Оглавление

Приложение 1. Практические задания для закрепления. Приложение Конвертор p1_p2.

Цель: Объектно-ориентированный анализ, проектирование и реализация приложения «Конвертор p1_p2» под Windows для преобразования действительных чисел представленных в системе счисления с основанием p1 в действительные числа представленные в системе счисления с основанием p2. В процессе выполнения работы студенты изучают: отношения между классами: ассоциация, агрегация, зависимость, их реализацию средствами языка программирования высокого уровня; этапы разработки приложений в технологии ООП; элементы технологии визуального программирования; диаграммы языка UML для документирования разработки.

Функциональные требования к приложению.

Интерфейс приложения может выглядеть так:

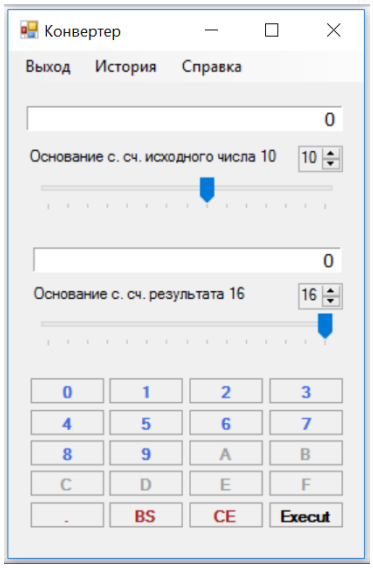
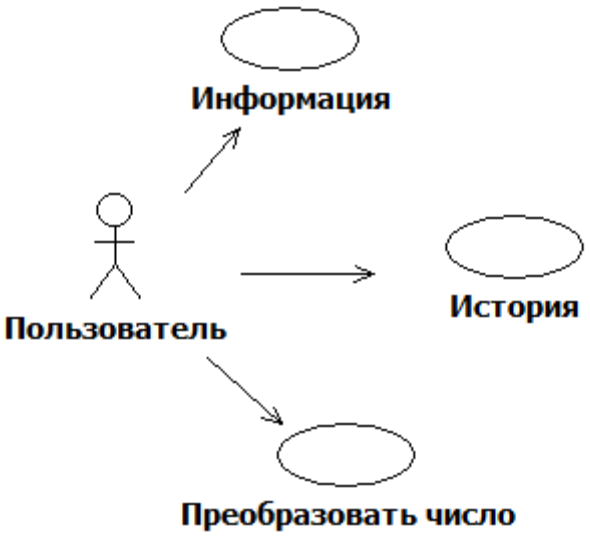


рис. 17. Главная форма приложения

Приложение должно обеспечивать пользователю: преобразование действительного числа представленного в системе счисления с основанием p1 в число, представленное в системе счисления с основанием p2; основания систем счисления p1, p2 для исходного числа и результата преобразования

выбираются пользователем из диапазона от 2..16; возможность ввода и редактирования действительного числа представленного в системе счисления с основанием p2 с помощью командных кнопок и мыши, а также с помощью клавиатуры; контекстную помощь по элементам интерфейса и справку о назначении приложения; просмотр истории сеанса (журнала) работы пользователя с приложением – исходные данные, результат преобразования и основания систем счисления, в которых они представлены; дополнительные повышенные требования: автоматический расчёт необходимой точности представления результата.

Функциональные требования представлены диаграммой прецедентов (use-case диаграммой) расположенной ниже.



Сценарий для прецедента «Преобразовать число»

Основной поток событий

1) Пользователь выбирает основание системы счисления p1 исходного числа.
2) Пользователь выбирает основание системы счисления p2 результата.
3) Пользователь вводит действительное число, представленное в системе счисления с основанием p1.
4) Пользователь вводит команду «Преобразовать».

5) Система выводит введённое пользователем число, представленное в системе счисления с основанием p2.
6) Система сохраняет исходные данные и результат преобразования в Историю.

Альтернативный поток событий 1. Введённое пользователем число выходит за границы допустимого диапазона.

3.1. Пользователь получает окно с сообщением.

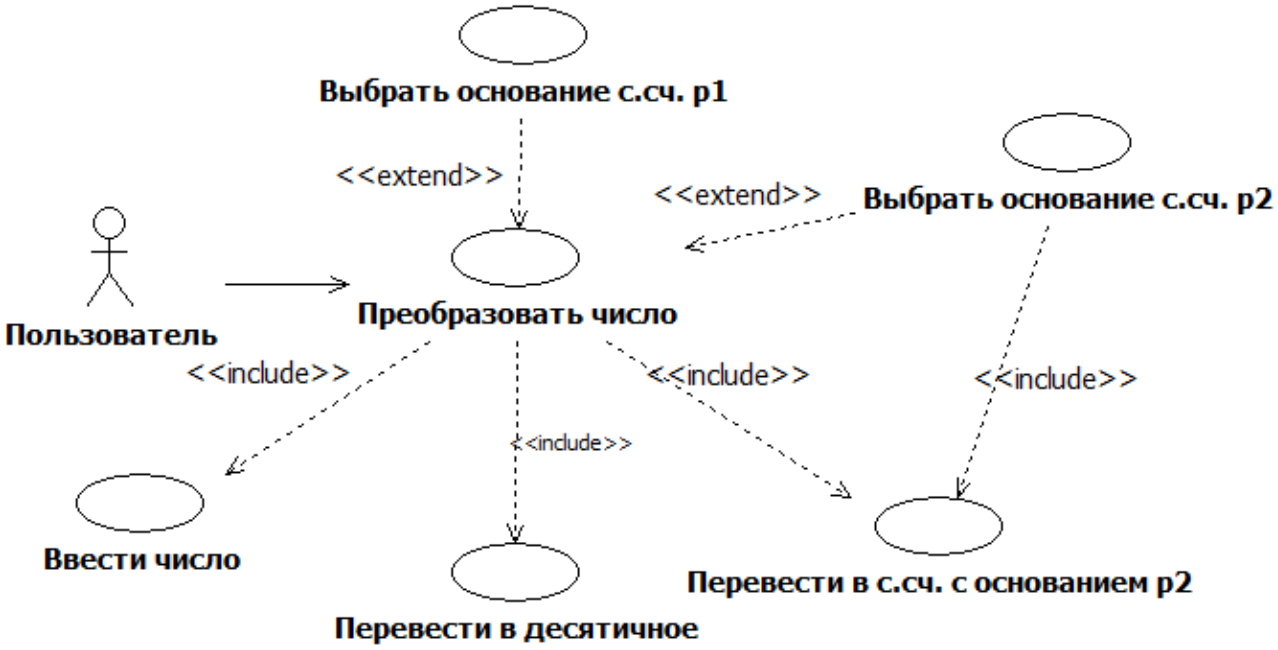
3.2. Приложение переходит в режим Ввод и редактирование.

Альтернативный поток событий 2. Количество разрядов в результате превышает размер поля вывода визуального компонента.

4.1. Пользователь получает окно с сообщением.

4.2. Приложение переходит в режим Ввод и редактирование.

Можно осуществить декомпозицию прецедента Преобразовать, в результате мы получим для него следующую диаграмму:



Сценарий для прецедента «Преобразовать»

Предусловие

Завершён ввод и редактирования исходного числа.

Основной поток событий

1) Пользователь вводит команду «Преобразовать».
2) Система выводит введённое пользователем число, представленное в системе счисления с основанием p2.
3) Система сохраняет исходные данные и результат преобразования в Историю.

Альтернативный поток событий 1. Введённое пользователем число выходит за границы допустимого диапазона.

1.1. Пользователь получает окно с сообщением.

1.2. Приложение переходит в режим Ввод и редактирование.

Альтернативный поток событий 2. Количество разрядов в результате превышает размер поля вывода визуального компонента.

1.1. Пользователь получает окно с сообщением.

1.2. Приложение переходит в режим Ввод и редактирование.

Сценарий для прецедента «Выбрать основание с. сч. p2»

Предусловие

Прецедент «Преобразовать» завершён.

Основной поток событий

1) Пользователь изменяет основания систем счисления p2.
2) Введённое пользователем число отображается в системе счисления с выбранным основанием.

Альтернативный поток событий 1. Количество разрядов в результате превышает размер поля вывода визуального компонента.

3.1. Пользователь получает окно с сообщением.

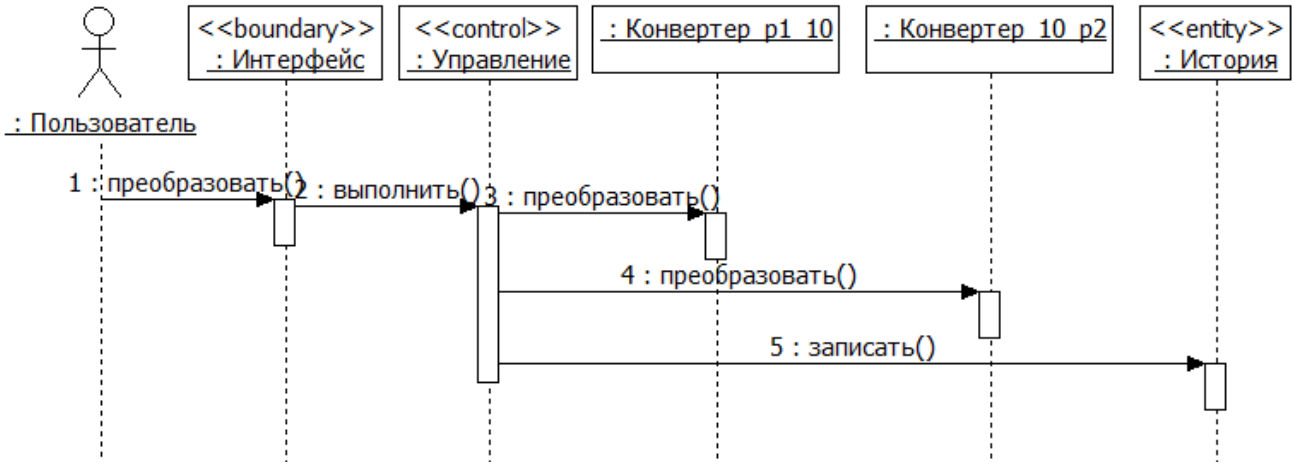
3.2. Приложение переходит в режим Ввод и редактирование.

Диаграмма классов модели объектно-ориентированного анализа.

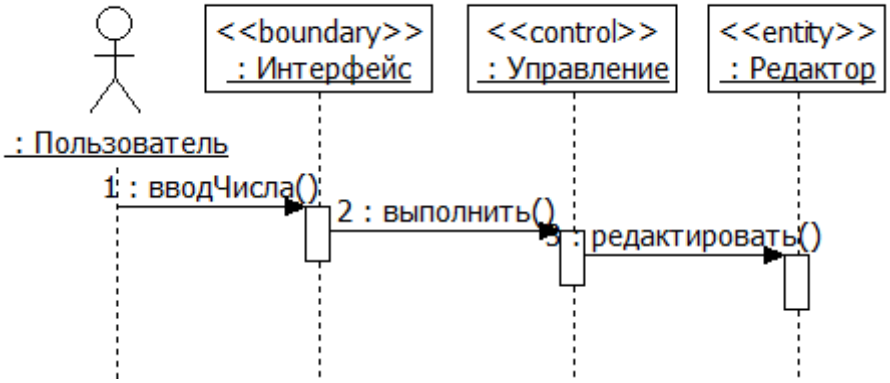
Проанализировав прецеденты можно выделить следующие классы анализа приложения. Они представлены на диаграмме классов анализа ниже.

Обмен сообщениями между объектами. Диаграмма последовательностей.

Давайте спроектируем обмен сообщениями между объектами в процессе выполнения прецедента «Преобразовать». Добавим объект класса Управление для организации обмена сообщениями между объектами в ходе выполнения прецедента. На диаграмме последовательностей приведённой ниже приведёна последовательность сообщений между объектами в основном потоке событий прецедента «Преобразовать».



На диаграмме последовательностей приведённой ниже приведёна последовательность сообщений между объектами в процессе реализации прецедента «Ввести число».



На диаграмме последовательностей приведённой ниже приведёна последовательность сообщений между объектами в процессе реализации прецедента «История».

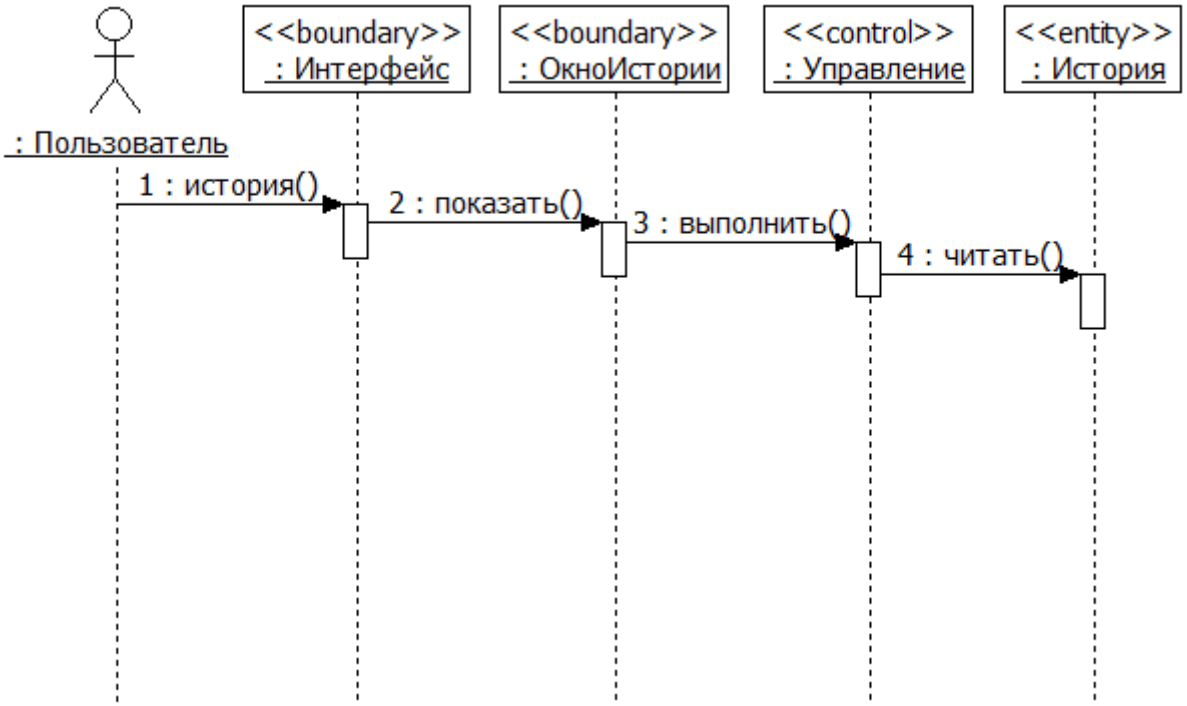
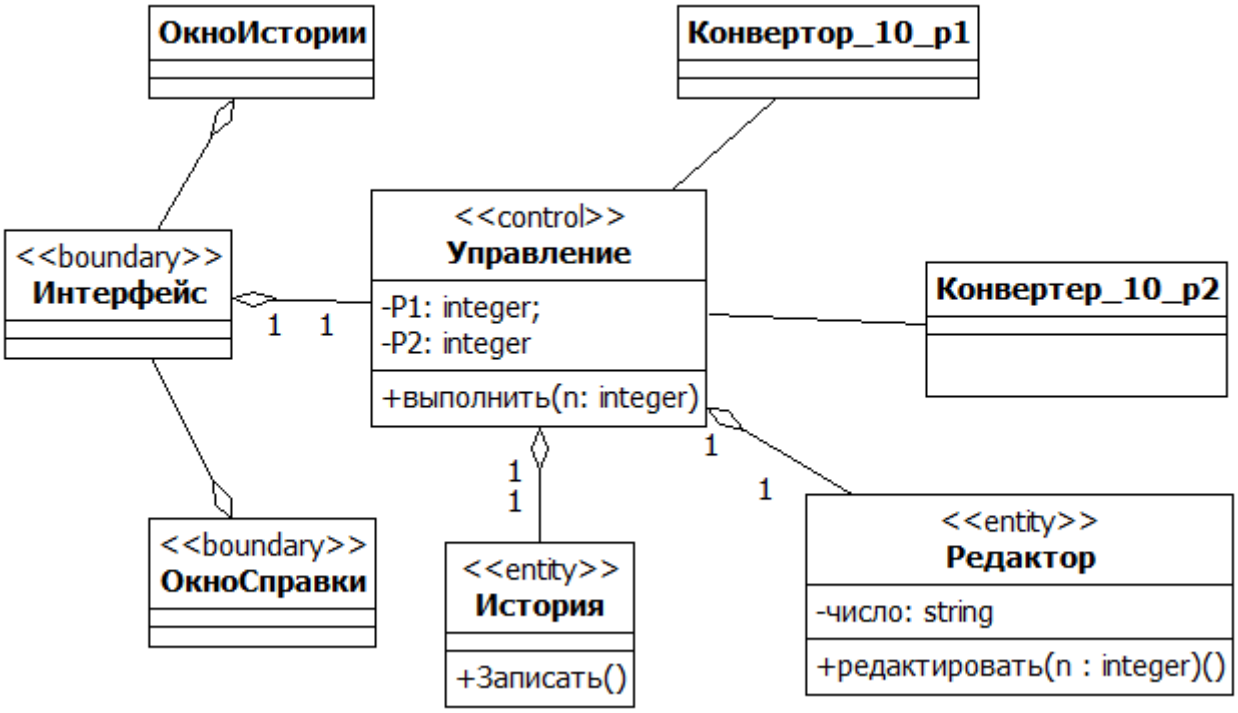
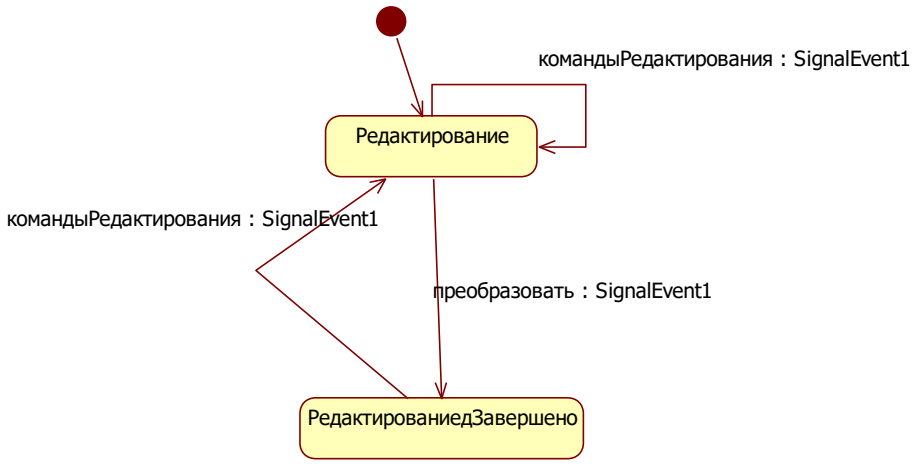


Диаграмма классов проекта

Проанализировав сообщения, которыми обмениваются классы в процессе выполнения прецедентов можно построить следующую диаграмму классов проекта. Для упрощения взаимодействия между классами в процессе работы приложения добавим класс «Управление». Тогда наша диаграмма примет следующий вид:



Из диаграммы классов видно, что объект класса «Интерфейс» вызывает методы класса «Управление» и «Справка». Объект же класса «Управление» в свою очередь вызывает методы объектов классов «Редактор», «История» и «Конвертор_p1_10», «Конвертор_10_p2». Диаграмма состояний для объекта класса «Управление» представлена ниже:



Объект класса «Управление» может находиться в двух состояниях: «Редактирование» и «Редактирование завершено».

Порядок выполнения разработки.

Разработка приложения разбита на практические работы, в каждой из которых вы реализуете один или несколько логически связанных классов приложения.

Практическая работа. Конвертор чисел из десятичной системы счисления в систему счисления с заданным основанием

Цель: Сформировать практические навыки реализации классов на языке C#.

Задание

1. Реализовать преобразователь действительных чисел со знаком из десятичной системы счисления в систему счисления с заданным основанием p, в соответствии с приведенной ниже спецификацией, используя класс. Основание системы счисления p принадлежит диапазону значений от 2 до 16.
2. Протестировать каждый метод класса.

Спецификация класса «Преобразователь чисел из десятичной системы счисления в систему счисления с заданным основанием p».

ADT Conver_10_p

Данные

Преобразователь действительных чисел из десятичной системы счисления в систему счисления с заданным основанием (тип Conver_10_p). Основание системы счисления p - это целое число, со значением, принадлежащим диапазону от 2 до 16 и целое число c, определяющее точность представления результата, выраженную в количестве разрядов.

Операции. Операции представлены в таблице ниже.

Do(double n, int p, int c)	*Выполнить преобразование*
Вход:	Десятичное действительное число n. Основание системы счисления p. Точность преобразования дроби, заданная числом разрядов дробной части результата c. Например: Do(-17.875,16,3) = “-A1.E” .
Процесс:	Выполняет преобразование десятичного действительного числа n, в систему счисления с основанием p и точностью c. Например: ***Do*** (”-17.875”,16,3) = “-A1.E”.
Выход:	Строка результата. Например: ***Do***(”-17.875”) = “-A1.E”.
int_to_Char(int d)	Преобразовать целое значение в цифру системы счисления с основанием p.
Вход:	d – значение типа int – целое, соответствующее цифре в системе счисления с основанием p.
Предусловия:	Нет.
Процесс:	Преобразует целое d в соответствующую ему цифру в системе счисления с основанием p, значение типа Char. Например: int_to_Char (14) = “E”.

Выход:	Значение типа char.
Постусловия:	Нет.
int_to_P(int n, int p)	Преобразовать целое в строку.
Вход:	n – целое число в системе счисления с основанием 10. p – основание системы счисления результата.
Предусловия:	Нет.
Процесс:	Преобразует целое n в строку, содержащую целое число в системе счисления с основанием p. Например: *int_to_P(161, 16) = "A1"*
Выход:	Строка.
Постусловия:	Нет.
flt_to_P(double n, int p, int c)	Преобразовать дробь в строку.
Вход:	n – дробь в системе счисления с основанием 10, p – основание системы счисления, c – точность представления дроби.
Предусловия:	Нет.
Процесс:	Преобразует дробь n в строку, содержащую дробь в системе счисления с основанием p с точностью c. Например: flt_to_P(0.9375, 2, 4) «1111»
Выход:	Строка.
Постусловия:	Нет.

end Conver_10_p

Рекомендации к выполнению.

1. Тип данных реализовать, используя статический класс.
2. Тип данных сохраните в отдельном файле Conver_10_p.

Ниже представлена заготовка описания класса Conver_10_p. Вам необходимо написать код методов и протестировать методы.

```
namespace Конвертор
{
```

```
    class Conver_10_P
    {
        //Преобразовать целое в символ.
        public static char int_to_Char(int n) {        }
        //Преобразовать десятичное целое в с.сч. с основанием р.
        public static string int_to_P(int n, int p) {        }
        //Преобразовать десятичную дробь в с.сч. с основанием р.
        public static string flt_to_P(double n, int p, int c) {
}
        //Преобразовать десятичное
        //действительное число в с.сч. с осн. р.
        public static string Do(double n, int p, int c) {        }
    }
}
```

Содержание отчета

1. Задание.
2. Текст программы.
3. Тестовые наборы данных для тестирования класса.

Контрольные вопросы

1. Что такое инкапсуляция?
2. Как синтаксически представлено поле в описании класса?
3. Как синтаксически представлен метод в описании класса?
4. Как синтаксически представлено простое свойство в описании класса?
5. Особенности описания методов класса?
6. Видимость идентификаторов в описании класса?
7. В чём особенности статических методов?
8. В чём особенности статических классов?

Практическая работа. Класс «Конвертер р_10» - преобразователь чисел из системы счисления с основанием р в десятичную систему счисления

Цель: Сформировать практические навыки реализации классов на языке C#.

Задание

1. Реализовать преобразователь действительных (конвертер p_10) чисел из системы счисления с основанием p в десятичную систему счисления в соответствии с приведенной ниже спецификацией, используя класс. Основание системы счисления p принадлежит диапазону значений от 2 до 16.
2. Протестировать каждый метод класса.

Спецификация класса «Конвертер p_10» - преобразователь действительных чисел со знаком из системы счисления с основанием p в десятичную систему счисления.

ADT Conver_p_10

Данные

Преобразователь действительных чисел из заданной системы счисления с основанием p в десятичную систему счисления (тип Conver_p_10). Основание системы счисления со значением, принадлежащим диапазону от 2 до 16.

Операции

Операции приведены в таблице ниже.

dval(string P_num, int P)	***Выполнить преобразование***
Вход:	P_num - строковое представление действительного числа в системе счисления с основанием p. Например: dval(“A5.E”, 16)
Процесс:	Выполняет преобразование действительного числа, представленного строкой в числовое представление. Например: dval(“A5.E”, 16) = -165.875.
Выход:	Вещественное число.
Постусловия:	Нет.

char_To_num(char ch)	***Преобразовать символ в целое***
Вход:	ch – значение типа char – символ, изображающий цифру системы счисления с основанием p.
Предусловия:	Нет.
Процесс:	Преобразует символ ch в значение целого типа. Например: ***PCharToInt('A')*** = 10.
Выход:	Вещественное число.
Постусловия:	Нет.
convert(string P_num, int P, double weight)	***Преобразовать строку в вещественное число.***
Вход:	P_num – строка, изображающая цифры целой и дробной частей вещественного числа в системе счисления с основанием p без разделителя. weight – вес единицы старшего разряда целой части числа.
Предусловия:	Нет.
Процесс:	Преобразует строку P_num, содержащую цифры целой и дробной частей вещественного числа в системе счисления с основанием p без разделителя в вещественное число. Например: convert ("A5E1", 16, 16)
Выход:	Вещественное число.
Постусловия:	Нет.

end Conver_p_10

Рекомендации к выполнению.

Описание класса может выглядеть следующим образом:

```
namespace Конвертор
{
    public class Conver_P_10
    {
```

```
        //Преобразовать цифру в число.
        static double char_To_num(char ch) {        }
        //Преобразовать строку в число
        private static double convert(string P_num, int P, double
weight) { }
        //Преобразовать из с.сч. с основанием p
        //в с.сч. с основанием 10.
        public static double dval(string P_num, int P) {         }
    }
}
```

3. Тип данных реализовать, используя статический класс.
4. Сохраните класс в отдельном файле Conver_p_10.

Содержание отчета

1. Задание.
2. Текст программы.
3. Тестовые наборы данных для тестирования класса.

Контрольные вопросы

1. Что такое инкапсуляция?
2. Как синтаксически представлено поле в описании класса?
3. Как синтаксически представлен метод в описании класса?
4. Как синтаксически представлено простое свойство в описании класса?
5. Особенности описания методов класса?
6. Видимость идентификаторов в описании класса?
7. В чём особенности статических методов?
8. В чём особенности статических классов?
9. Как вызываются статические методы?

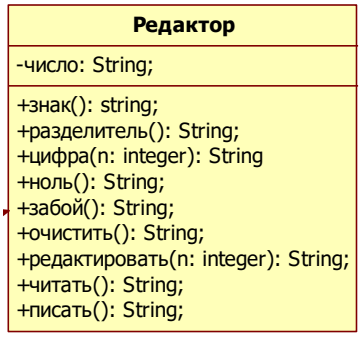
Практическая работа. Редактор чисел в системе счисления с основанием p

Цель: Сформировать практические навыки реализации классов средствами объектно-ориентированного языка программирования C#.

Задание

1. Разработать и реализовать класс Editor «Редактор действительных чисел представленных в системе счисления с основанием p», используя класс языка высокого уровня. Основание системы счисления p принимает значение из диапазона 2..16. Все команды редактора удобно пронумеровать, начиная с команды добавить 0 целыми числами от 0. При реализации интерфейса номера команд удобно хранить в свойстве Tag, которое имеется у визуальных компонентов.

Атрибуты и операции класс представлены ниже.



2. Ответственность класса Editor (редактор) – хранение, ввод и редактирование строкового представления числа, представленного в системе счисления с основанием p. Класс должен обеспечивать: добавление символов (AddDigit), соответствующих p-ичным цифрам (p от 2 до 16); добавления нуля (AddZero()); добавление разделителя целой и дробной частей (AddDelim()); забой символа - удаление символа, стоящего справа (BS); очистку - установку нулевого значения числа (Clear); чтение строкового представления p-ичного числа (Number).
3. Протестировать каждый метод класса.

Рекомендации к выполнению

Описание класса может выглядеть следующим образом:

```
namespace Конвертор
{
    class Editor
    {
        //Поле для хранения редактируемого числа.
        string number = "";
        //Разделитель целой и дробной частей.
```

```
        const string delim = ".";
        //Ноль.
        const string zero = "0";
         //Свойствое для чтения редактируемого числа.
        public string Number
        {  get {   }        }
        //Добавить цифру.
        public string AddDigit(int n) {        }
        //Точность представления результата.
        public int Acc(){        }
        //Добавить ноль.
        public string AddZero(){        }
        //Добавить разделитель.
        public string AddDelim(){        }
        //Удалить символ справа.
        public string Bs() {        }
        //Очистить редактируемое число.
        public string Clear() {        }
        //Выполнить команду редактирования.
        public string DoEdit(int j) {        }
    }
}
```

Класс сохраните в файле Editor. В разделе описания констант опишите следующие константы: «разделитель целой и дробной частей» строкового типа; «строковое представление нуля» строкового типа.

Содержание отчета

1. Задание.
2. Текст программы.
3. Тестовые наборы данных для тестирования класса.

Контрольные вопросы

1. В чём состоит особенность раздела описания класса с уровнем доступа protected?
2. В чём состоит особенность раздела описания класса с уровнем доступа private?
3. В чём состоит особенность раздела описания класса с уровнем доступа public?
4. В чём состоит особенность инициализации полей ссылочного типа и констант в конструкторе?
5. Что такое this?

6. Как описываются поля в классах?
7. Какой операцией создаются объекты классов?
8. Как вызвать нестатический метод класса?

Практическая работа. Класс История

Цель: Сформировать практические навыки реализации классов средствами объектно-ориентированного программирования языка C#; использования библиотечного класса обобщённой коллекции List<> для обработки данных.

Задание

1. Разработать и реализовать класс History «История», используя класс языка C#. Класс отвечает за документирование выполнения пользователем переводов чисел. Объекты класса хранят исходные числа, результаты преобразования и основания систем счисления исходного числа и результата.

   Атрибуты и операции класс представлены ниже.

История
Запись(i: integer): String; ДобавитьЗапись(a: String); Записей(): integer ОчиститьИсторию();
Обязанность: ввод, вывод, хранение данных введённых пользователем и полученных результатов.

2. Класс должен отвечать за ввод, вывод, хранение данных введённых пользователем и полученных результатов. Класс должен обеспечивать:
   - добавление записи (ДобавитьЗапись) - строки, содержащей введённое пользователем число, результат его преобразования и основания систем счисления исходной и той, в которую число преобразовано;
   - извлечение записи по её номеру в списке (Запись);
   - очистка списка (ОчиститьИсторию);

- конструктор (Запись);
- текущий размер списка в числе записей (Записей);

3. Протестировать каждый метод класса.

Рекомендации к выполнению

1. Создайте консольное приложение, в которое добавьте класс History и сохраните его в файле History.
2. В этот же файл добавьте структуру (struct) Record с четырьмя полями для хранения исходного числа, результата и оснований их систем счисления. В структуру добавьте конструктор и метод ToString() для преобразования значения в формат строки.
3. Класс History, постройте на основе библиотечного класса коллекции List<Record> из пространства имён System.Collections.Generic.
4. В классе History опишите поле List<Record> L – список значений типа Record. Для поля опишите уровень доступа private.
5. Следующие операции класса опишите с уровнем доступа public:
   - добавить запись (AddRecord) - строку, содержащую введённое пользователем число, результат его преобразования и основания их систем счисления;
   - извлечь запись по её номеру из списка L;
   - очистить историю (Clear);
   - конструктор (History);
   - текущий размер списка в числе записей (Count);

Описание структуры Record и класса History могут иметь следующий вид:

```
public struct Record
{
    int p1;
    int p2;
    string number1;
    string number2;
    public Record(int p1, int p2, string n1, string n2) {       }
    public override string ToString() {       }
}
```

```
    public class History
    {
        List<Record> L;
        public Record this[int i] {        }
        public void AddRecord(int p1, int p2, string n1, string n2) {
}
        public void Clear() {       }
        public int Count() {       }
        public History() {       }
    }
```

Содержание отчета

1. Задание.
2. Текст программы.
3. Тестовые наборы данных для тестирования класса.

Контрольные вопросы

1. В чём состоит особенность обобщённой коллекции List< >?
2. В чём состоит отличие типа struct от типа class?
3. Как создаются объекты типа struct?
4. В чём состоит особенность раздела описания класса с уровнем доступа private?
5. В чём состоит особенность раздела описания класса с уровнем доступа public?
6. В чём состоит особенность инициализации полей ссылочного типа и констант в конструкторе?
7. Что такое this?

Практическая работа. Класс Управление для «Конвертора p1_p2»

Цель: Сформировать практические навыки реализации классов на языке C#.

Задание

1. Реализовать Управление для «Конвертера p1_p2».
2. Протестировать каждый метод класса.

Спецификация класса Управление для «Конвертера p1_p2».

ADT Control_

Данные

Объект класса **Control_** (Управление) отвечают за координацию действий между классом «Интерфейс» и классами «Редактор», «Конвертер p1_10», «Конвертер 10_p2», «История». Объект класса **Control_** содержат поля: **ed** типа Editor, **his** типа История, и свойства: Pin типа int (основание системы счисления исходного числа), Pout типа int (основание системы счисления результата), **St** типа State (состояние конвертера). Он может находиться в одном из двух состояний: «Редактирование», «Преобразовано». Объекты этого типа изменяемы.

Операции

Операции представлены в таблице ниже.

Control_	***Конструктор***
Вход:	Нет.
Процесс:	Создаёт объект Управление типа (тип **Control_**) и инициирует поля объекта начальными значениями.
DoCommand	Выполнить команду.
Вход:	n - целое значение, номер выполняемой команды.
Предусловия:	Нет.
Процесс:	В зависимости от значения n и состояния (St) передаёт сообщение объекту Редактор или Преобразователь и изменяет состояние. Возвращает строку результата: либо отредактированное число, либо результат преобразования.
Выход:	Строка.
Постусловия:	Нет.

end Control_

Рекомендации к выполнению

1. Тип данных реализовать, используя класс.
2. Для записи и считывания полей «преобразователя» используйте свойства.
3. Тип данных реализуйте в отдельном файле Control_.

Пример описания класса Управление приведён ниже.

```
namespace Конвертор
{
    class Control_
    {
        //Основание системы сч. исходного числа.
        const int pin = 10;
        //Основание системы сч. результата.
        const int pout = 16;
        //Число разрядов в дробной части результата.
        const int accuracy = 10;
        public История his = new История();
        public enum State {Редактирование, Преобразовано}
        //Свойство для чтения и записи состояние Конвертера.
        public State St { get; set; }
        //Конструктор.
        public Control_()
        {
            St = State.Редактирование;
            Pin = pin;
            Pout = pout;
        }
        //объект редактор
        public Editor ed = new Editor();
        //Свойство для чтения и записи основание системы сч. p1.
        public int Pin { get; set; }
        //Свойство для чтения и записи основание системы сч. p2.
        public int Pout { get; set; }
        //Выполнить команду конвертера.
        public string DoCmnd(int j)
        {
            if (j == 19)
            {
                double r = Conver_P_10.dval(ed.Number, (Int16)Pin);
```

```
                string res = Conver_10_P.Do(r, (Int32)Pout,acc());
                St = State.Преобразовано;
                his.ДобавитьЗапись(Pin, Pout, ed.Number, res);
                return res;
            }
            else
            {
                St = State.Редактирование;
                return ed.DoEdit(j);
            }

        }
        //Точность представления результата.
        private int acc()
        {
            return (int)Math.Round(ed.Acc() * Math.Log(Pin) /
Math.Log(Pout) + 0.5);
        }
    }
}
```

Содержание отчета

1. Задание.
2. Текст программы.
3. Тестовые наборы данных для тестирования класса.

Контрольные вопросы

1. Что такое инкапсуляция?
2. Как синтаксически представлено поле в описании класса?
3. Как синтаксически представлен метод в описании класса?
4. Как синтаксически представлено простое свойство в описании класса?
5. Особенности описания методов класса?
6. Особенности описания и назначение конструктора класса?
7. Видимость идентификаторов в описании класса?

8. Особенности вызова методов применительно к объектам класса?

Практическая работа. Интерфейс приложения «Конвертор p1_p2»

Цель: Сформировать практические навыки реализации графических интерфейсов пользователя (GUI) на основе библиотеки визуальных компонентов.

Задание

1. Реализовать «Интерфейс» приложения «Конвертер p1_p2», используя библиотечный класс формы и визуальные компоненты.
2. Протестировать методы класса.

Спецификация класса «Интерфейс».

Интерфейс приложения представлен на рис. 18.

ADT TPanel_p_p

Данные

«Интерфейс» конвертера действительных чисел из системы счисления с основанием p1 в систему счисления с основанием p2 предназначен для: выбора оснований систем счисления p1, p2 из диапазона от 2..16; ввода и редактирования действительного числа со знаком в системе счисления с выбранным основанием p1; отображения результата – представления ввёдённого числа в системе счисления с основанием p2; отображения справки о приложении; отображения истории текущего сеанса работы пользователя с приложением.

«Интерфейс» несёт на себе визуальные компоненты, реализующие выполнения команд преобразователя и объект «Управление» класса Control_.

Операции. Операции представлены в таблице ниже.

Наименование	**Пояснение**

trackBar1_Scroll	Обработчик события Scroll для компонента trackBar1.
Вход:	object sender, EventArgs e. sender – указатель на объект, который явился инициатором события Scroll. e - это объект базового класса для классов, содержащих данные о событии.
Предусловия:	Пользователь перетаскивает бегунок компонента trackBar1.
Процесс:	Обновляет свойства визуальных компонентов формы, связанных с изменением основания системы счисления p1 исходного числа. Устанавливает новое значение основания системы счисления. Обновляет состояние командных кнопок.
Постусловия:	Обновления выполнены.
Выход:	Нет.
numericUpDown1_ValueChanged	Обработчик события ValueChanged для компонента numericUpDown1.
Вход:	object sender, EventArgs e. sender – указатель на объект, который явился инициатором события ValueChanged.
Предусловия:	Пользователь изменяет p1 с помощью компонента numericUpDown1.
Процесс:	Обновляет свойства визуальных компонентов формы, связанных с изменением основания системы счисления p1 исходного числа. Устанавливает новое значение основания системы счисления. Обновляет состояние

	командных кнопок.
Постусловия:	Обновления выполнены.
Выход:	Нет.
trackBar2_Scroll	Обработчик события Scroll для компонента trackBar2.
Вход:	object sender, EventArgs e. sender – указатель на объект, который явился инициатором события Scroll.
Предусловия:	Пользователь перетаскивает бегунок компонента trackBar2.
Процесс:	Обновляет свойства визуальных компонентов формы, связанных с изменением основания системы счисления p2 исходного числа. Устанавливает новое значение основания системы счисления.
Постусловия:	Обновления выполнены.
Выход:	Нет.
numericUpDown2_ValueChanged	Обработчик события ValueChanged для компонента numericUpDown2.
Вход:	object sender, EventArgs e. sender – указатель на объект, который явился инициатором события ValueChanged.
Предусловия:	Пользователь изменяет p2 с помощью компонента numericUpDown2.
Процесс:	Обновляет свойства визуальных компонентов формы, связанных с изменением основания системы счисления p2 результата. Устанавливает новое значение основания системы счисления. Обновляет состояние командных кнопок.

Постусловия:	Обновления выполнены.
Выход:	Нет.
numericUpDown1_ValueChanged	Обработчик события ValueChanged для компонента numericUpDown1.
Вход:	object sender, EventArgs e. sender – указатель на объект, который явился инициатором события ValueChanged.
Предусловия:	Пользователь изменяет p1 с помощью компонента numericUpDown1.
Процесс:	Обновляет свойства визуальных компонентов формы, связанных с изменением основания системы счисления p1 результата. Устанавливает новое значение основания системы счисления. Обновляет состояние командных кнопок.
Постусловия:	Обновления выполнены.
Выход:	Нет.
TPanelp_p_Load	Обработчик события Load для компонента TPanelp_p.
Вход:	(object sender, EventArgs e). sender – указатель на объект, который явился инициатором события Load.
Предусловия:	Форма загружается в память.
Процесс:	Устанавливает начальные значения свойств визуальных компонентов формы после загрузки формы.
Выход:	Нет.
Постусловия:	Установка свойств выполнена.
DoCmnd(int j)	Выполнить команду.
Вход:	j значение целого типа – номер команды

	преобразователя.
Предусловия:	Пользователь нажал командную кнопку команды с номером j.
Процесс:	Передаёт сообщение объекту Управление и отображает возвращаемый им результат. Вызывается метод объекта Управление и передаётся номер набранной пользователем команды Конвертора.
Выход:	Нет.
Постусловия:	Обновляется состояние Интерфейса.
button_Click	Обработчик события Click для командных кнопок.
Вход:	object sender, EventArgs e. sender: object – указатель на объект, который явился инициатором события Click.
Предусловия:	Пользователь нажал командную кнопку.
Процесс:	Извлекает из свойства Tag командной кнопки номер соответствующей ей команды. Вызывает метод DoCmnd Интерфейса и передаёт в него номер команды.
Выход:	Нет.
Постусловия:	Нет.
TPanelp_p_KeyPress	Обработчик события KeyPress для алфавитно-цифровых клавиш клавиатуры.
Вход:	object sender, KeyPressEventArgs e.
Предусловия:	Пользователь нажал алфавитно-цифровую клавишу клавиатуры.
Процесс:	Определяет по нажатой алфавитно-цифровой клавише номер соответствующей ей команды.

	Вызывает метод DoCmnd Интерфейса и передаёт в него номер команды.
Выход:	Нет.
Постусловия:	Команда пользователя вызвана.
TPanelp_p_KeyDown	Обработчик события KeyDown для клавиш управления клавиатуры.
Вход:	(object sender, KeyEventArgs e)
Предусловия:	Пользователь нажал клавишу управления клавиатуры.
Процесс:	Определяет по нажатой клавише управления номер соответствующей ей команды. Вызывает метод DoCmnd Интерфейса и передаёт в него номер команды.
Выход:	нет.
Постусловия:	Команда пользователя вызвана.
UpdateP1	Выполнить обновления связанные с изменением p1.
Вход:	Нет.
Предусловия:	Изменено основание с.сч. p1 исходного числа.
Процесс:	Выполняет необходимые обновления при смене ос. с. сч. p1.
Выход:	Нет.
Постусловия:	Состояние кнопок обновлено.
UpdateP2	Выполнить обновления связанные с изменением p2.
Вход:	Нет.
Предусловия:	Изменено основание с.сч. p2 результата.
Процесс:	Выполняет необходимые обновления при смене ос. с. сч. p2.

Выход:	Нет.
Постусловия:	Состояние кнопок обновлено.
UpdateButtons	
Вход:	Нет.
Предусловия:	Изменено основание с.сч. p1 исходного числа.
Процесс:	Обновляет состояния командных кнопок предназначенных для ввода цифр выбранной системы счисления p1.
Выход:	Нет.
Постусловия:	Состояние кнопок обновлено.
выходToolStripMenuItem_Click	Команда Выход основного меню класса TPanelp_p формы.
Вход:	object sender, EventArgs e.
Предусловия:	Пользователь кликает мышью на пункте Выход основного меню формы.
Процесс:	Завершает работу приложения.
Выход:	Нет.
Постусловия:	Приложение завершено.
справкаToolStripMenuItem_Click	Команда Справка основного меню класса TPanelp_p формы.
Вход:	object sender, EventArgs e
Предусловия:	Пользователь кликает мышью на пункте Справка основного меню формы.
Процесс:	Показывает окно со справкой по приложению.
Выход:	Нет.
Постусловия:	Отображено окно справки.
историяToolStripMenuItem_Click	Команда История основного меню класса TPanelp_p формы.
Вход:	object sender, EventArgs e

Предусловия:	Пользователь кликает мышью на пункте История основного меню формы.
Процесс:	Открывает окно История.
Выход:	Нет.
Постусловия:	Окно История - открыто.

end TPanel_p_p

Рекомендации к выполнению.

1. Интерфейс приложения будет состоять из трёх форм: основная форма класс TPanel_p_p, HistoryForm – форма для отображения истории, AboutBox – форма, информирующая о приложении. Все они наследники класса Form.
2. Для реализации интерфейса приложения разместите на форме компоненты, описанные в таблице ниже.

Имя компонента	**Имя класса**	**Назначение**
label1	Label	Исходное число.
label2	Label	Результат.
label3	Label	Подпись к исходному числу.
label4	Label	Подпись к результату.
trackBar1	TrackBar	Изменять основание с. сч. p1.
trackBar2	TrackBar	Изменять основание с. сч. p2.
numericUpDown1	NumericUpDown	Изменять основание с. сч. p1.
numericUpDown2	NumericUpDown	Изменять основание с. сч. p2.
button1 – button10	Button	Ввод цифр от 0 – 9.
button11 – button16	Button	Ввод цифр от A – F.
button17	Button	Разделитель целой и дробной частей (,).

button18	Button	Забой крайнего правого символа (BS).
button19	Button	Удалить исходное число (CL).
button20	Button	Выполнить (Execute).
menuStrip1	menuStrip1	Основное меню главной формы.
выходToolStripMenuItem	ToolStripMenuItem	Завершение работы приложения.
историяToolStripMenuItem	ToolStripMenuItem	Просмотр журнала сеанса работы.
справкаToolStripMenuItem	ToolStripMenuItem	Справка по приложению.

Для этого перейдите на вкладку Конструктор формы окна редактора и сделайте форму активной. Добавьте на форму из вкладки «Панели элементов» из раздела «Все формы Windows Forms» командные кнопки Button. Кнопки помещайте на форму снизу вверх слева направо, начиная с 0. Порядок добавления кнопок на форму должен совпадать с порядком их нумерации. Выделите все кнопку и в окне Свойства на закладке Свойства раскройте свойство Font (Шрифт) и установите требуемые вам свойства для шрифта. Затем в свойство Text (Текст) занесите соответствующие цифры. Затем в свойство Tag каждой кнопки занесите целое число соответствующее кнопке: кнопки для ввода цифр нумеруйте от 0 до 15; кнопка разделитель целой и дробной части – 16; удалить крайний символ слева (BS) - 17; очистить всё (CL) – 18; выполнить (Enter) - 19. Измените размер кнопок (свойство Size), если это необходимо.

Добавьте на форму два компонента TrackBar из вкладки «Панель элементов» из раздела «Все формы Windows Forms». С помощью окна «Свойства» установите в их свойства Minimum значение 2, а в Maximum - значение 16. Рядом с каждым из них разместите компонент NumericUpDown

«числовое поле со стрелками вверх/вниз». Добавьте на форму два компонента Label. В одном будем отображать основание системы счисления p1, в другом – подпись к нему. Добавьте на форму ещё два компонента Label для основания системы счисления результата p2.

Добавьте на форму компонент MenuStrip из вкладки «Панель элементов» из раздела «Все формы Windows Forms». Добавьте в компонент MenuStrip три пункта меню MenuItem. С помощью окна «Свойства» установите в их свойства Text значение Выход, История, Справка. Полученная форма будет иметь вид, как представлено на рис.18 .

Для настройки отклика приложения на действия пользователя необходимо создать обработчики событий для тех компонент интерфейса, которыми будет манипулировать пользователь. Обработчик события для выделенного компонента создаётся с помощью окна «Свойства» закладки «События». Правее выбранного события в свободном поле необходимо сделать двойной клик мышью и в классе формы появится текст шаблона обработчика этого события. Затем вам необходимо описать в обработчике необходимые вам действия. Обработчики событий для компонентов главной формы приведены в таблице ниже.

Имя компонента	**Событие**	**Обработчик**
TPanelp_p (главная форма)	Load	TPanelp_p_Load
TPanelp_p (главная форма)	KeyDown	TPanelp_p_KeyDown
TPanelp_p (главная форма)	KeyPress	TPanelp_p_KeyPress
trackBar1	Scroll	trackBar1_Scroll
trackBar2	Scroll	trackBar1_Scroll
numericUpDown1	ValueChanged	numericUpDown1_ValueChanged
numericUpDown2	ValueChanged	numericUpDown1_ValueChanged
button1 – button19	Click	button_Click
выходToolStripMenuItem	Click	выходToolStripMenuItem_Click

историяToolStripMenuItem	Click	историяToolStripMenuItem1_Click
справкаToolStripMenuItem	Click	справкаToolStripMenuItem_Click

Теперь можно запустить приложение и протестировать работу командных кнопок для ввода цифр действительного числа, представленного в выбранной системе счисления. А также для изменения оснований систем счисления исходного числа и результата. Ниже приведён текст модуля главного окна приложения.

```
namespace Конвертор
{
    public partial class Form1 : Form
    {
        //Объект класса Управление.
        Control_ ctl = new Control_();
        public Form1()
        {
            InitializeComponent();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            label1.Text = ctl.ed.Number;
            //Основание с.сч. исходного числа p1.
            trackBar1.Value = ctl.Pin;
            //Основание с.сч. результата p2.
            trackBar2.Value = ctl.Pout;
            label3.Text = "Основание с. сч. исходного числа " +
trackBar1.Value;
            label4.Text = "Основание с. сч. результата " +
trackBar2.Value;
            label2.Text = "0";
            //Обновить состояние командных кнопок.
            this.UpdateButtons();
        }
        //Обработчик события нажатия командной кнопки.
        private void button1_Click(object sender, EventArgs e)
        {
            //ссылка на компонент, на котором кликнули мышью
            Button but = (Button)sender;
            //номер выбранной команды
            int j = Convert.ToInt16(but.Tag.ToString());
            DoCmnd(j);
        }
        //Выполнить команду.
        private void DoCmnd(int j)
        {
```

```
            if (j == 19) { label2.Text = ctl.DoCmnd(j);  }
            else
            {
                if (ctl.St == Control_.State.Преобразовано)
                {
                    //очистить содержимое редактора
                    label1.Text = ctl.DoCmnd(18);
                }
                //выполнить команду редактирования
                label1.Text = ctl.DoCmnd(j);
                label2.Text = "0";
            }
        }
        //Обновляет состояние командных кнопок по основанию с. сч.
исходного числа.
        private void UpdateButtons()
        {
            //просмотреть все компоненты формы
            foreach (Control i in this.Controls)
            {
                if (i is Button)//текущий компонент - командная кнопка
                {
                    int j = Convert.ToInt16(i.Tag.ToString());
                    if (j < trackBar1.Value)
                    {
                        //сделать кнопку доступной
                        i.Enabled = true;
                    }
                    if ((j >= trackBar1.Value) && (j <= 15))
                    {
                        //сделать кнопку недоступной
                        i.Enabled = false;
                    }
                }
            }
        }
        //Изменяет значение основания с.сч. исходного числа.
        private void trackBar1_Scroll(object sender, EventArgs e)
        {
            numericUpDown1.Value = trackBar1.Value;
            //Обновить состояние командных кнопок.
            this.UpdateP1();
        }
        //Изменяет значение основания с.сч. исходного числа.
        private void numericUpDown1_ValueChanged(object sender,
EventArgs e)
        {
            //Обновить состояние.
            trackBar1.Value = Convert.ToByte(numericUpDown1.Value);
            //Обновить состояние командных кнопок.
            this.UpdateP1();
```

```
        }
        //Выполняет необходимые обновления при смене ос. с.сч. p1.
        private void UpdateP1()
        {
            label3.Text = "Основание с. сч. исходного числа " +
trackBar1.Value;
            //Сохранить p1 в объекте управление.
            ctl.Pin = trackBar1.Value;
            //Обновить состояние командных кнопок.
            this.UpdateButtons();
            label1.Text = ctl.DoCmnd(18);
            label2.Text = "0";
        }
        //Изменяет значение основания с.сч. результата.
        private void trackBar2_Scroll(object sender, EventArgs e)
        {
            //Обновить состояние.
            numericUpDown2.Value = trackBar2.Value;
            this.UpdateP2();
        }
        //Изменяет значение основания с.сч. результата.
        private void numericUpDown2_ValueChanged(object sender,
EventArgs e)
        {
            trackBar2.Value = Convert.ToByte(numericUpDown2.Value);
            this.UpdateP2();
        }
        //Выполняет необходимые обновления при смене ос. с.сч. p2.
        private void UpdateP2()
        {
            //Копировать основание результата.
            ctl.Pout = trackBar2.Value;
            //Пересчитать результат.
            label2.Text = ctl.DoCmnd(19);
            label4.Text = "Основание с. сч. результата " +
trackBar2.Value;
        }
        //Пункт меню Выход.
        private void выходToolStripMenuItem_Click(object sender,
EventArgs e)
        {
             Close();
        }
        //Пункт меню Справка.
        private void справкаToolStripMenuItem_Click(object sender,
EventArgs e)
        {
            AboutBox1 a = new AboutBox1();
            a.Show();
        }
        //Пункт меню История.
```

```
        private void историяToolStripMenuItem1_Click(object sender,
EventArgs e)
        {
            Form2 history = new Form2();
            history.Show();
            if (ctl.his.Записей() == 0)
            {
                history.textBox1.AppendText("История пуста");
                return;
            }
            //Ообразить историю.
            for (int i = 0; i < ctl.his.Записей(); i++)
            {
                history.textBox1.AppendText(ctl.his[i].ToString());
            }
        }
        //Обработка алфавитно-цифровых клавиш.
        private void Form1_KeyPress(object sender, KeyPressEventArgs
e)
        {
            int i = -1;
            if (e.KeyChar >= 'A' && e.KeyChar <= 'F') i =
(int)e.KeyChar - 'A' + 10;
            if (e.KeyChar >= 'a' && e.KeyChar <= 'f') i =
(int)e.KeyChar - 'a' + 10;
            if (e.KeyChar >= '0' && e.KeyChar <= '9') i =
(int)e.KeyChar - '0';
            if (e.KeyChar == '.') i = 16;
            if ((int)e.KeyChar == 8) i = 17;
            if ((int)e.KeyChar == 13) i = 19;
            if ((i < ctl.Pin)||(i >= 16)) DoCmnd(i);
        }
        //Обработка клавиш управления.
        private void Form1_KeyDown(object sender, KeyEventArgs e)
        {
            if (e.KeyCode == Keys.Delete)
                //Клавиша Delete.
                DoCmnd(18);
            if (e.KeyCode == Keys.Execute)
                //Клавиша Execute Separator.
                DoCmnd(19);
            if (e.KeyCode == Keys.Decimal)
                //Клавиша Decimal.
                DoCmnd(16);
        }
    }
}
```

Содержание отчета

1. Задание.

2. Текст программы.
3. Тестовые наборы данных для тестирования методов класса.

Контрольные вопросы

1. Назначения компонентов класса Button?
2. Назначения компонентов класса Label?
3. Назначения компонентов класса TextBox?
4. Назначения компонентов класса TrackBar?
5. Назначения компонентов класса numericUpDown?
6. Когда возникает событие Load?
7. Когда возникает событие Click?
8. Когда возникает событие Scroll?
9. Когда возникает событие ValueChanged?
10. Когда возникает событие KeyPress?
11. Когда возникает событие KeyDown?