ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет компьютерных наук
Департамент программной инженерии

СОГЛАСОВАНО
Профессор
факультета компьютерных наук
доктор физико-математических наук

_____ И.В. Аржанцев
«___» _____ 2017 г.

УТВЕРЖДАЮ
Академический руководитель
образовательной программы
«Программная инженерия»

_____ В.В. Шилов
«___» _____ 2017 г.

Инв. № подл	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата
RU.17701729.507140-01 12 01-1				

ПРОГРАММА КОДИРОВАНИЯ И ДЕКОДИРОВАНИЯ АЛГЕБРОГЕОМЕТРИЧЕСКИХ КОДОВ

Текст программы

ЛИСТ УТВЕРЖДЕНИЯ

RU.17701729.507140-01 12 01-1-ЛУ

Исполнитель:
студент группы БПИ162
_____ / Казанцева А.Р. /
«____»_____ 2017 г.

2017

УТВЕРЖДЕНО
RU.17701729.507140-01 12 01-1-ЛУ

ПРОГРАММА КОДИРОВАНИЯ И ДЕКОДИРОВАНИЯ АЛГЕБРОГЕОМЕТРИЧЕСКИХ КОДОВ

Текст программы

RU.17701729.507140-01 12 01-1

Листов 41

Инв. № подл	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата
RU.17701729.507140-01 12 01-1				

2016

Аннотация

В данном документе приведет текст «Программы кодирования и декодирования алгеброгеометрических кодов». Программа разработана на языке C#. Среда разработки - Microsoft Visual Studio Professional 2017.

Функциональным назначением программы является генерация и сохранение алгеброгеометрических кодов, а также кодирование и декодирование некоторых сообщений с их помощью.

Настоящий документ разработан в соответствии с требованиями:

1) ГОСТ 19.101-77 Виды программ и программных документов [1];

2) ГОСТ 19.102-77 Стадии разработки [2];

3) ГОСТ 19.103-77 Обозначения программ и программных документов [3];

4) ГОСТ 19.104-78 Основные надписи [4];

5) ГОСТ 19.105-78 Общие требования к программным документам [5];

6) ГОСТ 19.106-78 Требования к программным документам, выполненным печатным способом[6];

7) ГОСТ 19.401-78 Текст программы. Требования к содержанию и оформлению [7].

Изменения к данному документу оформляются согласно ГОСТ 19.603-78 [8], ГОСТ 19.604-78 [9].

Перед прочтением данного документа рекомендуется ознакомиться с терминологией, приведенной в Приложении 1 и 2.

СОДЕРЖАНИЕ

1 Текст программы

Программа состоит из библиотеки классов и десяти окон Windows Presentation Foundation. Далее будет приведен код, содержащийся в каждой из частей программы.

1.1 Код библиотеки Booler

1.1.1 Код класса Code

```
using System;
using Booler.Matrices;
using Booler.Exceptions;

namespace Booler
{
    /// <summary>
    /// Алгеброгеометрический код.
    /// </summary>
    [Serializable]
    public class Code
    {
        /// <summary>
        /// Порождающая матрица кода.
        /// </summary>
        SolutionsMatrix generatingMatrix;
        /// <summary>
        /// Матрица всех возможных кодовых слов.
        /// </summary>
        BaseMatrix allCodeWords;
        /// <summary>
        /// Максимальное количество исправляемых ошибок.
        /// </summary>
        int t;
        /// <summary>
        /// Система уравнений, порождающая данный код.
        /// </summary>
        string[] systemOfEquations;

        /// <summary>
        /// Порождающая матрица кода.
        /// </summary>
        public SolutionsMatrix GeneratingMatrix => generatingMatrix;
        /// <summary>
        /// Матрица всех возможных кодовых слов.
        /// </summary>
        public BaseMatrix AllCodeWords => allCodeWords;
        /// <summary>
        /// Длина кодируемого слова.
        /// </summary>
        public int K => generatingMatrix.Matrix[0].Length;
        /// <summary>
        /// Длина кодового слова.
        /// </summary>
        public int N => generatingMatrix.Matrix.Count;
        /// <summary>
```

```
        /// Максимальное количество исправляемых ошибок.
        /// </summary>
        public int T => t;
        /// <summary>
        /// Система уравнений, порождающая данный код.
        /// </summary>
        public string[] SystemOfEquations => systemOfEquations;

        /// <summary>
        /// Создает код на основе полученной матрицы, устонавливая его основные
параметры.
        /// </summary>
        /// <param name="generatingMatrix">Порождающая матрица</param>
        /// <param name="line">Система уравнений, порождающая код</param>
        public Code(SolutionsMatrix generatingMatrix, string[] line)
        {
            if (generatingMatrix.Matrix.Count <=generatingMatrix.Matrix[0].Length) throw
new CodeGeneratingException("Система имеет недостаточно решений для генерации кода.");
                this.generatingMatrix = generatingMatrix;
            systemOfEquations = line;
            allCodeWords = new BaseMatrix();
            for (int possibleCodeNumber = 0; possibleCodeNumber < Math.Pow(2, K);
possibleCodeNumber++)
            {
                int[] intArrayPossibleCodeNumber =
MyStatics.ToBinaryIntArray(possibleCodeNumber, K);
                BaseMatrix vectorPossibleCodeNumber = new BaseMatrix();
                for (int i = 0; i < K; i++)
                {
                    int[] nextLine = { intArrayPossibleCodeNumber[i] };
                    vectorPossibleCodeNumber.Matrix.Add(nextLine);
                }
                BaseMatrix vectorPossibleCode =
MyStatics.Multiplication(vectorPossibleCodeNumber, generatingMatrix);
                allCodeWords.Matrix.Add(vectorPossibleCode.Matrix[0]);
            }

            this.t = (MyStatics.FindMinDistance(AllCodeWords) - 1) / 2;
        }

        /// <summary>
        /// Кодирует полученное сообщение.
        /// </summary>
        /// <param name="ourMessage">Кодируемое слово</param>
        /// <returns>Код</returns>
        public int[] Encode(int[] ourMessage)
        {
            BaseMatrix A = new BaseMatrix(ourMessage);
            BaseMatrix C = MyStatics.Multiplication(A, generatingMatrix); ;
            return C.Matrix[0];
        }

        /// <summary>
        /// Декодирует полученное сообщение, исправляя ошибки.
        /// </summary>
        /// <param name="ourMessage">Декодируемое слово</param>
        /// <returns>Исходное сообщение</returns>
        public int[] Decode(int[] ourMessage)
        {
```

```
            int currentDistance = 0, minDistance = N + 1, number = 0;

            for (int i = 0; i < allCodeWords.Matrix.Count; i++)
            {
                for (int j = 0; j < allCodeWords.Matrix[i].Length; j++)
                    if (ourMessage[j] != allCodeWords.Matrix[i][j]) currentDistance++;
                if (currentDistance < minDistance)
                {
                    minDistance = currentDistance;
                    number = i;
                }
                currentDistance = 0;
                if (minDistance == 0) return MyStatics.ToBinaryIntArray(number, K);
            }
            if (minDistance > T)
                throw new MistakesNumberException("Количество ошибок, сделанных в кодовом
слове, превышено!");
            return MyStatics.ToBinaryIntArray(number, K);
        }

        /// <summary>
        /// Формирует имя кода в соответствии с основными характеристиками.
        /// </summary>
        /// <returns>Имя кода</returns>
        public override string ToString()
        {
            return $"Код[{K},{N},{T}]";
        }
    }
}
```

1.1.2 Код класса Equation

```
using System;
using System.Collections.Generic;
using Booler.Tokens;

namespace Booler
{
    /// <summary>
    /// Уравнение.
    /// </summary>
    public class Equation
    {
        /// <summary>
        /// Разбивает строку на токены, характерные для уравнений.
        /// </summary>
        Tokenizer tokenizer;
        /// <summary>
        /// Проверяет возможно ли разбиение строки на токены.
        /// </summary>
        Parser parser = new Parser();

        /// <summary>
        /// Максимальное количество переменных в уравнении.
        /// </summary>
        int groupSize;
        /// <summary>
```

```
        /// Уравнение в памяти.
        /// </summary>
        List<Token> tokens = null;
        /// <summary>
        /// Уравнение.
        /// </summary>
        Tuple<List<bool[]>, bool> eq = null;

        /// <summary>
        /// Уравнение.
        /// </summary>
        public Tuple<List<bool[]>, bool> Eq => eq;
        /// <summary>
        /// Максимальное количество переменных в уравнении.
        /// </summary>
        public int GroupSize => groupSize;
        /// <summary>
        /// Количество слагаемых в уравнении.
        /// </summary>
        public int VariableTokenCount => eq.Item1.Count;

        /// <summary>
        /// Строит в памяти уравнение из полученной строки.
        /// </summary>
        /// <param name="input">Строка, содержащая уравнение</param>
        /// <param name="groupSize">Максимальное количество переменных в этой
строке</param>
        /// <param name="index">Номер уравнения в системе.</param>
        public Equation(string input, int groupSize, int index)
        {
            this.groupSize = groupSize;
            tokenizer = new Tokenizer(groupSize);
            tokens = tokenizer.SplitToTokens(input, index);
            eq = parser.Parse(tokens, index);
        }
    }
}
```

1.1.3 Код пространства имен Exceptions

```
using System;

namespace Booler
{
    /// <summary>
    /// Исключения, характерные для создаваемых классов.
    /// </summary>
    namespace Exceptions
    {
        public class TokenizerException : ApplicationException
        {
            protected int index;
            public int Index => index;
            public TokenizerException(string message, int index) : base(message)
            {
                this.index = index;
```

```
            }
        }

        public class VariableGroupTokenException : TokenizerException
        {
            public VariableGroupTokenException(string message, int index) : base(message,
index) { }
        }

        public class UnexpectedSymbolException : TokenizerException
        {
            public UnexpectedSymbolException(string message, int index) : base(message,
index) { }
        }

        public class ParserException : ApplicationException
        {
            protected int index;
            public int Index => index;
            public ParserException(string message, int index) : base(message)
            {
                this.index = index;
            }
        }

        public class UnexpectedTokenException : ParserException
        {
            public UnexpectedTokenException(string message, int index) : base(message,
index) { }
        }

        public class UnknownCodeMessageException : ApplicationException
        {
            protected int index;
            public int Index => index;
            public UnknownCodeMessageException(string message, int index) : base(message)
            {
                this.index = index;
            }
        }

        public class CodeException : ApplicationException
        {
            public CodeException(string message) : base(message) { }
        }

        public class MistakesNumberException : CodeException
        {
            public MistakesNumberException(string message) : base(message) { }
        }

        public class CodeGeneratingException : CodeException
        {
            public CodeGeneratingException(string message) : base(message) { }
        }
    }
}
```

1.1.4 Код пространства имен Matrices

```
using System;
using System.Collections.Generic;

namespace Booler
{
    /// <summary>
    /// Матрицы, используемые для решения поставленной задачи.
    /// </summary>
    namespace Matrices
    {
        /// <summary>
        /// Матрица
        /// </summary>
        [Serializable]
        public class BaseMatrix
        {
            /// <summary>
            /// Матрица
            /// </summary>
            List<int[]> matrix;

            /// <summary>
            /// Матрица
            /// </summary>
            public List<int[]> Matrix => matrix;

            /// <summary>
            /// Обращается к элементу матрицы.
            /// </summary>
            /// <param name="i">Номер столбца</param>
            /// <param name="j">Номер строки</param>
            /// <returns></returns>
            public int this[int i, int j]
            {
                get
                {
                    if ((i < 0) || (i >= Matrix.Count) || (j < 0) || (j >=
Matrix[0].Length))
                        throw new ArgumentOutOfRangeException("Index of matrix's
coordinates", $"Use valid int");
                    return Matrix[i][j];
                }
            }

            /// <summary>
            /// Конструктор пустой матрицы.
            /// </summary>
            public BaseMatrix()
            {
                matrix = new List<int[]>();
            }

            public BaseMatrix(int[] ourLine) : this()
            {
                for (int i = 0; i < ourLine.Length; i++)
                {
                    int[] nextLine = { ourLine[i] };
```

```
                    matrix.Add(nextLine);
                }
            }

            /// <summary>
            ///
            /// </summary>
            /// <returns>Матрицу в строковом формате</returns>
            public override string ToString()
            {
                if (matrix.Count == 0)
                    throw new NullReferenceException("Matrix, that you have tried bring
to string, is empty!");

                string str = "";
                for (int i = 0; i < matrix[0].Length; str += "\n", i++)
                    for (int j = 0; j < matrix.Count; j++)
                        str += $" {matrix[j][i]}";

                return str;
            }
        }

        /// <summary>
        /// Матрица решений системы уравнений.
        /// </summary>
        [Serializable]
        public class SolutionsMatrix : BaseMatrix
        {
            /// <summary>
            /// Матрица решений системы уравнений.
            /// </summary>
            public SolutionsMatrix() : base() { }
        }
    }
}
```

1.1.5 Код класса MyStatics

```
using System;
using System.Collections.Generic;
using Booler.Matrices;

namespace Booler
{
    /// <summary>
    /// Статические методы, потребовавшиеся в ходе решения задачи.
    /// </summary>
    public static class MyStatics
    {
        /// <summary>
        /// Представляет число в двоичном виде.
        /// </summary>
        /// <param name="value">Число в десятичной системе</param>
        /// <param name="size">Размерность двоичного представления числа</param>
        /// <returns>Число в двоичной системе</returns>
        public static int[] ToBinaryIntArray(int value, int size)
        {
```

```
            int[] intBinaryCode = new int[size];
            string binaryCode = Convert.ToString(value, 2);

            if ((binaryCode.Length > size) || (value < 0))
                throw new ArgumentOutOfRangeException("Размерность двоичного
представления числа",
                    $"Размерность двоичного представления числа должна быть
0<=N<{Math.Pow(2, size)}.");

            // Заполнение массива ведущими нулями.
            for (int i = 0; i < size - binaryCode.Length; i++) intBinaryCode[i] = 0;
            for (int i = size - binaryCode.Length; i < size; i++) intBinaryCode[i] =
(int)binaryCode[i - size + binaryCode.Length] - 48;
            return intBinaryCode;
        }

        /// <summary>
        /// Перемножает матрицы.
        /// </summary>
        /// <param name="A">Первый множитель</param>
        /// <param name="B">Второй множитель</param>
        /// <returns></returns>
        public static BaseMatrix Multiplication(BaseMatrix A, BaseMatrix B)
        {
            if ((A.Matrix.Count == 0) || (B.Matrix.Count == 0) || (A == null) || (B ==
null))
                throw new NullReferenceException("You tried multiplicate empty matrix!");
            if (B.Matrix[0].Length != A.Matrix.Count)
                throw new ArgumentException("Matrices are incompatible for
multiplication!");

            BaseMatrix C = new BaseMatrix();

            for (int i = 0; i < A.Matrix[0].Length; i++)
            {
                int[] helper = new int[B.Matrix.Count];

                for (int j = 0; j < B.Matrix.Count; j++)
                {

                    for (int k = 0; k < A.Matrix.Count; k++)
                        helper[j] += A.Matrix[k][i] * B.Matrix[j][k];
                    helper[j] %= 2;
                }
                C.Matrix.Add(helper);
            }

            return C;
        }

        /// <summary>
        /// Преобразует строковое сообщение в массив
        /// </summary>
        /// <param name="input">Исходное сообщение</param>
        /// <param name="size">Размер сообщения</param>
        /// <returns>Исходное сообщение</returns>
        public static int[] ToIntArray(string input, int size)
        {
            int[] output = new int[size];
```

```
            for (int i = 0; i < input.Length; i++)
            {
                if ((input[i] != '0') && (input[i] != '1')) throw new
ArgumentException("Возможно использование только символов 0 и 1!");
                output[i] = input[i] - (int)'0';
            }
            for (int i = input.Length; i < size; i++) output[i] = 0;
            return output;
        }

        /// <summary>
        /// Ищет минимальное расстояние в коде.
        /// </summary>
        /// <param name="allCodeWords">Множество всех кодовых слов</param>
        /// <returns>Минимальное расстояние</returns>
        public static int FindMinDistance(BaseMatrix allCodeWords)
        {
            int minDistance = Int32.MaxValue, currentDistance = 0;
            for (int i = 0; i < allCodeWords.Matrix.Count; i++)
            {
                for (int j = i + 1; j < allCodeWords.Matrix.Count; j++)
                {
                    for (int k = 0; k < allCodeWords.Matrix[0].Length; k++)
                        currentDistance += (allCodeWords.Matrix[i][k] +
allCodeWords.Matrix[j][k]) % 2;
                    if (currentDistance < minDistance) minDistance = currentDistance;
                    currentDistance = 0;
                }
            }
            return minDistance;
        }

        /// <summary>
        /// Считывает уравнение и добавляет его в систему уравнениц.
        /// </summary>
        /// <param name="input">Строковое представление системы  уравнений</param>
        /// <param name="groupSize">Количество переменных всистеме уравнений</param>
        /// <param name="systemOfEquations">Система уравнений</param>
        /// <param name="index">Порядковый номер уравнения в системе</param>
        public static void Reading(string input, int groupSize, List<Equation>
systemOfEquations, int index)
        {
            if (input != "")
                systemOfEquations.Add(new Equation(input, groupSize, index));
        }
    }
}
```

1.1.6 Код класса Parser

```
using System;
using System.Collections.Generic;
using Booler.Tokens;
using Booler.Exceptions;

namespace Booler
{
```

```
    /// <summary>
    /// Проверяет корректность следования токенов друг за другом и делит уравнение на 2
части: зависящую от переменных и свободную.
    /// </summary>
    public class Parser
    {
        // Константы, обозначающие тип предыдущего токена
        const int START = 0;
        const int AFTER_VARIABLEGROUP = 1;
        const int AFTER_OPERATION = 2;
        const int AFTER_EQUATION = 3;
        const int AFTER_SCALAR = 4;

        /// <summary>
        /// Проверка правильности порядка токенов и преобразование их в уравнение
        /// </summary>
        /// <param name="tokens">Список токенов</param>
        /// <param name="index">Номер уравнения в системе</param>
        /// <returns>Уравнение</returns>
        public Tuple<List<bool[]>, bool> Parse(List<Token> tokens, int index)
        {
            // Представление в памяти части уравнеия, содержащей переменные.
            List<bool[]> eq = new List<bool[]>();
            // Скаляр
            bool scalar=false;
            // Проверка на наличие в уравнении знака равенства и переменных.
            bool flag1 = false, flag2 = false;

            int state = START;

            foreach(Token token in tokens)
            {
                if(token.GetType() == typeof(VariableGroupToken))
                {
                    flag1 = true;
                    if (state != AFTER_OPERATION && state != START && state !=
AFTER_EQUATION)
                        throw new UnexpectedTokenException("Группа переменных может
стоять только в начале, после знака равенства или сложения",  index);

                    eq.Add(((VariableGroupToken)token).variables);
                    state = AFTER_VARIABLEGROUP;
                }
                else if (token.GetType() == typeof(OperationToken))
                {
                    if (state != AFTER_VARIABLEGROUP && state != AFTER_SCALAR)
                        throw new UnexpectedTokenException("Знак сложения может стоять
только после группы переменных или скаляра", index);

                    state = AFTER_OPERATION;
                }
                else if (token.GetType() == typeof(EquationToken))
                {
                    if (flag2) throw new ParserException("В уравнении допустимо
исспользовать только один знак равенства", index);
                    flag2 = true;
                    if (state != AFTER_VARIABLEGROUP && state != AFTER_SCALAR)
                        throw new UnexpectedTokenException("Знак равенства может стоять
только после скаляра или группы переменных", index);
```

```
                    state = AFTER_EQUATION;
                }
                else if (token.GetType() == typeof(ScalarToken))
                {
                    if (state != AFTER_EQUATION  && state != START && state !=
AFTER_OPERATION)
                        throw new UnexpectedTokenException("Скаляр может стоять только в
начале уравнения, после знака равенства или сложения", index);

                    scalar = !(((ScalarToken)token).variable.Equals(scalar));
                    state = AFTER_SCALAR;
                } else
                {
                    throw new UnexpectedTokenException($"Не ожидался токен
{token.GetType()}", index);
                }
            }

            if (state == AFTER_EQUATION)
                throw new ParserException("Уравнение не должно заканчиваться знаком
равенства!", index);
            if (!(flag1 && flag2)) throw new TokenizerException("Уравнение неполное!",
index);

            return new Tuple<List<bool[]>, bool>(eq, scalar);
        }
    }
}
```

1.1.7 Код класса Solver

```
using System;
using System.Collections.Generic;
using Booler.Matrices;

namespace Booler
{
    /// <summary>
    /// Решает систкму уравнений.
    /// </summary>
    public class Solver
    {
       /// <summary>
       /// Матрица решений.
       /// </summary>
        SolutionsMatrix matrix;
        /// <summary>
        /// Проверочный лист.
        /// </summary>
        CheckList checkingList;

        /// <summary>
        /// Решает систему уравнений.
        /// </summary>
        /// <param name="equationsSystem">Система уравнений</param>
        /// <returns>Решение системы уравнений</returns>
```

```
        public SolutionsMatrix Solve(List<Equation> equationsSystem)
        {
            if (equationsSystem == null)
                throw new NullReferenceException("Система уравнений пуста!");
            if (equationsSystem.Count==0)
                throw new NullReferenceException("Введите систему уравнений!");

            matrix = new SolutionsMatrix();

            // Перебор всех возможных решений.
            for (int possibleSolution=0; possibleSolution<Math.Pow(2,
equationsSystem[0].GroupSize); possibleSolution++)
            {
                // Представление номера возможного решения в двоичном виде.
               int[] intArrayPossibleSolution =
MyStatics.ToBinaryIntArray(possibleSolution, equationsSystem[0].GroupSize);

                // Создание проверочного листа - элемента, помогающего определить
принадлежит ли решение множеству решений данной системы.
                checkingList = new CheckList(equationsSystem);

                // Заполнение проверочного листа возможным решением.
                checkingList.FillingTheCheckList(intArrayPossibleSolution);

                /// Сверка значений полученного в результате подстановки решения в
проверочный лист и значения,
                /// которое должно быть по условию.
                /// В случае полного совпадения решение, которое подставлялось в
проверочный лист?
                /// добавляется в матрицу решений системы уравнений.
                if (checkingList.IsItRightSolution())
matrix.Matrix.Add(intArrayPossibleSolution);
            }
            return matrix;
        }
    }
}
```

1.1.8 Код класса Tokenizer

```
using System;
using Booler.Tokens;
using Booler.Exceptions;
using System.Collections.Generic;

namespace Booler
{
    /// <summary>
    /// Производит деление входной строки на токены, если это возможно.
    /// </summary>
    public class Tokenizer
    {
        /// <summary>
        /// Количество различных переменных в системе уравнений.
        /// </summary>
        public readonly int group_size;
```

```
        public Tokenizer(int group_size)
        {
            this.group_size = group_size;
        }

        /// <summary>
        /// Разбивает строку на токены, характерные для уравнений.
        /// </summary>
        /// <param name="input">Исходная строка, поданная пользователем. </param>
        ///  <param name="mainIndex">Номер уравнения всистеме.</param>
        /// <returns>Список токенов.</returns>
        public List<Token> SplitToTokens(string input, int mainIndex)
        {
            List<Token> tokens = new List<Token>();

            for (int i = 0; i < input.Length;)
            {
                switch (input[i])
                {
                    case ' ':
                        i++;
                        break;
                    case 'x':
                        bool[] vars = new bool[group_size];
                        while (true)
                        {
                            if (i >= input.Length || input[i] != 'x')
                                break;

                            int j;
                            for (j = i + 1; j < input.Length && Char.IsDigit(input[j]);
j++) ;
                            if (j == (i + 1))
                            {
                                throw new VariableGroupTokenException("Ожидался индекс
при переменной.", mainIndex);
                            }

                            int index;
                            int.TryParse(input.Substring(i + 1, j - i - 1), out index);
                            index--;

                            if (index < 0 || index >= group_size)
                            {
                                throw new VariableGroupTokenException("Индекс выходит за
границы допустимых значений.", mainIndex);
                            }

                            vars[index] = true;

                            for (; j < input.Length && input[j] == ' '; j++) ;
                            i = j;
                        }

                        tokens.Add(new VariableGroupToken(vars));
                        break;
                    case '+':
                        tokens.Add(new OperationToken());
                        i++;
```

```
                    break;
                case '=':
                    i++;
                    tokens.Add(new EquationToken());
                    break;
                case '0':
                    i++;
                    tokens.Add(new ScalarToken(false));
                    break;
                case '1':
                    i++;
                    tokens.Add(new ScalarToken(true));
                    break;
                default:
                    throw new UnexpectedSymbolException($"Неизвестный символ
'{input[i]}'", mainIndex);
                    break;
                }
            }
            return tokens;
        }
    }
}
```

1.1.9 Код класса CheckList

```
using System;
using System.Collections.Generic;

namespace Booler
{
   /// <summary>
   /// Проверочный лист.
   /// </summary>
   public class CheckList
    {
        /// <summary>
        /// Результат подставления в систему уравнений конкретного решения.
        /// </summary>
        bool[] checkingList;
        /// <summary>
        /// Система уравнений, которую будут проверять данным проверочным листом.
        /// </summary>
        List<Equation> prototype=new List<Equation>();

        /// <summary>
        /// Возвращает значение уравнения с заданным индексом для заданной системы
уравнений и заданного решения.
        /// </summary>
        /// <param name="i">Номер уравнения</param>
        /// <returns>Результат подтавления в уравнение некоторого решения</returns>
        public bool this[int i]
        {
            get
            {
                if ((i < 0) || (i >= checkingList.Length))
```

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.502610-01 12				
Инв. № подл.	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата

```
                    throw new ArgumentOutOfRangeException("Индекс в проверочном листе",
$"Индекс должен быть меньше {checkingList.Length} и больше или равен 0!");

                return checkingList[i];
            }
        }

        /// <summary>
        /// Создание проверочного листа на основе системы уравнений.
        /// </summary>
        /// <param name="prototype">Система уравнений, которую будут проверять данным
проверочным листом</param>
        public CheckList(List<Equation> prototype)
        {
            if (prototype == null)
                throw new NullReferenceException("В проверочный лист в качестве прототипа
подана пустая система уравнений!");

            this.prototype = prototype;

            checkingList = new bool[prototype.Count];
        }

        /// <summary>
        /// Заполняет проверочный лист для конкретного решения.
        /// </summary>
        /// <param name="possibleSolution"></param>
        public void FillingTheCheckList(int[] possibleSolution)
        {
            // Просматриваем каждое уравнение.
            for (int equationNumber = 0; equationNumber < checkingList.Length;
equationNumber++)
            {
                int firstHelper = 0;

                // Отдельно смотрим каждую группу переменных.
                for (int variableGroupNumber=0;
variableGroupNumber<prototype[equationNumber].VariableTokenCount; variableGroupNumber++)
                {
                    int secondHelper = 1;

                    // Смотрим на наличие переменной в данной группе переменных.
                    for (int variableNumber = 0; variableNumber <
prototype[equationNumber].GroupSize; variableNumber++)
                    {
                        if
(prototype[equationNumber].Eq.Item1[variableGroupNumber][variableNumber])
                        {
                            secondHelper *= possibleSolution[variableNumber];
                            if (possibleSolution[variableNumber] == 0) break;
                        }

                    }
                    firstHelper = (firstHelper + secondHelper)%2;
                }
                checkingList[equationNumber] = (firstHelper == 1);
            }
        }
```

```
        /// <summary>
        /// Проверяет, совпадает ли правильный ответ к системе с полученным в проверочном
листе.
        /// </summary>
        /// <returns></returns>
        public bool IsItRightSolution()
        {
            if (this.checkingList == null)
                throw new NullReferenceException("Проверочный лист пуст!");
            if (this.prototype == null)
                throw new NullReferenceException("Матрица пуста!");

            for (int i = 0; i <checkingList.Length; i++)
            {
                if (checkingList[i] != prototype[i].Eq.Item2) return false;
            }
            return true;
        }
    }
}
```

1.1.10 Код пространства имен Tokens

```
namespace Booler
{
    /// <summary>
    ///  Пространство, содержащее в себе все виды блоков(токенов), которые могут
содержаться в исходных уравнениях.
    /// </summary>
    namespace Tokens
    {
        /// <summary>
        /// Общий вид искомых токенов.
        /// </summary>
        public class Token
        {
            public override string ToString()
            {
                return "Token.(Empty)";
            }
        }

        /// <summary>
        /// Токен-группа переменных, с промежуточным знаком умножения.
        /// </summary>
        class VariableGroupToken : Token
        {
            /// <summary>
            /// Массив, показывающий наличие или отсутствие каждой возможной для системы
уравнений переменной в текущей группе.
            /// </summary>
            public readonly bool[] variables;
            /// <summary>
            /// Количество переменных в текущей группе.
            /// </summary>
            public readonly int size;
```

```
        /// <summary>
        /// Конструктор токена-группы переменных.
        /// </summary>
        /// <param name="variables"> Массив, показывающий наличие или отсутствие
каждой возможной для системы уравнений переменной в текущей группе.</param>
        public VariableGroupToken(bool[] variables)
        {
            this.variables = variables;
            size = 0;
            for (int i = 0; i < variables.Length; i++)
                if(variables[i])
                    size++;
        }
    }

    /// <summary>
    /// Токен-операция сложения.
    /// </summary>
    class OperationToken : Token
    {
        public override string ToString()
        {
            return "Tokens.Operation: +";
        }
    }

    /// <summary>
    /// Токен-знак равенства.
    /// </summary>
    class EquationToken : Token
    {
        public override string ToString()
        {
            return "Tokens.Equation: =";
        }
    }

    /// <summary>
    /// Токен-свободный член.
    /// </summary>
    class ScalarToken : Token
    {
        /// <summary>
        /// Значение свободного члена.
        /// </summary>
        public readonly bool variable;

        /// <summary>
        /// Конструктор токена-свободного члена.
        /// </summary>
        /// <param name="variable">Значение свободного члена</param>
        public ScalarToken(bool variable)
        {
            this.variable = variable;
        }

        public override string ToString()
        {
            return "Tokens.Scalar: " + variable.ToString();
```

```
            }
        }
    }
}
```

1.2 Код окон WPF

1.2.1 Код окна MainWindow

```
using System.Windows;

namespace Генератор_алгеброгеометрических_кодов
{
    /// <summary>
    /// Логика взаимодействия для MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        /// <summary>
        /// Точка вхада в программу.
        /// Конструктор главного окна MainWindow.
        /// </summary>
        public MainWindow()
        {
            InitializeComponent();
        }

        /// <summary>
        /// Начало работы. Переход к выбору алгеброгеометрического кода.
        /// </summary>
        /// <param name="sender">MainWindow.</param>
        /// <param name="e">RoutedEventArgs.</param>
        private void startButton_Click(object sender, RoutedEventArgs e)
        {
            SelectCodeWindow win = new SelectCodeWindow();
            win.Top = this.Top;
            win.Left = this.Left;
            win.Show();
            this.Close();
        }

        /// <summary>
        /// Переход к руководству оператора.
        /// </summary>
        /// <param name="sender">MainWindow</param>
        /// <param name="e">RoutedEventArgs</param>
        private void fqButton_Click(object sender, RoutedEventArgs e)
        {
            FQWindow win = new FQWindow();
            win.Top = this.Top;
            win.Left = this.Left;
            win.Show();
            this.Close();
        }

        /// <summary>
        /// Переход к информаци о разработчике програмыю
        /// </summary>
```

```
        /// <param name="sender">MainWindow</param>
        /// <param name="e">RoutedEventArgs</param>
        private void infoButton_Click(object sender, RoutedEventArgs e)
        {
            InfoWindow win = new InfoWindow();
            win.Top = this.Top;
            win.Left = this.Left;
            win.Show();
            this.Close();

        }

    }
}
```

1.2.2 Код окна FQWindow

```
using System.Windows;

namespace Генератор_алгеброгеометрических_кодов
{
    /// <summary>
    /// Логика взаимодействия для FQWindow.xaml
    /// </summary>
    public partial class FQWindow : Window
    {
        /// <summary>
        /// Конструктор окна полной справки
        /// </summary>
        public FQWindow()
        {
            InitializeComponent();
        }

        /// <summary>
        /// Возвращает к предыдущему окну.
        /// </summary>
        /// <param name="sender">FQWindow</param>
        /// <param name="e">RoutedEventArgs</param>
        private void PreviousWindowButton_Click(object sender, RoutedEventArgs e)
        {
            MainWindow win = new MainWindow()
            {
                Top = this.Top,
                Left = this.Left
            };
            win.Show();
            this.Close();
        }
    }
}
```

1.2.3 Код окна InfoWindow

```
using System.Windows;

namespace Генератор_алгеброгеометрических_кодов
{
```

```
    /// <summary>
    /// Логика взаимодействия для InfoWindow.xaml
    /// </summary>
    public partial class InfoWindow : Window
    {
        /// <summary>
        /// Конструктор окнаинформации о разработчике.
        /// </summary>
        public InfoWindow()
        {
            InitializeComponent();
        }

        /// <summary>
        /// Возврат к главному окну.
        /// </summary>
        /// <param name="sender">InfoWindow</param>
        /// <param name="e">RoutedEventArgs</param>
        private void previousWindowButton_Click(object sender, RoutedEventArgs e)
        {
            MainWindow win = new MainWindow();
            win.Top = this.Top;
            win.Left = this.Left;
            win.Show();
            this.Close();
        }
    }
}
```

1.2.4 Код окна SelectCodeWindow

```
using System.Collections.Generic;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Media;
using System.Runtime.Serialization.Formatters.Binary;
using Booler;
using System.IO;
using System;

namespace Генератор_алгеброгеометрических_кодов
{
    /// <summary>
    /// Логика взаимодействия для SelectCodeWindow.xaml
    /// </summary>
    public partial class SelectCodeWindow : Window
    {
        /// <summary>
        /// Инструмент бинарной сериализации.
        /// </summary>
        BinaryFormatter formatter = new BinaryFormatter();
        /// <summary>
        /// Список сохраненных кодов.
        /// </summary>
        List<Code> codes = new List<Code>();
        /// <summary>
        /// Имя выбранного элемента codeSelector.
        /// </summary>
```

```
        string itemName;

        /// <summary>
        /// Канструктор окна SelectCodeWindow.
        /// </summary>
        public SelectCodeWindow()
        {
            InitializeComponent();
            try
            {
                using (FileStream fs = new FileStream("SavedCodes.nk", FileMode.Open))
                {
                    while (fs.Position < fs.Length)
                    {
                        Code newCode = (Code)formatter.Deserialize(fs);
                        codes.Add(newCode);
                    }
                }
            }
            catch (Exception)
            {

            }
            CreateComboBox();
        }

        /// <summary>
        /// Открывает предыдущее окно.
        /// </summary>
        /// <param name="sender">SelectCodeWindow</param>
        /// <param name="e">RoutedEventArgs</param>
        private void previousWindowButton_Click(object sender, RoutedEventArgs e)
        {
            MainWindow win = new MainWindow();
            win.Top = this.Top;
            win.Left = this.Left;
            win.Show();
            this.Close();
        }

        /// <summary>
        /// Готовит код к возможному удалению.
        /// </summary>
        /// <param name="sender">SelectCodeWindow</param>
        /// <param name="e">RoutedEventArgs</param>
        private void trashButton_Click(object sender, RoutedEventArgs e)
        {
            if ((codeSelector.SelectedIndex >= 0) && (codeSelector.SelectedIndex <
codes.Count))
            {
                ChoiceWindow win2 = new ChoiceWindow(itemName,
codeSelector.SelectedIndex, codes, this);
                win2.Top = this.Top+130;
                win2.Left = this.Left+180;
                win2.ShowDialog();
            }
        }

        /// <summary>
```

```
        /// Переходит к окну с характеристиками выбранного кода или к окну для создания
нового кода.
        /// </summary>
        /// <param name="sender">SelectCodeWindow</param>
        /// <param name="e">RoutedEventArgs</param>
        private void nextWindowButton_Click(object sender, RoutedEventArgs e)
        {

            if ((codeSelector.SelectedIndex >= 0) && (codeSelector.SelectedIndex <
codes.Count))
            {
                CodeDescriptionWindow win2 = new
CodeDescriptionWindow(codes[codeSelector.SelectedIndex]);
                win2.Top = this.Top;
                win2.Left = this.Left;
                win2.Show();
                this.Close();
            }
            else
            if (itemName == "Создать новый код...")
            {
                CodeGeneratingWindow win2 = new CodeGeneratingWindow();
                win2.Top = this.Top;
                win2.Left = this.Left;
                win2.Show();
                this.Close();
            }
            else
            {
                codeSelector.IsDropDownOpen = true;
            }
        }

        /// <summary>
        /// Переписывает текст codeSelector.
        /// </summary>
        public void CreateComboBox()
        {

            ComboBoxItem newItem = new ComboBoxItem();
            codeSelector.Items.Clear();
            for (int i = 0; i < codes.Count; i++)
            {
                newItem = new ComboBoxItem();
                newItem.Content = codes[i].ToString();
                newItem.Height = 22;
                newItem.Width = 191;
                newItem.FontSize = 13;
                newItem.BorderBrush = null;
                newItem.Foreground = new SolidColorBrush(Color.FromArgb(255, 12, 26,
62));
                newItem.Background = new SolidColorBrush(Color.FromArgb(255, 255, 255,
255));
                newItem.Margin = new Thickness(0, 0, 0, 0);
                codeSelector.Items.Add(newItem);

            }

            newItem = new ComboBoxItem();
```

```
            newItem.Content = "Создать новый код...";
            newItem.Height = 22;
            newItem.Width = 191;
            newItem.FontSize = 13;
            newItem.BorderBrush = null;
            newItem.Foreground = new SolidColorBrush(Color.FromArgb(255, 12, 26, 62));
            newItem.Background = new SolidColorBrush(Color.FromArgb(255, 255, 255, 255));
            newItem.Margin = new Thickness(0, 0, 6, 0);

            codeSelector.Items.Add(newItem);
        }

        /// <summary>
        /// Отмечает, какой код выбран, или открывает окно для создания нового кода.
        /// </summary>
        /// <param name="sender">SelectCodeWindow</param>
        /// <param name="e">SelectionChangedEventArgs</param>
        private void codeSelectorComboBox_SelectionChanged(object sender,
SelectionChangedEventArgs e)
        {
            ComboBox comboBox = (ComboBox)sender;
            ComboBoxItem selectedItem = (ComboBoxItem)comboBox.SelectedItem;

            if (selectedItem!=null) itemName = selectedItem.Content.ToString();

            if (itemName == "Создать новый код...")
            {
                CodeGeneratingWindow win2 = new CodeGeneratingWindow();
                win2.Top = this.Top;
                win2.Left = this.Left;
                win2.Show();
                this.Close();
            }
        }
    }
}
```

1.2.5 Код окна ChoiceWindow

```
using System.Collections.Generic;
using System.Windows;
using Booler;
using System.IO;
using System.Runtime.Serialization.Formatters.Binary;


namespace Генератор_алгеброгеометрических_кодов
{
    /// <summary>
    /// Логика взаимодействия для ChoiceWindow.xaml
    /// </summary>
    public partial class ChoiceWindow : Window
    {
        /// <summary>
        /// Инструмент для бинарной сериализации.
        /// </summary>
        BinaryFormatter formatter = new BinaryFormatter();
        /// <summary>
```

```
        /// Номер кода в списке сохраненных.
        /// </summary>
        int index;
        /// <summary>
        /// Выбранный код.
        /// </summary>
        List<Code> codes;
        /// <summary>
        /// Окно, вызвавшее конструктор ChoiceWindow.
        /// </summary>
        SelectCodeWindow win;

        /// <summary>
        /// Конструктор окна ChoiceWindow.
        /// </summary>
        /// <param name="name">Название кода</param>
        /// <param name="index">Номер кода в списке сохраненных</param>
        /// <param name="codes">Выбранный код</param>
        /// <param name="win">Окно, вызвавшее конструктор ChoiceWindow</param>
        public ChoiceWindow(string name, int index, List<Code> codes, SelectCodeWindow
win)
        {
            InitializeComponent();

            label.Content = "Вы уверены, что хотите \nудалить " + name + "?";
            this.index = index;
            this.codes = codes;
            this.win = win;
        }

        /// <summary>
        /// Удаляет выбранный код.
        /// </summary>
        /// <param name="sender">ChoiceWindow</param>
        /// <param name="e">RoutedEventArgs</param>
        private void YesButton_Click(object sender, RoutedEventArgs e)
        {
            if ((index >= 0) && (index < codes.Count))
            {
                using (FileStream fs = new FileStream("SavedCodes.nk", FileMode.Create))
                {
                    codes.Remove(codes[index]);
                    foreach (Code c in codes)
                        formatter.Serialize(fs, c);
                }
                win.CreateComboBox();
            }
            this.Close();
        }

        /// <summary>
        /// Закрывает окно.
        /// </summary>
        /// <param name="sender">ChoiceWindow</param>
        /// <param name="e">RoutedEventArgs</param>
        private void NoButton_Click(object sender, RoutedEventArgs e)
        {
            this.Close();
        }
```

```
    }
}
```

1.2.6 Код окна CodeGeneratingWindow

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using Booler;
using Booler.Exceptions;
using Booler.Matrices;
using System.Text.RegularExpressions;

namespace Генератор_алгеброгеометрических_кодов
{
    /// <summary>
    /// Логика взаимодействия для CodeGeneratingWindow.xaml
    /// </summary>
    public partial class CodeGeneratingWindow : Window
    {
        /// <summary>
        /// Количество переменных в системе уравнений.
        /// </summary>
        int groupSize = 0;
        /// <summary>
        /// Строковое представление системы уравнений.
        /// </summary>
        string line;
        /// <summary>
        /// Строковое представление уравнений.
        /// </summary>
        string[] lines;
        /// <summary>
        /// Система уравнений.
        /// </summary>
        List<Equation> systemOfEquations = new List<Equation>();
        /// <summary>
        /// Инструмент для решения системы уравнений.
        /// </summary>
        Solver solver = new Solver();
        /// <summary>
        /// Матрица из решений системы уравнений.
        /// </summary>
        SolutionsMatrix theAnswer;
        /// <summary>
        /// Алгеброгеометрический код.
        /// </summary>
        Code AGCode;
        /// <summary>
        /// Индикатор наличия ошибки в системе уравнений.
        /// </summary>
        bool flag = true;
```

```
        /// <summary>
        /// Допустимые для ввода символы.
        /// </summary>
        private Regex regex = new Regex("[^x^0-9\\+\\=]+");
        /// <summary>
        /// Подсказка для ползователя.
        /// </summary>
        ToolTip t;


        /// <summary>
        /// Количество переменных в системе уравнений.
        /// </summary>
        public int GroupSize => groupSize;

        /// <summary>
        /// Конструктор окна CodeGeneratingWindow.
        /// </summary>
        public CodeGeneratingWindow()
        {
            InitializeComponent();
            t = new ToolTip();
            equations.ToolTip = t;
        }

        /// <summary>
        /// Заполняет поле groupSize.
        /// </summary>
        /// <param name="sender">CodeGeneratingWindow</param>
        /// <param name="e">SelectionChangedEventArgs</param>
        private void SizeComboBox_SelectionChanged(object sender,
SelectionChangedEventArgs e)
        {
            ComboBox comboBox = (ComboBox)sender;
            ComboBoxItem selectedItem = (ComboBoxItem)comboBox.SelectedItem;

            groupSize = int.Parse(selectedItem.Content.ToString());
        }

        /// <summary>
        /// Генерирует алгеброгеометрический код.
        /// </summary>
        /// <param name="sender">CodeGeneratingWindow</param>
        /// <param name="e">RoutedEventArgs</param>
        private void NextWindowButton_Click(object sender, RoutedEventArgs e)
        {
            try
            {
                t.Content = "";
                if (groupSize == 0) throw new ArgumentNullException("количество
переменных", "Невведено количество переменных!");

                for (int i = 0; i < equations.Document.Blocks.Count; i++)
                {
                    equations.Document.Blocks.ElementAt<Block>(i).Foreground = new
SolidColorBrush(Color.FromArgb(255, 0, 0, 0));
                }

                #region Формирование системы уравнений в памяти
```

```
                line = new TextRange(equations.Document.ContentStart,
equations.Document.ContentEnd).Text;
                lines = line.Split('\n');

                for (int i = 0; i < lines.Length; i++)
                {
                    try
                    {
                        if (i != lines.Length - 1) lines[i] =
lines[i].Remove(lines[i].Length - 1, 1);
                        MyStatics.Reading(lines[i], groupSize, systemOfEquations, i);
                    }
                    catch (TokenizerException ex)
                    {
                        t.Content += (i+1) +" уравнение: "+ ex.Message + "\n";
                        equations.Document.Blocks.ElementAt<Block>(ex.Index).Foreground =
new SolidColorBrush(Color.FromArgb(255, 200, 21, 21));
                        flag = false;
                    }
                    catch (ParserException ex)
                    {
                        t.Content += (i + 1) + " уравнение: " + ex.Message + "\n";
                        equations.Document.Blocks.ElementAt<Block>(ex.Index).Foreground =
new SolidColorBrush(Color.FromArgb(255, 200, 21, 21));
                        flag = false;

                    }
                    catch (UnknownCodeMessageException ex)
                    {
                        t.Content += (i + 1) + " уравнение: " + ex.Message + "\n";
                        equations.Document.Blocks.ElementAt<Block>(ex.Index).Foreground =
new SolidColorBrush(Color.FromArgb(255, 200, 21, 21));
                        flag = false;

                    }
                }
                #endregion

                if (flag)
                {
                    #region Решение системы уравнений
                    theAnswer = solver.Solve(systemOfEquations);
                    if (theAnswer.Matrix.Count == 0) throw new
NullReferenceException("Система уравнений не имеет решений!");
                    #endregion

                    #region Формирование кода и вывод на экран его основных параметров
                    AGCode = new Code(theAnswer, lines);
                    CodeDescriptionWindow win = new CodeDescriptionWindow(AGCode);
                    win.Top = this.Top;
                    win.Left = this.Left;
                    win.Show();
                    this.Close();
                    #endregion
                }
                flag = true;
            }
            catch (CodeGeneratingException ex)
            {
```

```
                t.Content = ex.Message;
                foreach (Block block in equations.Document.Blocks)
                    block.Foreground = new SolidColorBrush(Color.FromArgb(255, 200, 21,
21));

            }
            catch (ArgumentNullException)
            {
                size.IsDropDownOpen = true;
            }
            catch (NullReferenceException ex)
            {
                t.Content += ex.Message;
                foreach (Block block in equations.Document.Blocks)
                    block.Foreground = new SolidColorBrush(Color.FromArgb(255, 200, 21,
21));

            }
            catch (TokenizerException ex)
            {
                t.Content += ex.Message + "\n";
                equations.Document.Blocks.ElementAt(ex.Index).Foreground = new
SolidColorBrush(Color.FromArgb(255, 200, 21, 21));
            }
            catch (ParserException ex)
            {
                t.Content += ex.Message + "\n";
                equations.Document.Blocks.ElementAt(ex.Index).Foreground = new
SolidColorBrush(Color.FromArgb(255, 200, 21, 21));
            }
            catch (UnknownCodeMessageException ex)
            {
                t.Content += ex.Message + "\n";
                equations.Document.Blocks.ElementAt(ex.Index).Foreground = new
SolidColorBrush(Color.FromArgb(255, 200, 21, 21));
            }
        }

        /// <summary>
        /// Возвращает на одно окно назад.
        /// </summary>
        /// <param name="sender">CodeGeneratingWindow</param>
        /// <param name="e">RoutedEventArgs</param>
        private void PreviousWindowButton_Click(object sender, RoutedEventArgs e)
        {
            SelectCodeWindow win = new SelectCodeWindow();
            win.Top = this.Top;
            win.Left = this.Left;
            win.Show();
            this.Close();
        }

        /// <summary>
        /// Вызывает справку.
        /// </summary>
        /// <param name="sender">CodeGeneratingWindow</param>
        /// <param name="e">RoutedEventArgs</param>
        private void HelpButton_Click(object sender, RoutedEventArgs e)
        {
```

```
            RulesWindow win = new RulesWindow();
            win.Top = this.Top + 50;
            win.Left = this.Left + 50;
            win.ShowDialog();
        }

        /// <summary>
        /// Проверяют допустимость символа, который вводится.
        /// </summary>
        /// <param name="sender">CodeGeneratingWindow</param>
        /// <param name="e">TextCompositionEventArgs</param>
        private void Equations_PreviewTextInput(object sender, TextCompositionEventArgs
e)
        {
            e.Handled = regex.IsMatch(e.Text);
        }
    }
}
```

1.2.7 Код окна RulesWindow

```
using System.Windows;

namespace Генератор_алгеброгеометрических_кодов
{
    /// <summary>
    /// Логика взаимодействия для RulesWindow.xaml
    /// </summary>
    public partial class RulesWindow : Window
    {
        /// <summary>
        /// Конструктор окна RulesWindow.
        /// </summary>
        public RulesWindow()
        {
            InitializeComponent();

        }
    }
}
```

1.2.8 Код окна CodeDescriptionWindow

```
using System.Windows;
using System.Windows.Documents;
using Booler;
using System.IO;
using System.Runtime.Serialization.Formatters.Binary;
using System;

namespace Генератор_алгеброгеометрических_кодов
{
    /// <summary>
    /// Логика взаимодействия для CodeDescriptionWindow.xaml
    /// </summary>
    public partial class CodeDescriptionWindow : Window
    {
        /// <summary>
```

```
        /// Выбранный код.
        /// </summary>
        Code AGCode;
        /// <summary>
        /// Инструмент для бинарной сериализации.
        /// </summary>
        BinaryFormatter formatter = new BinaryFormatter();

        /// <summary>
        /// Конструктор окна CodeDescriptionWindow.
        /// </summary>
        /// <param name="AGCode">Выбранный код</param>
        public CodeDescriptionWindow(Code AGCode)
        {
            InitializeComponent();
            this.AGCode = AGCode;
            infoTextBox.Width = infoTextBox.FontSize * AGCode.N * 1.25;
            infoTextBox.Text = "Длина кодируемых слов - " + AGCode.K;
            infoTextBox.Text += "\nДлина кодовых слов - " + AGCode.N;
            infoTextBox.Text += "\nМаксимальное количество ошибок - " + AGCode.T;
            infoTextBox.Text += "\n\nПорождающая матрица:\n";
            infoTextBox.Text += (AGCode.GeneratingMatrix.ToString());
        }

        /// <summary>
        /// Возвращается к предыдущему окну.
        /// </summary>
        /// <param name="sender">CodeDescriptionWindow</param>
        /// <param name="e">RoutedEventArgs</param>
        private void PreviousWindowButton_Click(object sender, RoutedEventArgs e)
        {
            CodeGeneratingWindow win = new CodeGeneratingWindow();
            win.Top = this.Top;
            win.Left = this.Left;
            win.equations.Document.Blocks.Clear();
            for (int i = 0; i < AGCode.SystemOfEquations.Length; i++)
                win.equations.Document.Blocks.Add(new Paragraph(new
Run(AGCode.SystemOfEquations[i])));
            win.size.SelectedIndex = AGCode.K - 1;
            win.Show();
            this.Close();
        }

        /// <summary>
        /// Открывает следующее окно.
        /// </summary>
        /// <param name="sender">CodeDescriptionWindow</param>
        /// <param name="e">RoutedEventArgs</param>
        private void NextWindowButton_Click(object sender, RoutedEventArgs e)
        {
            CodeWindow win = new CodeWindow(AGCode);
            win.Top = this.Top;
            win.Left = this.Left;
            win.Show();
            this.Close();
        }

        /// <summary>
        /// Выполняет сохранение полученного кода.
```

```
        /// </summary>
        /// <param name="sender">CodeDescriptionWindow</param>
        /// <param name="e">RoutedEventArgs</param>
        private void SaveButton_Click(object sender, RoutedEventArgs e)
        {
            try
            {
                using (FileStream fs = new FileStream("SavedCodes.nk", FileMode.Open))
                {
                    fs.Seek(0, SeekOrigin.End);
                    formatter.Serialize(fs, AGCode);
                }
            }
            catch (FileNotFoundException)
            {
                using (FileStream fs = new FileStream("SavedCodes.nk", FileMode.Create))
                {
                    fs.Seek(0, SeekOrigin.End);
                    formatter.Serialize(fs, AGCode);
                }
            }
            catch (Exception)
            {

            }
            littleTextBlock.Text = "Сохранено!";
        }

    }
}
```

1.2.9 Код окна CodeWindow

```
using System.Windows;
using System.Windows.Controls;
using System.Windows.Input;
using Booler;
using Booler.Exceptions;
using System;
using System.Text.RegularExpressions;

namespace Генератор_алгеброгеометрических_кодов
{
    /// <summary>
    /// Логика взаимодействия для CodeWindow.xaml
    /// </summary>
    public partial class CodeWindow : Window
    {
        /// <summary>
        /// Выбранный код.
        /// </summary>
        Code AGCode;
        /// <summary>
        /// Указываетна выбор операции.
        /// </summary>
        bool flag;
        /// <summary>
        /// Допустимые для ввода символы.
```

```
        /// </summary>
        Regex regex = new Regex("[^0-1]+");

        /// <summary>
        /// Конструктор окна CodeWindow.
        /// </summary>
        /// <param name="AGCode">Выбранный код</param>
        public CodeWindow(Code AGCode)
        {
            this.AGCode = AGCode;
            InitializeComponent();
            first.MaxLength = AGCode.K;
            flag = true;
            ToolTip t = new ToolTip()
            {
          Content = "Длина кодируемого слова - " + AGCode.K + "\nДлина кодового слова - "
            + AGCode.N + "\nМаксимальное количество исправляемых ошибок - " + AGCode.T
            };
            helpButton.ToolTip = t;
        }


        /// <summary>
        /// Возвращает к предыдущем окну.
        /// </summary>
        /// <param name="sender">CodeWindow</param>
        /// <param name="e">RoutedEventArgs</param>
        private void PreviousWindowButton_Click(object sender, RoutedEventArgs e)
        {
            CodeDescriptionWindow win = new CodeDescriptionWindow(AGCode);
            win.Top = this.Top;
            win.Left = this.Left;
            win.Show();
            this.Close();
        }

        /// <summary>
        /// Кодирует или декодирует сообщение.
        /// </summary>
        /// <param name="sender">CodeWindow</param>
        /// <param name="e">RoutedEventArgs</param>
        private void CodeButton_Click(object sender, RoutedEventArgs e)
        {
            if (flag)
            {
                if (first.Text.Length != AGCode.K)
                {
                 error.Visibility = Visibility.Visible;
                 error.Text = $"   Длина кодируемого слова должна равняться {AGCode.K}.";
                    return;
                }
                try
                {
                    int[] ourMessage = MyStatics.ToIntArray(first.Text, AGCode.K);
                    int[] ourCodeMessage = AGCode.Encode(ourMessage);
                    second.Text = "";

                    for (int i = 0; i < ourCodeMessage.Length; i++) second.Text +=
ourCodeMessage[i];
```

```
                }
                catch (Exception ex)
                {
                    error.Visibility = Visibility.Visible;
                    error.Text = ex.Message;
                }
            }
            else
            {
                if (first.Text.Length != AGCode.N)
                {
                    error.Visibility = Visibility.Visible;
                    error.Text = $"Длина кодового слова должна равняться {AGCode.N}.";
                    return;
                }


                try
                {
                    int[] ourDecodedMessage;
                    int[] ourMessage = MyStatics.ToIntArray(first.Text, AGCode.N);
                    ourDecodedMessage = AGCode.Decode(ourMessage);
                    second.Text = "";
                    for (int i = 0; i < ourDecodedMessage.Length; i++) second.Text +=
ourDecodedMessage[i];
                }
                catch (MistakesNumberException ex)
                {
                    error.Visibility = Visibility.Visible;
                    error.Text = ex.Message;
                    return;
                }
                catch (Exception ex)
                {
                    error.Visibility = Visibility.Visible;
                    error.Text = ex.Message;
                }

            }
        }

        /// <summary>
        /// Переключает режим кодирования.
        /// </summary>
        /// <param name="sender">CodeWindow</param>
        /// <param name="e">SelectionChangedEventArgs</param>
        private void CodeOrDecodeComboBox_SelectionChanged(object sender,
SelectionChangedEventArgs e)
        {
            ComboBox comboBox = (ComboBox)sender;
            ComboBoxItem selectedItem = (ComboBoxItem)comboBox.SelectedItem;
            if ((selectedItem.Content.ToString()) == "Кодирование")
            {
                first.Text = "";
                second.Text = "";
                first.MaxLength = AGCode.K;
                flag = true;
            }
```

```
            if ((selectedItem.Content.ToString()) == "Декодирование")
            {
                first.Text = "";
                second.Text = "";
                first.MaxLength = AGCode.N;
                flag = false;
            }
        }

        /// <summary>
        /// Синхронизирует работу полей для кодового и кодируемого сообщения.
        /// </summary>
        /// <param name="sender">CodeWindow</param>
        /// <param name="e">TextChangedEventArgs</param>
        private void First_TextChanged(object sender, TextChangedEventArgs e)
        {
            second.Text = "";
            error.Visibility = Visibility.Hidden;

        }

        /// <summary>
        /// Запрещает ввод пробелов.
        /// </summary>
        /// <param name="sender">CodeWindow</param>
        /// <param name="e">KeyEventArgs</param>
        private void FirstPreviewKeyDown(object sender, KeyEventArgs e)
        {
            e.Handled = (e.Key == Key.Space);
            if (e.Key == Key.Return)
            {
                CodeButton_Click(this, new RoutedEventArgs());
            }
        }

        /// <summary>
        /// Проверяет допустимость вводимых символов. Допускает ввод только 0 и 1.
        /// </summary>
        /// <param name="sender">CodeWindow</param>
        /// <param name="e">TextCompositionEventArgs</param>
        private void First_PreviewTextInput(object sender, TextCompositionEventArgs e)
        {
            e.Handled = regex.IsMatch(e.Text);
        }

        /// <summary>
        ///
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void helpButton_Click(object sender, RoutedEventArgs e)
        {
            DescribtionWindow win = new DescribtionWindow(AGCode);
            win.Top = this.Top + 100;
            win.Left = this.Left + 140;
            win.ShowDialog();
        }
    }
```

```
}
```

1.2.10 Код окна DescribtionWindow

```
using System.Windows;
using Booler;

namespace Генератор_алгеброгеометрических_кодов
{
    /// <summary>
    /// Логика взаимодействия для DescribtionWindow.xaml
    /// </summary>
    public partial class DescribtionWindow : Window
    {
        /// <summary>
        /// Конструктор окна DescribtionWindow.
        /// </summary>
        /// <param name="code">Алгеброгеометрический код</param>
        public DescribtionWindow(Code code)
        {
            InitializeComponent();
            K.Content = "Длина кодируемого слова - " + code.K;
            N.Content = "Длина кодового слова - " + code.N;
            T.Content = "Максимальное количество исправляемых ошибок - " + code.T;
        }
    }
}
```

ПРИЛОЖЕНИЕ 1

ТЕРМИНОЛОГИЯ

Таблица 1

Термин	**Определение**
Алгебраическая геометрия	Раздел алгебры, основным предметом изучения которого являются алгебраические многообразия.
Алгебраическое многообразие	Множество решений системы алгебраических уравнений над комплексными или действительными числами.
Кодирование	Процесс преобразования исходной информации в удобную для передачи форму.
Алгеброгеометрический код	Линейный блочный код, построенный на основе алгебраического многообразия какой-либо системы алгебраических уравнений. Основной принцип его работы: • составляется порождающая матрица – базис подпространства кодовых слов; • для кодирования порождающая матрица умножается на вектор – слово, заданное пользователем; • для декодирования происходит проверка кода с помощью проверочной матрицы, для которой порождающая матрица является базисом ядра гомоморфизма линейных подпространств кодовых и кодируемых слов. В случае обнаружения ошибки, последняя исправляется. Если ошибок не найдено, происходит процесс декодирования: матрица, обратная порождающей, умножается на вектор кодового слова.
Расстояние Хэмминга	Число позиций, в которых два слова одной длины отличаются.
Токен	Условное название наименьшей значащей группы символов в уравнении.

ПРИЛОЖЕНИЕ 2

ОПИСАНИЕ ФОРМАТА .nk

Файл формата .nk содержит информацию об алгеброгеометрическом коде, созданном в программе Генератор алгеброгеометрических кодов, в сериализованном виде.

Открыть файл формата .nk можно программой Генератор алгеброгеометрических кодов.

Программа Генератор алгеброгеометрических кодов при этом осуществляет корректное открытие только файлов формата .nk, созданных или измененных в этой программе.

ЛИСТ РЕГИСТРАЦИИ ИЗМЕНЕНИЙ

Лист регистрации изменений									
Номера листов (страниц)					Всего листов (страниц в докум.)	№ документа	Входящий № сопроводительного докум. и дата	Подп.	Дата
Изм.	Измененных	Заменен ных	Новых	Аннулированных					

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.502610-01 12				
Инв. № подл.	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата