
        public void Sum2()
                                      memory = new
                                      TMemory<TFrac>(frac);
            TFrac frac = new
                                                  memory.Add(frac);
TFrac(8, 9);
                                                  bool expected =
            TMemory<TFrac>
                                      true;
memory = new
TMemory<TFrac>(frac);
                                      Assert.AreEqual(expected,
                                      memory.FState);
            TFrac summator = new
TFrac(-16, 3);
memory.Add(summator);
           string answer = "-
                                           [TestClass]
40/9";
                                          public class TProcTest
Assert.AreEqual(answer,
                                               [TestMethod]
memory.FNumber.ToString());
                                              public void Init1()
       }
                                                   TFrac leftFrac = new
        [TestMethod]
                                      TFrac();
        public void
                                                  TFrac rightFrac =
TestFState1()
                                      new TFrac();
                                                  ADT Proc<TFrac> proc
                                      = new ADT Proc<TFrac>(leftFrac,
            TFrac frac = new
TFrac(8, 9);
                                      rightFrac);
            TMemory<TFrac>
                                                  string answer =
                                      "0/1";
memory = new
TMemory<TFrac>(frac);
            memory.Clear();
                                      Assert.AreEqual (answer,
            bool expected =
                                      proc.Left Result operand.ToStrin
false;
                                      g());
Assert.AreEqual(expected,
                                      Assert.AreEqual(answer,
memory.FState);
                                      proc.Right operand.ToString());
        [TestMethod]
                                               [TestMethod]
        public void
                                              public void Init2()
TestFState2()
                                                  TFrac leftFrac = new
            TFrac frac = new TFrac(11, 3);
TFrac(8, 9);
```

```
TFrac rightFrac =
                                                   ADT Proc<TFrac> proc
                                       = new ADT Proc<TFrac>(leftFrac,
new TFrac();
            ADT Proc<TFrac> proc
                                       rightFrac);
= new ADT Proc<TFrac>(leftFrac,
                                                   proc.Operation =
rightFrac);
                                       ADT_Proc<TFrac>.Operations.Sub;
                                                   proc.DoOperation();
            string answer =
"11/3";
                                                    string answer = "-
                                       1/12";
Assert.AreEqual(answer,
proc.Left Result operand.ToStrin
                                       Assert.AreEqual(answer,
                                       proc.Left Result operand.ToStrin
g());
                                       g());
        [TestMethod]
        public void Init3()
                                                [TestMethod]
                                               public void Operation3()
            TFrac leftFrac = new
TFrac(16, 4);
                                                    TFrac leftFrac = new
            TFrac rightFrac =
                                       TFrac(12, 7);
new TFrac(17, 9);
                                                   TFrac rightFrac =
            ADT Proc<TFrac> proc
                                       new TFrac(5, 9);
= new ADT Proc<TFrac>(leftFrac,
                                                   ADT Proc<TFrac> proc
rightFrac);
                                       = new ADT Proc<TFrac>(leftFrac,
                                       rightFrac);
            string answer =
"17/9";
                                                   proc.Operation =
                                       ADT Proc<TFrac>.Operations.Mul;
Assert.AreEqual(answer,
                                                   proc.DoOperation();
proc.Right operand.ToString());
                                                    string answer =
                                       "20/21";
        }
        [TestMethod]
                                       Assert.AreEqual(answer,
        public void Operation1()
                                       proc.Left Result operand.ToStrin
                                       g());
            TFrac leftFrac = new
TFrac(1, 2);
            TFrac rightFrac =
                                                [TestMethod]
                                               public void Operation4()
new TFrac(1, 2);
            ADT Proc<TFrac> proc
= new ADT Proc<TFrac>(leftFrac,
                                                    TFrac leftFrac = new
rightFrac);
                                       TFrac(56, 7);
            proc.Operation =
                                                   TFrac rightFrac =
ADT Proc<TFrac>.Operations.Add;
                                       new TFrac(-22, 3);
            proc.DoOperation();
                                                   ADT Proc<TFrac> proc
            string answer =
                                       = new ADT Proc<TFrac>(leftFrac,
"1/1";
                                       rightFrac);
                                                   proc.Operation =
Assert.AreEqual(answer,
                                       ADT Proc<TFrac>.Operations.Div;
proc.Left Result operand.ToStrin
                                                   proc.DoOperation();
                                                    string answer = "-
g());
                                       12/11";
        [TestMethod]
                                       Assert.AreEqual(answer,
        public void Operation2()
                                       proc.Left Result operand.ToStrin
                                       g());
            TFrac leftFrac = new
TFrac(3, 4);
                                               [TestMethod]
            TFrac rightFrac =
                                               public void
                                       TestFState1()
new TFrac(5, 6);
```

```
TFrac leftFrac = new
                                                   TFrac leftFrac = new
TFrac(56, 7);
                                       TFrac(56, 7);
                                                   TFrac rightFrac =
            TFrac rightFrac =
                                       new TFrac(-22, 3);
new TFrac(-22, 3);
           ADT Proc<TFrac> proc
                                                   ADT Proc<TFrac> proc
= new ADT Proc<TFrac>(leftFrac,
                                       = new ADT Proc<TFrac>(leftFrac,
rightFrac);
                                       rightFrac);
proc.DoFunction(ADT_Proc<TFrac>.
                                       proc.DoFunction(ADT_Proc<TFrac>.
Functions.Rev);
                                       Functions.Sqr);
            string answer = "-
                                                   string answer =
                                       "484/9";
3/22";
                                       Assert.AreEqual(answer,
Assert.AreEqual(answer,
proc.Right_operand.ToString());
                                       proc.Right operand.ToString());
                                               }
        [TestMethod]
        public void
                                       }
TestFState2()
```

## Результат работы тестов

▲ 🕢 calculator_tests (140)	64 мс
	64 мс
▷ 🕢 TComplexEditorTest (8)	10 мс
▷ 🕢 TComplexTest (16)	< 1 мс
▷ 🕢 TFracEditorTest (18)	1 мс
▷ 🕢 TFracTest (46)	< 1 мс
▷ 🕢 TMemoryTest (8)	46 мс
D	3 мс
Delta TPNumberTest (21)	< 1 мс
▷ Ø TProcTest (9)	4 мс

## 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 С#: 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.