Федеральное государственное бюджетное образовательное учреждение высшего образования

«Сибирский государственный университет

телекоммуникаций и информатики»

Кафедра ПМиК

Современные технологии программирования 2

Расчетно-графическое задание

Вариант 13

Выполнил: студент 4 курса

Ф. ИВТ, группа: ИП-711

Мартасов И. О.

Проверил: ассистент кафедры ПМиК

Агалаков Антон Александрович

Новосибирск, 2021

**Содержание**

1. Задание……………………………………………………………………………..3
2. Диаграмма прецедентов UML. Сценарии прецедентов………………………...8
3. Диаграмма последовательностей для прецедентов……………………………..8
4. Диаграмма классов для прецедентов…………………………………………...19
5. Спецификации к типам данных…………………………………………………20
6. Тестовые наборы данных для тестирования абстрактных типов данных, классов и приложения…………………………………………………………...44
7. Заключение……………………………………………………………………….46
8. Текст программы………………………………………………………………...47

**Задание**

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

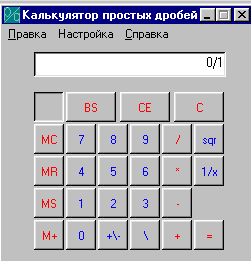
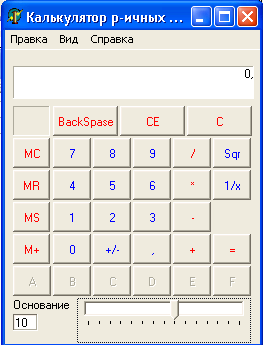
Вариант чисел - простые дроби.

Общие требования

Калькулятор обеспечивает вычисление выражений с использованием операций: +, -, \*. / и функций: Sqr (возведение в

квадрат), Rev (1/x - вычисление обратного значения) без учёта приоритета операций. Приоритет функций одинаковый, выше приоритета операций. Операции имеют равный приоритет.

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

1. Приложение должно иметь основное окно для ввода исходных данных, операций и отображения результата и окно для вывода сведений о разработчиках приложения.
2. Основное окно должно содержать список из трёх меню:
   * Правка:

Содержит два пункта: «Копировать» и «Вставить». Эти команды используются для работы с буфером обмена;

* + Настройка:

Содержит команды выбора режима работы приложения;

* + Справка:

Этот команда для вызова справки о приложении.

1. Калькулятор должен обеспечивать возможность ввода исходных данных с помощью:
   * командных кнопок (мышью),
   * клавиатуры: цифровой и алфавитно-цифровой.
2. Вводимые числа выравнивать по правому краю.
3. Калькулятор должен быть снабжён памятью. Для работы с памятью необходимы команды:
   * MC («Очистить»),
   * MS («Сохранить»),
   * MR («Копировать»),
   * M+ («Добавить к содержимому памяти»).

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

* «Включена» (M). В памяти храниться занесённое значение
* «Выключена» ( ). В памяти находится ноль.

Состояние памяти меняется командами «Сохранить» и

«Добавить к содержимому памяти».

1. Для редактирования вводимых значений необходимы команды:
   * BackSpase (удалить крайний справа символ отображаемого числа),
   * CE (заменить отображаемое число нулевым значением)
   * Добавить символ, допустимый в изображении числа (арабские цифры, знак, разделители).
2. Для просмотра выполненных за сеанс вычислений калькулятор необходимо снабдить «Историей».

13. Снабдите компоненты интерфейса всплывающими подсказками.

**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>::= ‘/’ | ‘|’

* 1. 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.

1. Выполнение операций с одним операндом:

«операнд» «операция» «=» «результат» Пример. 5/1 \* = 25/1.

1. Повторное выполнение операции:

«=»«результат» «=» «результат» Пример. 5/1 + 4/1 = 9/1 = 13/1 = 17.

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

«результат» «операция» «=» «результат» Пример. 2/1 + 3/1 = 5/1 = 8/1 + = 16/1.

1. Вычисление функций:

«операнд» «Sqr» «результат» Пример. 5/1 «Sqr» 25/1.

1. Вычисление выражений:

«операнд1» «функция1» «операция1» «операнд2» «функция2»

«операция2» …«операндN» «операцияN» «=»«результат» Пример.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ввод | 6/1 | Sqr | + | 2/1 | Sqr | / | 10/1 | + | 6/1 | = |
| Отображаемы  й результат | 6/1 | 36/1 | 36/1 | 2/1 | 4/1 | 40/1 | 10/1 | 4/1 | 6/1 | 10/1 |

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

**Диаграмма прецедентов UML. Сценарии прецедентов**

Пользователь

**Сценарий для прецедента Выполнение одиночных операций:**

**Основной поток событий:**

1. Пользователь вводит первый операнд, представленный в виде простой дроби.
2. Пользователь вводит арифметическое выражение (+,-,\*,/)
3. Пользователь вводит второй операнд, представленный в виде простой дроби.
4. Пользователь нажимает кнопку Вычислить результат (=)
5. Приложение выводит на экран результат операции.

**Альтернативный поток событий 1. Результат выполнения операции выходит за границы допустимого диапазона.**

1.1. Пользователь получает окно с сообщением ERROR.

1.2. Приложение переходит в режим Ввод и редактирование.

**Альтернативный поток событий 2. При выполнении операции деления вторым операндом являлся ноль.**

2.1. Пользователь получает окно с сообщением ERROR.

2.2. Приложение переходит в режим Ввод и редактирование

**Альтернативный поток событий 3. При вводе операндов пользователь не ввел знаменатель.**

3.1. Операнд заменится на ноль (0/1)

3.2. Приложение переходит в режим Ввод и редактирование

**Сценарий для прецедента Выполнение операций с одним операндом:**

**Основной поток событий:**

1. Пользователь вводит первый операнд, представленный в виде простой дроби.
2. Пользователь вводит арифметическое выражение (+,-,\*,/)
3. Пользователь нажимает кнопку Вычислить результат (=)
4. Приложение выводит на экран результат операции.

**Альтернативный поток событий 1. Результат выполнения операции выходит за границы допустимого диапазона.**

1.1. Пользователь получает окно с сообщением ERROR.

1.2. Приложение переходит в режим Ввод и редактирование.

**Альтернативный поток событий 2. При выполнении операции деления вторым операндом являлся ноль.**

2.1. Пользователь получает окно с сообщением ERROR.

2.2. Приложение переходит в режим Ввод и редактирование

**Альтернативный поток событий 3. При вводе операндов пользователь не ввел знаменатель.**

3.1. Операнд заменится на ноль (0/1)

3.2. Приложение переходит в режим Ввод и редактирование

**Сценарий для прецедента Повторное выполнение операции:**

**Предусловие**

**Прецедент Выполнение операций с одним операндом или Выполнение одиночных операций завершен.**

**Основной поток событий:**

1. Пользователь нажимает кнопку Вычислить результат (=)
2. Приложение выводит на экран результат операции.

**Альтернативный поток событий 1. Результат выполнения операции выходит за границы допустимого диапазона.**

1.1. Пользователь получает окно с сообщением ERROR.

1.2. Приложение переходит в режим Ввод и редактирование.

**Альтернативный поток событий 2. При выполнении операции деления вторым операндом являлся ноль.**

2.1. Пользователь получает окно с сообщением ERROR.

2.2. Приложение переходит в режим Ввод и редактирование

**Альтернативный поток событий 3. При вводе операндов пользователь не ввел знаменатель.**

3.1. Операнд заменится на ноль (0/1)

3.2. Приложение переходит в режим Ввод и редактирование

**Сценарий для прецедента Выполнение операции над отображаемым значением в качестве обоих операндов:**

**Предусловие**

**Прецедент Выполнение операций с одним операндом или Выполнение одиночных операций или Повторное выполнение операции завершен.**

**Основной поток событий:**

1. Пользователь вводит арифметическое выражение (+,-,\*,/)
2. Пользователь нажимает кнопку Вычислить результат (=)
3. Приложение выводит на экран результат операции.

**Альтернативный поток событий 1. Результат выполнения операции выходит за границы допустимого диапазона.**

1.1. Пользователь получает окно с сообщением ERROR.

1.2. Приложение переходит в режим Ввод и редактирование.

**Альтернативный поток событий 2. При выполнении операции деления вторым операндом являлся ноль.**

2.1. Пользователь получает окно с сообщением ERROR.

2.2. Приложение переходит в режим Ввод и редактирование

**Альтернативный поток событий 3. При вводе операндов пользователь не ввел знаменатель.**

3.1. Операнд заменится на ноль (0/1)

3.2. Приложение переходит в режим Ввод и редактирование

**Сценарий для прецедента Вычисление функций:**

**Основной поток событий:**

1. **Пользователь вводит операнд, представленный в виде простой дроби.**
2. **Пользователь нажимает кнопку Вычислить функцию (SQR или 1/x)**
3. **Приложение выводит на экран результат выполнения функции.**

**Альтернативный поток событий 1. Результат вычисления функции выходит за границы допустимого диапазона.**

1.1. Пользователь получает окно с сообщением ERROR.

1.2. Приложение переходит в режим Ввод и редактирование.

**Альтернативный поток событий 2. При выполнении функции 1/X числитель был равен нулю.**

2.1. Пользователь получает окно с сообщением ERROR.

2.2. Приложение переходит в режим Ввод и редактирование

**Альтернативный поток событий 3. При вводе операндов пользователь не ввел знаменатель.**

3.1. Операнд заменится на ноль (0/1)

3.2. Приложение переходит в режим Ввод и редактирование

**Сценарий для прецедента Вычисление выражений:**

**Основной поток событий:**

1. **Пользователь вводит первый операнд, представленный в виде простой дроби.**
2. **Пользователь нажимает кнопку Вычислить функцию (SQR или 1/x)**
3. **Пользователь вводит арифметическое выражение (+,-,\*,/)**
4. **Пользователь вводит первый операнд, представленный в виде простой дроби.**
5. **Пользователь нажимает кнопку Вычислить результат (=)**
6. **Приложение выводит на экран результат вычисления выражения.**

**Альтернативный поток событий 1. Результат вычисления функции или операции выходит за границы допустимого диапазона.**

1.1. Пользователь получает окно с сообщением ERROR.

1.2. Приложение переходит в режим Ввод и редактирование.

**Альтернативный поток событий 2. При выполнении операции деления вторым операндом являлся ноль.**

2.1. Пользователь получает окно с сообщением ERROR.

2.2. Приложение переходит в режим Ввод и редактирование

**Альтернативный поток событий 3. При выполнении функции 1/X числитель был равен нулю.**

3.1. Пользователь получает окно с сообщением ERROR.

3.2. Приложение переходит в режим Ввод и редактирование

**Альтернативный поток событий 4. При вводе операндов пользователь не ввел знаменатель.**

4.1. Операнд заменится на ноль (0/1)

4.2. Приложение переходит в режим Ввод и редактирование

**Диаграмма последовательностей для прецедентов**

## **Выполнение одиночных операций**

TFrac

:Простая дробь

TProc

:Процессор

TFracEditor

:Редактор

Form1.cs

:Интерфейс

TCtrl

:Управление

:Пользователь

|ВводЧисла() | | | | |

| |ВыполнитьРед() | | | |

| | |Редактировать()|Редактировать() |

|ВводОперации() | | | |

| |ВыполнитьОп() | | | |

| | |ВыполнитьОперацию() | |

| | |Очистить() | |Операция() |

|ВводЧисла() | | | | |

| |ВыполнитьРед() | | | |

| | |Редактировать()|Редактировать() | | | | | | |

|Результат() | | | | |

| |Расчитать() | | | |

| | |ВыполнитьОперацию() | |

| | | | |Операция() |

## **Выполнение операций с одним операндом**

:Пользователь

TFrac

:Простая дробь

TProc

:Процессор

TFracEditor

:Редактор

TCtrl

:Управление

Form1.cs

:Интерфейс

|ВводЧисла() | | | | |

| |ВыполнитьРед() | | | |

| | |Редактировать()|Редактировать() |

|ВводОперации() | | | |

| |ВыполнитьОп() | | | |

| | |ВыполнитьОперацию() | |

| | |Очистить() | |Операция()

|Результат() | | | | |

| |Расчитать() | | | |

| | |ВыполнитьОперацию() | |

| | | | |Операция() |

## **Повторное выполнение операции**

:Пользователь

TFrac

:Простая дробь

TProc

:Процессор

TFracEditor

:Редактор

TCtrl

:Управление

Form1.cs

:Интерфейс

|ВводЧисла() | | | | |

| |ВыполнитьРед() | | | |

| | |Редактировать()|Редактировать() |

|ВводОперации() | | | |

| |ВыполнитьОп() | | | |

| | |ВыполнитьОперацию() | |

| | |Очистить() | |Операция()

|Результат() | | | | |

| |Расчитать() | | | |

| | |ВыполнитьОперацию() | |

| | | | |Операция() |

|Результат() | | | | |

| |Расчитать() | | | |

| | |ВыполнитьОперацию() | |

| | | | |Операция() |

**Выполнение операции над отображаемым значением в качестве обоих операндов**

:Пользователь

TFrac

:Простая дробь

TProc

:Процессор

TFracEditor

:Редактор

TCtrl

:Управление

Form1.cs

:Интерфейс

|ВводЧисла() | | | | |

| |ВыполнитьРед() | | | |

| | |Редактировать()|Редактировать() |

|ВводОперации() | | | |

| |ВыполнитьОп() | | | |

| | |ВыполнитьОперацию() | |

| | |Очистить() | |Операция()

|Результат() | | | | |

| |Расчитать() | | | |

| | |ВыполнитьОперацию() | |

| | | | |Операция() |

|ВводОперации() | | | |

| |ВыполнитьОп() | | | |

| | |ВыполнитьОперацию() | |

| | |Очистить() | |Операция()

|Результат() | | | | |

| |Расчитать() | | | |

| | |ВыполнитьОперацию() | |

| | | | |Операция() |

## **Вычисление функции**

:Пользователь

TFrac

:Простая дробь

TProc

:Процессор

TFracEditor

:Редактор

TCtrl

:Управление

Form1.cs

:Интерфейс

|ВводЧисла() | | | | |

| |ВыполнитьРед() | | | |

| | |Редактировать()|Редактировать() |

|ВводФункции() | | | |

| |ВыполнитьФун() | | | |

| | |ВыполнитьФункцию() | |

| | | | |Функция()

|Результат() | | | | |

| |Расчитать() | | | |

| | |ВыполнитьОперацию() | |

| | | | |Операция() |

## **Вычисление выражений**

:Пользователь

TFrac

:Простая дробь

TProc

:Процессор

TFracEditor

:Редактор

TCtrl

:Управление

Form1.cs

:Интерфейс

|ВводЧисла() | | | | |

| |ВыполнитьРед() | | | |

| | |Редактировать()|Редактировать() |

|ВводФункции() | | | |

| |ВыполнитьФун() | | | |

| | |ВыполнитьФункцию() | |

| | | | |Функция()

|Результат() | | | | |

| |Расчитать() | | | |

| | |ВыполнитьОперацию() | |

| | | | |Операция() |

|ВводОперации() | | | |

| |ВыполнитьОп() | | | |

| | |ВыполнитьОперацию() | |

| | |Очистить() | |Операция() |

|ВводЧисла() | | | | |

| |ВыполнитьРед() | | | |

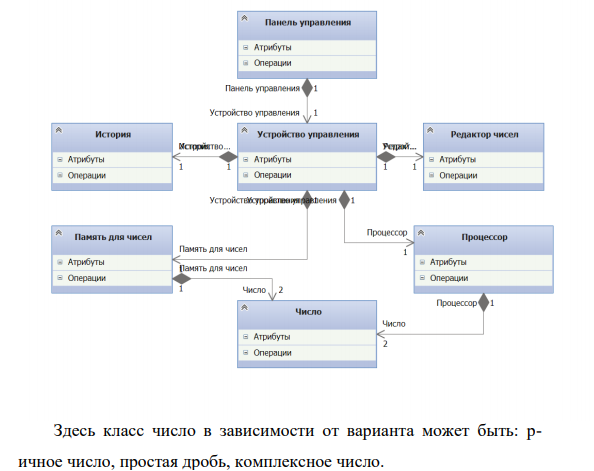
| | |Редактировать()|Редактировать() | | | | | | |

|Результат() | | | | |

| |Расчитать() | | | |

| | |ВыполнитьОперацию() | |

| | | | |Операция() |

**Диаграмма классов для прецедентов**

**Спецификации к типам данных**

Спецификация типа данных «простые дроби».

# ADT TFrac

**Данные**

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

# Операции

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

|  |  |
| --- | --- |
| ***Конструктор*** |  |
| Начальные значения: | Пара целых чисел (a) и (b). |
| Процесс: | Инициализирует поля простой дроби  (тип TFrac): числитель значением a, |
|  | знаменатель - (b). В случае необходимости дробь предварительно сокращается.  Например:  *Конструктор*(6,3) = (2/1)  *Конструктор*(0,3) = (0/3). |
|  | |
| ***Конструктор*** |  |
| Начальные значения: | Строковое представление простой дроби  . Например: ‘7/9’. |
| Процесс: | Инициализирует поля простой дроби (тип TFrac) строкой f =’a/b’. Числитель значением a, знаменатель - b. В случае необходимости дробь предварительно сокращается.  Например:  *Конструктор*(‘6/3’) = 2/1  *Конструктор* (‘0/3’) = 0/3 |
|  | |
| **Копировать:** |  |
| Вход: | Нет. |
| Предусловия: | Нет. |
| Процесс: | Создаёт копию самой дроби (тип TFrac) с числителем, и знаменателем такими же, как у самой дроби. |
| Выход: | Простая дробь (тип TFrac).  Например:  c = 2/1, Копировать(c) = 2/1 |

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

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

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

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

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

*end TFracRatio*

Спецификация типа данных Редактор простых дробей

На Унифицированном языке моделирования UML (Unified Modeling Language) класс можно обозначить следующим образом:

|  |
| --- |
| **РедакторКомплексныхЧисел** |
| строка: String |
| комплексноеЧислоЕстьНоль: Boolean |
| добавитьЗнак: String |
| добавитьЦифру(a: Integer): String |
| добавитьНоль: String |
| забойСимвола: String |
| очистить: String |
| конструктор |
| читатьСтрокаВформатеСтроки: String (метод свойства) |
| писатьСтрокаВформатеСтроки(a: String) (метод свойства) |
| редактировать(a: Integer): String |
| Обязанность:  ввод, хранение и редактирование строкового представления комплексных чисел |

Спецификация типа данных «память».

**ADT** TMemory

# Данные

Память (тип TMemory, в дальнейшем - память) - это память для хранения

«числа» объекта типа T в поле FNumber, и значения «состояние памяти» в поле FState. Объект память - изменяемый. Он имеет два состояния, обозначаемых значениями: «Включена» (\_On), «Выключена» (\_Off). Её изменяют операции: Записать (Store), Добавить (Add), Очистить (Clear).

# Операции

|  |  |
| --- | --- |
| ***Конструктор*** |  |
| Начальные значения: | Нет. |
| Процесс: | Инициализирует поле FNumber объекта  «память» (тип TMemory) объектом «число» (тип T) со значением по умолчанию. Например для числа типа TFrac со значением 0/1. Память устанавливается в состояние «Выключена», в поле FState  «состояние памяти» заносится значение  (\_Off). |
| ***Записать*** |  |
| Вход: | (E) – объект тип T. |
| Предусловия: | Нет. |

|  |  |
| --- | --- |
| Процесс: | В объект «память» (тип TMemory) в поле FNumber записывается копия объекта Е. Память устанавливается в состояние  «Включена», в поле FState «состояние  памяти» заносится значение (\_On). |
| Выход: | Нет. |
| Постусловия: | Состояние памяти поле FState –  «Включена» (\_On). |
|  |  |
| ***Взять*** |  |
| Вход: | Нет. |
| Предусловия: | Нет. |
| Процесс: | Создаёт и возвращает копию объекта хранящегося в объекте «память» (тип  TMemory) в поле FNumber. |
| Выход: | Объект типа T. |
| Постусловия: | Состояние памяти поле FState –  «Включена» (\_On). |
|  |  |
| ***Добавить*** |  |
| Вход: | (Е) – число объект типа T. |
| Предусловия: | Нет. |
| Процесс: | В поле FNumber объекта «память» (тип TMemory) записывается объект типа T, полученный в результате сложения числа (Е) и числа, хранящегося в памяти в поле  FNumber. |
| Выход: | Нет. |
| Постусловия: | Состояние памяти поле FState – |

|  |  |
| --- | --- |
|  | «Включена» (\_On). |
|  |  |
| ***Очистить*** |  |
| Вход: | Нет. |
| Предусловия: | Нет. |
| Процесс: | В поле числа (FNumber) объекта «память» (тип TMemory) записывается объект типа T со значением по умолчанию. Например, для простой дроби - 0/1. Память (поле FState) устанавливается в состояние «Выключена»  (\_Off). |
| Выход: | Нет. |
| Постусловия: | Состояние памяти поле FState –  «Выключена» (\_Off). |
|  |  |
| ***ЧитатьСостояниеПамяти*** |  |
| Вход: | Нет. |
| Предусловия: | Нет. |
| Процесс: | Копирует и возвращает значение поля FState «состояние памяти» объекта  «память» (тип TMemory) в формате строки. |
| Выход: | Значение поля «состояния памяти» (типа  String). |
| Постусловия: | Нет. |
|  |  |
| ***ЧитатьЧисло*** |  |
| Вход: | Нет. |
| Предусловия: | Нет. |
| Процесс: | Копирует и возвращает значение поля |

|  |  |
| --- | --- |
|  | «число» (FNumber) объекта «память» (тип  TMemory). |
| Выход: | Объект число (тип T). |
| Постусловия: | Нет. |

***end*** TCMemory

Спецификация типа данных «Процессор».

**ADT** TProc

# Данные

Процессор (тип TProc) выполняет двухоперандные операции TOprtn = (None, Add, Sub, Mul, Dvd) и однооперандные операции - функции TFunc = (Rev, Sqr) над значениями типа T. Левый операнд и результат операции хранится в поле Lop\_Res, правый - в поле Rop. Оба поля имеют тип T. Процессор может находиться в состояниях: «операция установлена» - поле Operation не равно None (значение типа TOprtn) или в состоянии «операция не установлена» - поле Operation = None. Значения типа TProc - изменяемые. Они изменяются операциями: «Сброс операции» (OprtnClear), «Выполнить операцию» (OprtnRun), «Вычислить функцию» (FuncRun), «Установить операцию» (OprtnSet), «Установить левый операнд» (Lop\_Res\_Set), «Установить правый операнд» (Rop\_Set), «Сброс калькулятора» (ReSet). На значениях типа T должны быть определены указанные выше операции и функции.

# Операции

|  |  |
| --- | --- |
| ***Конструктор*** |  |

|  |  |
| --- | --- |
| Начальные  значения: | Нет |
| Процесс: | Инициализирует поля объекта процессор типа TProc. Поля Lop\_Res, Rop инициализируются объектами (тип T) со значениями по умолчанию. Например, для простых дробей - 0/1. Процессор устанавливается в состояние: «операция не  установлена»: (Operation = None). |
|  |  |
| ***СбросПроцессора*** |  |
| Вход: | Нет. |
| Предусловия: | Нет. |
| Процесс: | Поля объекта процессор: Lop\_Res, Rop инициализируются объектами (тип T) со значениями по умолчанию. Например, для простых дробей - 0/1. Процессор устанавливается в состояние: «операция не  установлена»: (Operation = None). |
| Выход: | Нет. |
| Постусловия: | Состояние процессора – «операция  сброшена» (Operation = None). |
|  |  |
| ***СбросОперации*** |  |
| Вход: | Нет. |
| Предусловия: | Нет. |
| Процесс | Процессор устанавливается в состояние:  «операция не установлена»: (Operation = None). |

|  |  |
| --- | --- |
| Выход: | Нет. |
| Постусловия: | Состояние процессора – «операция  сброшена» (Operation = None). |
|  |  |
| ***ВыполнитьОперацию*** |  |
| Вход: | Нет. |
| Предусловия: | Нет. |
| Процесс | Вызывает выполнение текущей операции (записанной в поле Operation). Операция (Operation) выполняется над значениями, хранящимися в полях Rop и Lop\_Res. Результат сохраняется в поле Lop\_Res. Если Operation = None, никакие действия не выполняются. Состояние объекта не  изменяется. |
| Выход: | Нет. |
| Постусловия: | Состояние процессора не изменяется. |
|  |  |
| ***ВычислитьФункцию*** |  |
| Вход: | Вид функции (Func: TFunc). |
| Предусловия: | Нет. |
| Процесс | Вызывает выполнение текущей функции (Func). Функция (Func) выполняется над значением, хранящимся в поле Rop. Результат сохраняется в нём же. Состояние  объекта не изменяется. |
| Выход: | Нет. |
| Постусловия: | Состояние процессора не меняется. |
|  |  |

|  |  |
| --- | --- |
| ***ЧитатьЛевыйОперанд*** |  |
| Вход: | Нет. |
| Предусловия: | Нет. |
| Процесс | Создаёт и возвращает копию объекта,  который хранится в поле Lop\_Res. |
| Выход: | Объект типа T. |
| Постусловия: | Состояние процессора не изменяется. |
|  |  |
| ***ЗаписатьЛевыйОперанд*** |  |
| Вход: | Переменная Operand типа T. |
| Предусловия: | Нет. |
| Процесс | Создаёт копию объекта Operand и заносит  её в поле Lop\_Res. |
| Выход: | Нет. |
| Постусловия: | Состояние процессора не изменяется. |
|  |  |
| ***ЧитатьПравыйОперанд*** |  |
| Вход: | Нет. |
| Предусловия: | Нет. |
| Процесс | Создаёт и возвращает копию объекта,  который хранится в Rop. |
| Выход: | Объект типа T. |
| Постусловия: | Состояние процессора не меняется. |
|  |  |
| ***ЗаписатьПравыйОперанд*** |  |
| Вход: | Переменная Operand типа T. |
| Предусловия: | Нет. |
| Процесс | Создаёт копию объекта Operand и заносит  её в поле Rop. |

|  |  |
| --- | --- |
| Выход: | Нет. |
| Постусловия: | Состояние процессора не изменяется. |
|  |  |
| ***ЧитатьСостояние*** |  |
| Вход: | Нет. |
| Предусловия: | Нет. |
| Процесс | Копирует и возвращает значение поля  Operation. |
| Выход: | Значение поля Operation. |
| Постусловия: | Состояние процессора не изменяется. |
|  |  |
| ***ЗаписатьСостояние*** |  |
| Вход: | Переменная Oprtn типа TOprtn. |
| Предусловия: | Нет. |
| Процесс | Заносит значение Oprtn в поле Operation. |
| Выход: | Нет. |
| Постусловия: | Состояние процессора изменяется на Oprtn. |

Спецификация типа данных Управление Калькулятором

На Унифицированном языке моделирования UML (Unified Modeling Language) класс можно обозначить следующим образом:

|  |
| --- |
| **УправлениеКалькуляторомПростыхДробей (тип TCtrl)** |
| состояниеКалькулятора: TCtrlState |
| редактор: TEditor |
| процессор: TProc |
| память: TMemory |
| число: TFrac |
| выполнитьКомандуКалькулятора(a: Integer; var b, MState: String):  String |
| выполнитьКомандуРедактора(a: Integer): String |
| выполнитьОперацию(a: Integer): String |
| выполнитьФункцию(a: Integer): String |
| вычислитьВыражение(a: Integer): String |
| установитьНачальноеСостояниеКалькулятора(a: Integer): String |
| выполнитьКомандуюПамяти(a: Integer; var MState: String): String |
| читатьПисатьСостояниеКалькулятора: TCtrlState |
| выполнитьКомандуБуфераОбмена(a: Integer; var b: String): String |
| конструктор |
| деструктор |
| Обязанность:  управление выполнением команд калькулятора |

Спецификация типа данных Интерфейс

На Унифицированном языке моделирования UML (Unified Modeling Language) класс можно обозначить следующим образом:

|  |
| --- |
| **ИнтерфейсКалькулятораПростыхДробей** |
| строкаПростаяДробь: TStaticText |
| состояниеПамяти: TStaticText |
| кнопки ввода: TBitButton |
| FormCreate(Sender: TObject) |
| ButtonClick(Sender: TObject) |
| FormKeyPress(Sender: TObject; var Key: Char) |
| Методы для обработки команд меню |
| Обязанность:  Обеспечить пользователю возможность управления калькулятором через клавиатуру и командные кнопки для выполнения вычислений |

**Тестовые наборы данных для тестирования абстрактных типов данных, классов и приложения**

**Класс TFrac: Конструкторы**

|  |  |  |
| --- | --- | --- |
| **№** | **Исходные данные** | **Ожидаемый результат** |
| **Вход** | **Возвращаемое значение** |
| **1** | **Строка “1/2”** | **Числитель – 1**  **Знаменатель - 2** |
| **2** | **Строка “555/2”** | **Числитель – 555**  **Знаменатель - 2** |
| **3** | **Строка “1/555”** | **Числитель – 1**  **Знаменатель - 555** |
| **4** | **Строка “50/-25”** | **Числитель – -2**  **Знаменатель - 1** |
| **5** | **Строка “-1/555”** | **Числитель – -1**  **Знаменатель - 555** |
| **6** | **Строка “-50/-25”** | **Числитель – 2**  **Знаменатель - 1** |
| **7** | **Строка “1/0”** | **Исключение** |
| **8** | **Строка “0/1”** | **Числитель – 0**  **Знаменатель - 1** |

**Класс TFrac: Операции**

|  |  |  |
| --- | --- | --- |
| **№** | **Исходные данные** | **Ожидаемый результат** |
| **Вход** | **Возвращаемое значение** |
| **1** | **Frac = 1/2 + 3/4** | **Frac = 5/4** |
| **2** | **Frac = -1/2 + 3/4** | **Frac = 1/4** |
| **3** | **Frac = -5/9 + 5/9** | **Frac = 0/1** |
| **4** | **Frac = 1/2 +0/1** | **Frac = 1/2** |
| **5** | **Frac = 1/2 - 3/4** | **Frac = -1/4** |
| **6** | **Frac = -1/2 - 3/4** | **Frac = -5/4** |
| **7** | **Frac = -5/9 - 5/9** | **Frac = -10/9** |
| **8** | **Frac = 1/2 -0/1** | **Frac = 1/2** |
| **9** | **Frac = 1/2 \* 3/4** | **Frac = 3/8** |
| **10** | **Frac = -1/2 \* 3/4** | **Frac = -3/8** |
| **11** | **Frac = -5/9 \* 5/9** | **Frac = -25/81** |
| **12** | **Frac = 1/2 \* 0/1** | **Frac = 0/1** |
| **13** | **Frac = 1/2 / 3/4** | **Frac = 2/3** |
| **14** | **Frac = -1/2 / 3/4** | **Frac = -2/3** |
| **15** | **Frac = -5/9 / 5/9** | **Frac = 0/1** |
| **16** | **Frac = 1/2 / 0/1** | **Исключение** |

**Класс TFrac: Функции**

|  |  |  |
| --- | --- | --- |
| **№** | **Исходные данные** | **Ожидаемый результат** |
| **Вход** | **Возвращаемое значение** |
| **1** | **Frac = Reverse(1/2)** | **Frac = 2/1** |
| **2** | **Frac = Reverse(-1/2)** | **Frac = -2/1** |
| **3** | **Frac = Reverse(0/1)** | **Frac = 0/1** |
| **4** | **Frac = Reverse(3/9)** | **Frac = 3/1** |
| **5** | **Frac = Reverse(0/1)** | **Исключение** |
| **6** | **Frac = Square(1/2)** | **Frac = 1/4** |
| **7** | **Frac = Square(-1/2)** | **Frac = 1/4** |
| **8** | **Frac = Sqare(0/1)** | **Frac = 0/1** |
| **9** | **Frac = Square(3/9)** | **Frac = 1/9** |
| **10** | **Frac = Square(5/9)** | **Frac = 25/81** |

**Заключение**

В ходе выпонения расчетно-графического задания были сформированы практические навыки проектирования программ в технологии «абстрактных типов данных» и «объектно-ориентированного программирования» и построения диаграмм UML; реализации абстрактных типов данных с помошью классов C#, C++; использования библиотеки визуальных компонентов VCL для построения интерфейса; тестирования программ.

**Текст программы**

**TNumber.cs**

using System;

using System.Globalization;

namespace Calculator

{

public abstract class TANumber

{

public abstract TANumber Add(TANumber a);

public abstract TANumber Mul(TANumber a);

public abstract TANumber Div(TANumber a);

public abstract TANumber Sub(TANumber a);

public abstract bool IsZero();

public abstract object Square();

public abstract object Reverse();

public abstract void SetString(string str);

}

public sealed class TNumber : TANumber

{

public double Number

{

get;

}

public TNumber()

{

Number = 0;

}

public TNumber(double n)

{

Number = n;

}

public TNumber(string str)

{

if (double.TryParse(str, NumberStyles.Float, CultureInfo.InvariantCulture, out double NewNumber))

{

Number = NewNumber;

}

else

{

Number = 0;

}

}

public TNumber(TNumber num)

{

Number = num.Number;

}

public override TANumber Add(TANumber a)

{

return new TNumber(Number + (a as TNumber).Number);

}

public override TANumber Div(TANumber a)

{

if ((a as TNumber).IsZero())

{

throw new DivideByZeroException();

}

return new TNumber(Number / (a as TNumber).Number);

}

public override TANumber Sub(TANumber a)

{

return new TNumber(Number - (a as TNumber).Number);

}

public override TANumber Mul(TANumber a)

{

return new TNumber(Number \* (a as TNumber).Number);

}

public override bool IsZero()

{

return Number == 0;

}

public override object Reverse()

{

return new TNumber(1 / Number);

}

public override object Square()

{

return new TNumber(Number \* Number);

}

public override string ToString()

{

return Number.ToString("0.###", CultureInfo.InvariantCulture);

}

public override bool Equals(object obj)

{

var number = obj as TNumber;

return number != null && Number == number.Number;

}

public override void SetString(string str) { }

public static TNumber operator +(TNumber a, TNumber b)

{

return new TNumber(a.Number + b.Number);

}

public static TNumber operator \*(TNumber a, TNumber b)

{

return new TNumber(a.Number \* b.Number);

}

public static TNumber operator -(TNumber a, TNumber b)

{

return new TNumber(a.Number - b.Number);

}

public static TNumber operator /(TNumber a, TNumber b)

{

if (b.IsZero())

{

throw new DivideByZeroException();

}

return new TNumber(a.Number / b.Number);

}

public static bool operator ==(TNumber a, TNumber b)

{

return a.Number == b.Number;

}

public static bool operator ==(TNumber a, int b)

{

return a.Number == b;

}

public static bool operator !=(TNumber a, TNumber b)

{

return a.Number != b.Number;

}

public static bool operator !=(TNumber a, int b)

{

return a.Number != b;

}

public static bool operator >(TNumber a, TNumber b)

{

return (a.Number) > (b.Number);

}

public static bool operator <(TNumber a, TNumber b)

{

return (a.Number < b.Number);

}

public static bool operator >(TNumber a, int b)

{

return (a.Number) > (b);

}

public static bool operator <(TNumber a, int b)

{

return (a.Number < b);

}

public static TNumber operator -(TNumber a)

{

return new TNumber(-a.Number);

}

public static TNumber operator /(TNumber a, int b)

{

if (b == 0)

{

throw new DivideByZeroException();

}

return new TNumber(a.Number / b);

}

public static TNumber operator /(TNumber a, long b)

{

if (b == 0)

{

throw new DivideByZeroException();

}

return new TNumber(a.Number / b);

}

public static TNumber operator +(TNumber a, int b)

{

return new TNumber(a.Number + b);

}

public static TNumber operator -(TNumber a, int b)

{

return new TNumber(a.Number - b);

}

public static TNumber operator \*(TNumber a, int b)

{

return new TNumber(a.Number \* b);

}

}

}

**TFrac.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text.RegularExpressions;

namespace Calculator

{

public sealed class TFrac : TANumber

{

public TNumber Numerator;

public TNumber Denominator;

static void Swap<T>(ref T lhs, ref T rhs)

{

T temp;

temp = lhs;

lhs = rhs;

rhs = temp;

}

public static long GCD(long a, long b)

{

a = Math.Abs(a);

b = Math.Abs(b);

while (b > 0)

{

a %= b;

Swap(ref a, ref b);

}

return a;

}

public TFrac()

{

Numerator = new TNumber(0);

Denominator = new TNumber(1);

}

public TFrac(TNumber a, TNumber b)

{

try

{

if (a < 0 && b < 0)

{

a \*= -1;

b \*= -1;

}

else if (b < 0 && a > 0)

{

b \*= -1;

a \*= -1;

}

else if (a == 0 && b == 0 || b == 0 || a == 0 && b == 1)

{

Numerator = new TNumber(0);

Denominator = new TNumber(1);

return;

}

Numerator = new TNumber(a);

Denominator = new TNumber(b);

long gcdResult = GCD((long)a.Number, (long)b.Number);

if (gcdResult > 1)

{

Numerator /= gcdResult;

Denominator /= gcdResult;

}

}

catch

{

throw new OverflowException();

}

}

public TFrac(int a, int b)

{

if (a < 0 && b < 0)

{

a \*= -1;

b \*= -1;

}

else if (b < 0 && a > 0)

{

b \*= -1;

a \*= -1;

}

else if (a == 0 && b == 0 || b == 0 || a == 0 && b == 1)

{

Numerator = new TNumber(0);

Denominator = new TNumber(1);

return;

}

Numerator = new TNumber(a);

Denominator = new TNumber(b);

long gcdResult = GCD(a, b);

if (gcdResult > 1)

{

Numerator /= gcdResult;

Denominator /= gcdResult;

}

}

public TFrac(string fraction)

{

Regex FracRegex = new Regex(@"^-?(\d+)/(\d+)$");

Regex NumberRegex = new Regex(@"^-?\d+/?$");

if (FracRegex.IsMatch(fraction))

{

List<string> FracParts = fraction.Split('/').ToList();

Numerator = new TNumber(FracParts[0]);

Denominator = new TNumber(FracParts[1]);

if (Denominator.IsZero())

{

Numerator = new TNumber(0);

Denominator = new TNumber(1);

return;

}

long gcdResult = GCD((long)Numerator.Number, (long)Denominator.Number);

if (gcdResult > 1)

{

Numerator /= gcdResult;

Denominator /= gcdResult;

}

return;

}

else if (NumberRegex.IsMatch(fraction))

{

Numerator = new TNumber(fraction);

Denominator = new TNumber(1);

return;

}

else

{

Numerator = new TNumber(0);

Denominator = new TNumber(1);

return;

}

}

public TFrac(TFrac anotherFrac)

{

Numerator = anotherFrac.Numerator;

Denominator = anotherFrac.Denominator;

}

public static TFrac operator +(TFrac a, TFrac b)

{

return new TFrac(a.Numerator \* b.Denominator + a.Denominator \* b.Numerator, a.Denominator \* b.Denominator);

}

public static TFrac operator \*(TFrac a, TFrac b)

{

return new TFrac(a.Numerator \* b.Numerator, a.Denominator \* b.Denominator);

}

public static TFrac operator -(TFrac a, TFrac b)

{

return new TFrac(a.Numerator \* b.Denominator - a.Denominator \* b.Numerator, a.Denominator \* b.Denominator);

}

public static TFrac operator /(TFrac a, TFrac b)

{

if (b.IsZero())

{

throw new DivideByZeroException();

}

return new TFrac(a.Numerator \* b.Denominator, a.Denominator \* b.Numerator);

}

public static TFrac operator -(TFrac a)

{

return new TFrac(-a.Numerator, a.Denominator);

}

public static bool operator ==(TFrac a, TFrac b)

{

return a.Numerator == b.Numerator && a.Denominator == b.Denominator;

}

public static bool operator !=(TFrac a, TFrac b)

{

return a.Numerator != b.Numerator && a.Denominator != b.Denominator;

}

public static bool operator >(TFrac a, TFrac b)

{

return (a.Numerator / a.Denominator) > (b.Numerator / b.Denominator);

}

public static bool operator <(TFrac a, TFrac b)

{

return (a.Numerator / a.Denominator) < (b.Numerator / b.Denominator);

}

public override TANumber Add(TANumber a)

{

return new TFrac(Numerator \* (a as TFrac).Denominator + Denominator \* (a as TFrac).Numerator, Denominator \* (a as TFrac).Denominator);

}

public override TANumber Mul(TANumber a)

{

return new TFrac((a as TFrac).Numerator \* Numerator, (a as TFrac).Denominator \* Denominator);

}

public override TANumber Div(TANumber a)

{

return new TFrac((a as TFrac).Numerator \* Denominator, (a as TFrac).Denominator \* Numerator);

}

public override TANumber Sub(TANumber a)

{

return new TFrac((a as TFrac).Numerator \* Denominator - (a as TFrac).Denominator \* Numerator, (a as TFrac).Denominator \* Denominator);

}

public override object Square()

{

return new TFrac((TNumber)Numerator.Square(), (TNumber)Denominator.Square());

}

public override object Reverse()

{

return new TFrac(Denominator, Numerator);

}

public override bool IsZero()

{

return Numerator.IsZero();

}

public override void SetString(string str)

{

TFrac TempFrac = new TFrac(str);

Numerator = TempFrac.Numerator;

Denominator = TempFrac.Denominator;

}

public override string ToString()

{

return Numerator.ToString() + "/" + Denominator.ToString();

}

public override bool Equals(object obj)

{

var frac = obj as TFrac;

return frac != null &&Numerator == frac.Numerator && Denominator == frac.Denominator;

}

}

}

**TAEditor.cs**

using System;

namespace Calculator

{

public abstract class TAEditor

{

public abstract string Number

{

get;

set;

}

public enum Command

{

cZero, cOne, cTwo, cThree, cFour, cFive, cSix,

cSeven, cEight, cNine, cA, cB, cC, cD, cE, cF,

cSign, cSeparator, cBS, CE, cNumbSeparator, cNone

}

public abstract bool IsZero();

public abstract string ToogleSign();

public abstract string AddNumber(int num);

public abstract string AddZero();

public abstract string RemoveSymbol();

public abstract string Clear();

public abstract string Edit(Enum com);

public abstract string AddSeparator();

public abstract bool HaveSeparator();

}

}

**TFracEditor.cs**

using System;

using System.Linq;

namespace Calculator

{

public sealed class TFracEditor : TAEditor

{

const int BothSideLimit = 22;

const int LeftSideOnlyLimit = 14;

const string Separator = "/";

const string ZeroFraction = "0/";

private string number;

public override string Number

{

get => number;

set

{

number = new TFrac(value).ToString();

}

}

public TFracEditor()

{

number = "0";

}

public TFracEditor(int a, int b)

{

number = new TFrac(a, b).ToString();

}

public TFracEditor(string str)

{

number = new TFrac(str).ToString();

}

public override bool IsZero()

{

return number.StartsWith(ZeroFraction) || number == "0";

}

public override string ToogleSign()

{

if (number.ElementAt(0) == '-')

{

number = number.Remove(0, 1);

}

number = "-" + number;

return number;

}

public override string AddNumber(int a)

{

if ((!HaveSeparator() && number.Length > LeftSideOnlyLimit)

|| (number.Length > BothSideLimit)

|| (a < 0 || a > 9))

{

return number;

}

if (a == 0)

{

AddZero();

}

else if (number == "0" || number == "-0")

{

number = number.First() == '-' ? "-" + a.ToString() : a.ToString();

}

else

{

number += a.ToString();

}

return number;

}

public override bool Equals(object obj)

{

return obj is TFracEditor editor && number == editor.number;

}

public override string AddZero()

{

if ((HaveSeparator() && number.Last().ToString() == Separator)

|| (number == "0" || number == "0/"))

{

return number;

}

number += "0";

return number;

}

public override string RemoveSymbol()

{

if (number.Length == 1)

{

number = "0";

}

else if (number.Length == 2 && number.First() == '-')

{

number = "-0";

}

else

{

number = number.Remove(number.Length - 1);

}

return number;

}

public override string Clear()

{

number = "0";

return number;

}

public override string Edit(Enum com)

{

switch (com)

{

case Command.cZero:

AddZero();

break;

case Command.cOne:

AddNumber(1);

break;

case Command.cTwo:

AddNumber(2);

break;

case Command.cThree:

AddNumber(3);

break;

case Command.cFour:

AddNumber(4);

break;

case Command.cFive:

AddNumber(5);

break;

case Command.cSix:

AddNumber(6);

break;

case Command.cSeven:

AddNumber(7);

break;

case Command.cEight:

AddNumber(8);

break;

case Command.cNine:

AddNumber(9);

break;

case Command.cSign:

ToogleSign();

break;

case Command.cSeparator:

AddSeparator();

break;

case Command.cBS:

RemoveSymbol();

break;

case Command.CE:

Clear();

break;

default:

break;

}

return Number;

}

public override string AddSeparator()

{

if (!number.Contains(Separator))

{

number += Separator;

}

return number;

}

public override bool HaveSeparator()

{

return number.Contains(Separator);

}

public override string ToString()

{

return Number;

}

}

}

**TMemory.cs**

namespace Calculator

{

public sealed class TMemory<T> where T : TANumber, new()

{

public enum NumStates

{

OFF, ON

}

public enum Commands

{

Store, Add, Clear, Copy

}

T fNumber;

NumStates fState;

public T FNumber

{

get

{

fState = NumStates.ON; return fNumber;

}

set

{

fNumber = value;

fState = NumStates.ON;

}

}

public NumStates FState

{

get => fState;

set => fState = value;

}

public TMemory()

{

FNumber = new T();

FState = NumStates.OFF;

}

public TMemory(T number)

{

FNumber = number;

FState = NumStates.OFF;

}

public T Add(T number)

{

FState = NumStates.ON;

dynamic a = fNumber;

dynamic b = number;

fNumber = (T)(a + b);

return fNumber;

}

public void Clear()

{

fNumber = new T();

FState = NumStates.OFF;

}

public (T, NumStates) Edit(Commands command, T newNumber)

{

switch (command)

{

case Commands.Store:

FState = NumStates.ON;

fNumber = newNumber;

break;

case Commands.Add:

FState = NumStates.ON;

dynamic a = fNumber;

dynamic b = newNumber;

fNumber = (T)(a + b);

break;

case Commands.Clear:

Clear();

break;

}

return (fNumber, fState);

}

}

}

**TProc.cs**

namespace Calculator

{

public sealed class TProc<T> where T : TANumber, new()

{

public enum Oper

{

None, Add, Sub, Mul, Div

}

public enum Func

{

Rev, Sqr

}

Oper operation;

T lop\_Res;

T rop;

public T Lop\_Res

{

get => lop\_Res;

set => lop\_Res = value;

}

public T Rop

{

get => rop;

set => rop = value;

}

public Oper Operation

{

get => operation;

set => operation = value;

}

public TProc()

{

operation = Oper.None;

Lop\_Res = new T();

Rop = new T();

}

public TProc(T leftObj, T rightObj)

{

operation = Oper.None;

Lop\_Res = leftObj;

Rop = rightObj;

}

public void Reset()

{

operation = Oper.None;

T newObj = new T();

Lop\_Res = Rop = newObj;

}

public void DoOper()

{

try

{

dynamic a = Lop\_Res;

dynamic b = Rop;

switch (operation)

{

case Oper.Add:

Lop\_Res = (T)(a + b);

break;

case Oper.Sub:

Lop\_Res = (T)(a - b);

break;

case Oper.Mul:

Lop\_Res = (T)(a \* b);

break;

case Oper.Div:

Lop\_Res = (T)(a / b);

break;

default:

break;

}

}

catch

{

throw new System.OverflowException();

}

}

public void DoFunc(Func func)

{

dynamic a = Rop;

switch (func)

{

case Func.Rev:

a = a.Reverse();

Rop = (T)a;

break;

case Func.Sqr:

a = a.Square();

Rop = (T)a;

break;

default:

break;

}

}

}

}

**TCtrl.cs**

namespace Calculator

{

public sealed class TCtrl<T, Editor>

where T : TANumber, new()

where Editor : TAEditor, new()

{

public enum TCtrlState

{

cStart, cEditing, FunDone, cOperDone, cExpDone, cOpChange, cError

}

Editor edit;

TProc<T> proc;

TMemory<T> memory;

TCtrlState curState;

public TCtrlState CurState

{

get => curState;

set => curState = value;

}

public TProc<T> Proc

{

get => proc;

set => proc = value;

}

public TMemory<T> Memory

{

get => memory;

set => memory = value;

}

public Editor Edit

{

get => edit;

set => edit = value;

}

public TCtrl()

{

Edit = new Editor();

Proc = new TProc<T>();

Memory = new TMemory<T>();

curState = TCtrlState.cStart;

}

public string ExecCommandEditor(TAEditor.Command command)

{

string ToReturn;

if (CurState == TCtrlState.cExpDone)

{

Proc.Reset();

CurState = TCtrlState.cStart;

}

if (CurState != TCtrlState.cStart)

{

CurState = TCtrlState.cEditing;

}

ToReturn = Edit.Edit(command);

T TempObj = new T();

if (TempObj is TPNumber)

{

dynamic a = TempObj;

dynamic b = Edit;

a.Notation = new TNumber(b.Notation);

TempObj = a;

}

TempObj.SetString(ToReturn);

Proc.Rop = TempObj;

return ToReturn;

}

public string ExecOperation(TProc<T>.Oper oper)

{

if (oper == TProc<T>.Oper.None)

{

return Edit.Number;

}

string ToReturn;

try

{

switch (CurState)

{

case TCtrlState.cStart:

Proc.Lop\_Res = Proc.Rop;

Proc.Operation = oper;

CurState = TCtrlState.cOperDone;

Edit.Clear();

break;

case TCtrlState.cEditing:

Proc.DoOper();

Proc.Operation = oper;

Edit.Clear();

CurState = TCtrlState.cOperDone;

break;

case TCtrlState.FunDone:

if (Proc.Operation == TProc<T>.Oper.None)

{

Proc.Lop\_Res = Proc.Rop;

}

else

{

Proc.DoOper();

}

Proc.Operation = oper;

Edit.Clear();

CurState = TCtrlState.cOpChange;

break;

case TCtrlState.cOperDone:

CurState = TCtrlState.cOpChange;

Edit.Clear();

break;

case TCtrlState.cExpDone:

Proc.Operation = oper;

Proc.Rop = Proc.Lop\_Res;

CurState = TCtrlState.cOpChange;

Edit.Clear();

break;

case TCtrlState.cError:

Proc.Reset();

return "ERR";

case TCtrlState.cOpChange:

Proc.Operation = oper;

Edit.Clear();

break;

default:

break;

}

ToReturn = Proc.Lop\_Res.ToString();

}

catch

{

Reset();

return "ERROR";

}

return ToReturn;

}

public string ExecFunction(TProc<T>.Func func)

{

string ToReturn;

try

{

if (CurState == TCtrlState.cExpDone)

{

Proc.Rop = Proc.Lop\_Res;

Proc.Operation = TProc<T>.Oper.None;

}

Proc.DoFunc(func);

CurState = TCtrlState.FunDone;

ToReturn = Proc.Rop.ToString();

}

catch

{

Reset();

return "ERROR";

}

return ToReturn;

}

public string Calculate()

{

string ToReturn;

try

{

if (CurState == TCtrlState.cStart)

{

Proc.Lop\_Res = Proc.Rop;

}

Proc.DoOper();

CurState = TCtrlState.cExpDone;

ToReturn = Proc.Lop\_Res.ToString();

}

catch

{

Reset();

return "ERROR";

}

return ToReturn;

}

public string Reset()

{

Edit.Clear();

Proc.Reset();

Memory.Clear();

curState = TCtrlState.cStart;

return Edit.ToString();

}

public (T, TMemory<T>.NumStates) ExecCommandMemory(TMemory<T>.Commands command, string str)

{

T TempObj = new T();

TempObj.SetString(str);

(T, TMemory<T>.NumStates) obj = (null, TMemory<T>.NumStates.OFF);

try

{

obj = Memory.Edit(command, TempObj);

}

catch

{

Reset();

return obj;

}

if (command == TMemory<T>.Commands.Copy)

{

Edit.Number = obj.Item1.ToString();

Proc.Rop = obj.Item1;

}

return obj;

}

}

}

**Form1.cs (интерфейс)**

using System;

using System.Linq;

using System.Windows.Forms;

namespace Calculator

{

public partial class Form1 : Form

{

readonly TCtrl<TFrac, TFracEditor> frac\_ctrl;

readonly TCtrl<TPNumber, TPNumberEditor> pNumber\_ctrl;

readonly TCtrl<TComplex, TComplexEditor> complex\_ctrl;

const string OPERATIONS = "+-/\*";

const string TAG\_FRAC = "FRAC\_";

const string TAG\_COMPLEX = "COMPLEX\_";

const string TAG\_PNUMBER = "PNUMBER\_";

bool frac\_mode = true;

bool pNumber\_mode = true;

bool complex\_mode = true;

enum Complex\_func

{

Pwr, Root, Abs, Dgr, Rad

}

private string Number\_to\_state(string tag, string str)

{

if ("ERROR" == str)

{

return str;

}

string return\_str = str;

switch (tag)

{

case TAG\_PNUMBER:

break;

case TAG\_FRAC:

if (true == frac\_mode)

{

return\_str = str;

}

else if (new TFrac(str).Denominator == 1)

{

return\_str = new TFrac(str).Numerator.ToString();

}

break;

case TAG\_COMPLEX:

if (true == complex\_mode)

{

return\_str = str;

}

else if (new TComplex(str).Imaginary == 0)

{

return\_str = new TComplex(str).Real.ToString();

}

break;

}

return return\_str;

}

private static TAEditor.Command Char\_to\_edit\_command(char ch)

{

TAEditor.Command command = TAEditor.Command.cNone;

switch (ch)

{

case '0':

command = TAEditor.Command.cZero;

break;

case '1':

command = TAEditor.Command.cOne;

break;

case '2':

command = TAEditor.Command.cTwo;

break;

case '3':

command = TAEditor.Command.cThree;

break;

case '4':

command = TAEditor.Command.cFour;

break;

case '5':

command = TAEditor.Command.cFive;

break;

case '6':

command = TAEditor.Command.cSix;

break;

case '7':

command = TAEditor.Command.cSeven;

break;

case '8':

command = TAEditor.Command.cEight;

break;

case '9':

command = TAEditor.Command.cNine;

break;

case 'A':

command = TAEditor.Command.cA;

break;

case 'B':

command = TAEditor.Command.cB;

break;

case 'C':

command = TAEditor.Command.cC;

break;

case 'D':

command = TAEditor.Command.cD;

break;

case 'E':

command = TAEditor.Command.cE;

break;

case 'F':

command = TAEditor.Command.cF;

break;

case '.':

case '/':

command = TAEditor.Command.cSeparator;

break;

case '-':

command = TAEditor.Command.cSign;

break;

}

return command;

}

private static TProc<T>.Oper Char\_to\_operation\_command<T>(char ch) where T : TANumber, new()

{

TProc<T>.Oper command = TProc<T>.Oper.None;

switch (ch)

{

case '+':

command = TProc<T>.Oper.Add;

break;

case '-':

command = TProc<T>.Oper.Sub;

break;

case '\*':

command = TProc<T>.Oper.Mul;

break;

case '/':

command = TProc<T>.Oper.Div;

break;

}

return command;

}

private static TAEditor.Command Keycode\_to\_edit\_command(Keys ch)

{

TAEditor.Command command = TAEditor.Command.cNone;

switch (ch)

{

case Keys.Back:

command = TAEditor.Command.cBS;

break;

case Keys.Delete:

case Keys.Escape:

command = TAEditor.Command.CE;

break;

}

return command;

}

public Form1()

{

frac\_ctrl = new TCtrl<TFrac, TFracEditor>();

pNumber\_ctrl = new TCtrl<TPNumber, TPNumberEditor>();

complex\_ctrl = new TCtrl<TComplex, TComplexEditor>();

InitializeComponent();

Size = new System.Drawing.Size(355, 433);

}

private void Button\_Number\_Edit(object sender, EventArgs e)

{

TAEditor.Command parsed\_command;

Button button = (Button)sender;

string tag = button.Tag.ToString();

string part\_tag = tag.Substring(0, tag.IndexOf('\_')) + "\_";

switch (part\_tag)

{

case TAG\_PNUMBER:

pNumber\_ctrl.Edit.Notation = new TNumber(trackBar\_PNumber.Value);

Enum.TryParse(tag.Replace(TAG\_PNUMBER, string.Empty), out parsed\_command);

tB\_PNumber.Text = pNumber\_ctrl.ExecCommandEditor(parsed\_command);

break;

case TAG\_FRAC:

Enum.TryParse(tag.Replace(TAG\_FRAC, string.Empty), out parsed\_command);

tB\_Frac.Text = frac\_ctrl.ExecCommandEditor(parsed\_command);

break;

case TAG\_COMPLEX:

Enum.TryParse(tag.Replace(TAG\_COMPLEX, string.Empty), out parsed\_command);

tB\_Complex.Text = complex\_ctrl.ExecCommandEditor(parsed\_command);

break;

}

}

private void Button\_Number\_Operation(object sender, EventArgs e)

{

TProc<TFrac>.Oper parsed\_frac\_oper;

TProc<TPNumber>.Oper parsed\_pNumber\_oper;

TProc<TComplex>.Oper parsed\_complex\_oper;

Button button = (Button)sender;

string tag = button.Tag.ToString();

string part\_tag = tag.Substring(0, tag.IndexOf('\_')) + "\_";

switch (part\_tag)

{

case TAG\_PNUMBER:

Enum.TryParse(tag.Replace(TAG\_PNUMBER, string.Empty), out parsed\_pNumber\_oper);

tB\_PNumber.Text = pNumber\_ctrl.ExecOperation(parsed\_pNumber\_oper);

break;

case TAG\_FRAC:

Enum.TryParse(tag.Replace(TAG\_FRAC, string.Empty), out parsed\_frac\_oper);

tB\_Frac.Text = Number\_to\_state(TAG\_FRAC, frac\_ctrl.ExecOperation(parsed\_frac\_oper));

break;

case TAG\_COMPLEX:

Enum.TryParse(tag.Replace(TAG\_COMPLEX, string.Empty), out parsed\_complex\_oper);

tB\_Complex.Text = Number\_to\_state(TAG\_COMPLEX, complex\_ctrl.ExecOperation(parsed\_complex\_oper));

break;

}

}

private void Button\_Number\_Function(object sender, EventArgs e)

{

TProc<TFrac>.Func parsed\_frac\_func;

TProc<TPNumber>.Func parsed\_pNumber\_func;

TProc<TComplex>.Func parsed\_complex\_func;

Button button = (Button)sender;

string tag = button.Tag.ToString();

string part\_tag = tag.Substring(0, tag.IndexOf('\_')) + "\_";

switch (part\_tag)

{

case TAG\_PNUMBER:

Enum.TryParse(tag.Replace(TAG\_PNUMBER, string.Empty), out parsed\_pNumber\_func);

tB\_PNumber.Text = pNumber\_ctrl.ExecFunction(parsed\_pNumber\_func);

break;

case TAG\_FRAC:

Enum.TryParse(tag.Replace(TAG\_FRAC, string.Empty), out parsed\_frac\_func);

tB\_Frac.Text = Number\_to\_state(TAG\_FRAC, frac\_ctrl.ExecFunction(parsed\_frac\_func));

break;

case TAG\_COMPLEX:

Enum.TryParse(tag.Replace(TAG\_COMPLEX, string.Empty), out parsed\_complex\_func);

tB\_Complex.Text = Number\_to\_state(TAG\_COMPLEX, complex\_ctrl.ExecFunction(parsed\_complex\_func));

break;

}

}

private void Button\_Reset(object sender, EventArgs e)

{

Button button = (Button)sender;

string tag = button.Tag.ToString();

string part\_tag = tag.Substring(0, tag.IndexOf('\_')) + "\_";

switch (part\_tag)

{

case TAG\_PNUMBER:

tB\_PNumber.Text = pNumber\_ctrl.Reset();

label\_PNumber\_Memory.Text = string.Empty;

break;

case TAG\_FRAC:

tB\_Frac.Text = frac\_ctrl.Reset();

label\_Frac\_Memory.Text = string.Empty;

break;

case TAG\_COMPLEX:

tB\_Complex.Text = complex\_ctrl.Reset();

tB\_Complex\_SpecialOut.Text = label\_Complex\_Memory.Text = string.Empty;

break;

}

}

private void Button\_FinishEval(object sender, EventArgs e)

{

Button button = (Button)sender;

string tag = button.Tag.ToString();

string part\_tag = tag.Substring(0, tag.IndexOf('\_')) + "\_";

switch(part\_tag)

{

case TAG\_PNUMBER:

tB\_PNumber.Text = pNumber\_ctrl.Calculate();

break;

case TAG\_FRAC:

tB\_Frac.Text = Number\_to\_state(TAG\_FRAC, frac\_ctrl.Calculate()); ;

break;

case TAG\_COMPLEX:

tB\_Complex.Text = Number\_to\_state(TAG\_COMPLEX, complex\_ctrl.Calculate());

break;

}

}

private void Button\_Memory(object sender, EventArgs e)

{

TMemory<TPNumber>.Commands parsed\_pNumber\_command;

TMemory<TFrac>.Commands parsed\_frac\_command;

TMemory<TComplex>.Commands parsed\_complex\_command;

dynamic exec;

Button button = (Button)sender;

string tag = button.Tag.ToString();

string part\_tag = tag.Substring(0, tag.IndexOf('\_')) + "\_";

switch(part\_tag)

{

case TAG\_PNUMBER:

Enum.TryParse(tag.Replace(TAG\_PNUMBER, string.Empty), out parsed\_pNumber\_command);

exec = pNumber\_ctrl.ExecCommandMemory(parsed\_pNumber\_command, tB\_PNumber.Text);

if (TMemory<TPNumber>.Commands.Copy == parsed\_pNumber\_command)

{

tB\_PNumber.Text = exec.Item1.ToString();

}

label\_PNumber\_Memory.Text = exec.Item2 == TMemory<TPNumber>.NumStates.ON ? "M" : string.Empty;

break;

case TAG\_FRAC:

Enum.TryParse(tag.Replace(TAG\_FRAC, string.Empty), out parsed\_frac\_command);

exec = frac\_ctrl.ExecCommandMemory(parsed\_frac\_command, tB\_Frac.Text);

if (TMemory<TFrac>.Commands.Copy == parsed\_frac\_command)

{

tB\_Frac.Text = exec.Item1.ToString();

}

label\_Frac\_Memory.Text = exec.Item2 == TMemory<TFrac>.NumStates.ON ? "M" : string.Empty;

break;

case TAG\_COMPLEX:

Enum.TryParse(tag.Replace(TAG\_COMPLEX, string.Empty), out parsed\_complex\_command);

exec = complex\_ctrl.ExecCommandMemory(parsed\_complex\_command, tB\_Complex.Text);

if (TMemory<TComplex>.Commands.Copy == parsed\_complex\_command)

{

tB\_Complex.Text = exec.Item1.ToString();

}

label\_Complex\_Memory.Text = exec.Item2 == TMemory<TComplex>.NumStates.ON ? "M" : string.Empty;

break;

}

}

private void СправкаToolStripMenuItem\_Click(object sender, EventArgs e)

{

MessageBox.Show("Калькулятор чисел", "Калькулятор", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

private void TrackBar\_PNumber\_ValueChanged(object sender, EventArgs e)

{

label\_PNumber\_P.Text = trackBar\_PNumber.Value.ToString();

pNumber\_ctrl.Edit.Notation = new TNumber(trackBar\_PNumber.Value);

tB\_PNumber.Text = pNumber\_ctrl.Reset();

label\_PNumber\_Memory.Text = string.Empty;

string AllowedEndings = "0123456789ABCDEF";

foreach (Control i in tabPage\_PNumber.Controls.OfType<Button>())

{

if (AllowedEndings.Contains(i.Name.ToString().Last()) && i.Name.ToString().Substring(i.Name.ToString().Length - 2, 1) == "\_")

{

int j = AllowedEndings.IndexOf(i.Name.ToString().Last());

if (j < trackBar\_PNumber.Value)

{

i.Enabled = true;

}

if ((j >= trackBar\_PNumber.Value) && (j <= 15))

{

i.Enabled = false;

}

}

}

pNumber\_ctrl.Proc.Lop\_Res.Notation = new TNumber(trackBar\_PNumber.Value);

pNumber\_ctrl.Proc.Rop.Notation = new TNumber(trackBar\_PNumber.Value);

}

private void Button\_Complex\_Special(object sender, EventArgs e)

{

try

{

Button button = (Button)sender;

Enum.TryParse(button.Tag.ToString().Replace(TAG\_COMPLEX, string.Empty), out Complex\_func ParsedEnum);

TComplex number = new TComplex(tB\_Complex.Text);

switch (ParsedEnum)

{

case Complex\_func.Pwr:

int PwrN = Convert.ToInt32(nUD\_Complex\_Pwr.Value);

tB\_Complex\_SpecialOut.Text = number.Pwr(PwrN).ToString();

break;

case Complex\_func.Root:

int RootN = Convert.ToInt32(nUD\_Complex\_Root\_N.Value);

int Rooti = Convert.ToInt32(nUD\_Complex\_Root\_i.Value);

tB\_Complex\_SpecialOut.Text = number.Root(RootN, Rooti).ToString();

break;

case Complex\_func.Abs:

tB\_Complex\_SpecialOut.Text = number.Abs().ToString();

break;

case Complex\_func.Dgr:

tB\_Complex\_SpecialOut.Text = number.GetDegree().ToString();

break;

case Complex\_func.Rad:

tB\_Complex\_SpecialOut.Text = number.GetRad().ToString();

break;

}

}

catch

{

tB\_Complex\_SpecialOut.Text = "ERROR";

}

}

private void Button\_Complex\_ShowAllRoots(object sender, EventArgs e)

{

ShowRoots RootsForm = new ShowRoots();

int RootN = Convert.ToInt32(nUD\_Complex\_Root\_N.Value);

TComplex number = new TComplex(tB\_Complex.Text);

for (int i = 0; i < RootN; ++i)

{

RootsForm.richTB\_Roots.Text += "Root " + i.ToString() + ": " + number.Root(RootN, i).ToString() + Environment.NewLine;

}

RootsForm.ShowDialog();

}

private void NUD\_Complex\_Root\_N\_ValueChanged(object sender, EventArgs e)

{

nUD\_Complex\_Root\_i.Maximum = nUD\_Complex\_Root\_N.Value;

}

private void TabControl\_SelectedIndexChanged(object sender, EventArgs e)

{

switch (tabControl.SelectedIndex)

{

case 0:

Size = new System.Drawing.Size(355, 433);

break;

case 1:

Size = new System.Drawing.Size(310, 382);

break;

case 2:

Size = new System.Drawing.Size(445, 418);

break;

default:

break;

}

}

private void Form1\_KeyPress(object sender, KeyPressEventArgs e)

{

switch (tabControl.SelectedIndex)

{

case 0:

{

if ((e.KeyChar >= '0' && e.KeyChar <= '9')

|| (e.KeyChar >= 'A' && e.KeyChar <= 'F')

|| (e.KeyChar == '.' && pNumber\_mode))

{

tB\_PNumber.Text = pNumber\_ctrl.ExecCommandEditor(Char\_to\_edit\_command(e.KeyChar));

}

else if (OPERATIONS.Contains(e.KeyChar))

{

tB\_PNumber.Text = Number\_to\_state(TAG\_PNUMBER, pNumber\_ctrl.ExecOperation(Char\_to\_operation\_command<TPNumber>(e.KeyChar)));

}

break;

}

case 1:

{

if ((e.KeyChar >= '0' && e.KeyChar <= '9') || e.KeyChar == '.')

{

tB\_Frac.Text = frac\_ctrl.ExecCommandEditor(Char\_to\_edit\_command(e.KeyChar));

}

else if (OPERATIONS.Contains(e.KeyChar))

{

tB\_Frac.Text = Number\_to\_state(TAG\_FRAC, frac\_ctrl.ExecOperation(Char\_to\_operation\_command<TFrac>(e.KeyChar)));

}

break;

}

case 2:

{

if ((e.KeyChar >= '0' && e.KeyChar <= '9') || e.KeyChar == '.')

{

tB\_Complex.Text = complex\_ctrl.ExecCommandEditor(Char\_to\_edit\_command(e.KeyChar));

}

else if (OPERATIONS.Contains(e.KeyChar))

{

tB\_Complex.Text = Number\_to\_state(TAG\_COMPLEX, complex\_ctrl.ExecOperation(Char\_to\_operation\_command<TComplex>(e.KeyChar)));

}

break;

}

default:

break;

}

}

private void Form1\_KeyDown(object sender, KeyEventArgs e)

{

switch (tabControl.SelectedIndex)

{

case 0:

{

if (e.KeyCode == Keys.Enter)

{

b\_PNumber\_Eval.PerformClick();

}

else

{

TAEditor.Command comm = Keycode\_to\_edit\_command(e.KeyCode);

if (comm != TAEditor.Command.cNone)

{

tB\_PNumber.Text = pNumber\_ctrl.ExecCommandEditor(comm);

}

}

break;

}

case 1:

{

if (e.KeyCode == Keys.Enter)

{

b\_Frac\_Eval.PerformClick();

}

else

{

TAEditor.Command comm = Keycode\_to\_edit\_command(e.KeyCode);

if (comm != TAEditor.Command.cNone)

{

tB\_Frac.Text = pNumber\_ctrl.ExecCommandEditor(comm);

}

}

break;

}

case 2:

{

if (e.KeyCode == Keys.Enter)

{

b\_Complex\_Eval.PerformClick();

}

else

{

TAEditor.Command comm = Keycode\_to\_edit\_command(e.KeyCode);

if (comm != TAEditor.Command.cNone)

{

tB\_Complex.Text = pNumber\_ctrl.ExecCommandEditor(comm);

}

}

break;

}

default:

break;

}

}

private void ДействительныеPNumberTSMI\_Click(object sender, EventArgs e)

{

целыеPNumberTSMI.Checked = false;

действительныеPNumberTSMI.Checked = true;

pNumber\_mode = true;

b\_PNumber\_Separator.Enabled = true;

b\_PNumber\_Clear.PerformClick();

}

private void ЦелыеPNumberTSMI\_Click(object sender, EventArgs e)

{

целыеPNumberTSMI.Checked = true;

действительныеPNumberTSMI.Checked = false;

pNumber\_mode = false;

b\_PNumber\_Separator.Enabled = false;

b\_PNumber\_Clear.PerformClick();

}

private void ДробьFracTSMI\_Click(object sender, EventArgs e)

{

дробьFracTSMI.Checked = true;

числоFracTSMI.Checked = false;

frac\_mode = true;

}

private void ЧислоFracTSMI\_Click(object sender, EventArgs e)

{

дробьFracTSMI.Checked = false;

числоFracTSMI.Checked = true;

frac\_mode = false;

}

private void КомплексноеComplexTSMI\_Click(object sender, EventArgs e)

{

комплексноеComplexTSMI.Checked = true;

действительноеComplexTSMI.Checked = false;

complex\_mode = true;

}

private void ДействительноеComplexTSMI\_Click(object sender, EventArgs e)

{

комплексноеComplexTSMI.Checked = false;

действительноеComplexTSMI.Checked = true;

complex\_mode = false;

}

private void КопироватьToolStripMenuItem\_Click(object sender, EventArgs e)

{

switch(tabControl.SelectedIndex)

{

case 0:

Clipboard.SetText(tB\_PNumber.Text);

break;

case 1:

Clipboard.SetText(tB\_Frac.Text);

break;

case 2:

Clipboard.SetText(tB\_Complex.Text);

break;

}

}

private void ВставитьToolStripMenuItem\_Click(object sender, EventArgs e)

{

string text = Clipboard.GetText();

switch (tabControl.SelectedIndex)

{

case 0:

tB\_PNumber.Text = "";

for (int i = 0; i < text.Length; i++)

{

tB\_PNumber.Text = pNumber\_ctrl.ExecCommandEditor(Char\_to\_edit\_command(text[i]));

}

break;

case 1:

tB\_Frac.Text = "";

for (int i = 0; i < text.Length; i++)

{

tB\_Frac.Text = frac\_ctrl.ExecCommandEditor(Char\_to\_edit\_command(text[i]));

}

break;

case 2:

tB\_Complex.Text = "";

for (int i = 0; i < text.Length; i++)

{

tB\_Complex.Text = complex\_ctrl.ExecCommandEditor(Char\_to\_edit\_command(text[i]));

}

break;

}

}

}

}