Министерство образования и науки Российской Федерации

Рязанский государственный радиотехнический университет имени В.Ф.Уткина

Кафедра вычислительной и прикладной математики

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе

по дисциплине

«ВИЗУАЛЬНОЕ ПРОГРАММИРОВАНИЕ»

на тему

«Шифр Цезаря».

Выполнил: студент группы 744

Ванюков А.А.

Проверил: старший преподаватель

Жулева С.Ю.

Рязань 2019

Оглавление

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
РЯЗАНСКИЙ ГОСУДАРСТВЕННЫЙ РАДИОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМЕНИ
В.Ф УТКИНА
КАФЕДРА ВЫЧИСЛИТЕЛЬНОЙ И ПРИКЛАДНОЙ МАТЕМАТИКИ

ЗАДАНИЕ

К курсовой работе по дисциплине: «Визуальное программирование»

студента группы 744 Ванюкова Александра Андреевича

Создать приложение **«Шифр Цезаря»**

_____

с применением стандартных и пользовательских интерфейсных элементов.

Содержательная сторона пояснительной записки должна включать в себя описание всех этапов проделанной работы. Материал пояснительной записки рекомендуется располагать в следующем порядке:

- **Титульный лист**.
- **Оглавление**.
- **Задание на курсовую работу**.
- **Введение в проблему и ее состояние на сегодняшний день** (необходимость решения поставленной задачи).
- **Анализ и постановка задачи**. Этот пункт состоит из разделов:
  - *проектная часть*. Она включает в себя:
    - описание предметной области (цель задачи). В этот пункт входит описание правил разрабатываемой игры либо структура разрабатываемой БД;
    - разработка алгоритма. Реализация задач в программе поставленных в предыдущем пункте;
  - *интерфейсная часть*. В этот пункт включается подробное рассмотрение компонентов, реализующих разработанный алгоритм программного продукта: меню, формы и т.д.;
  - *отладка программы* (описание возникавших проблем при создании программного продукта на этапе проектирования и компиляции).
- **Руководство пользователя**. Этот пункт включает в себя:
  - *системные требования* (т.е. какое программное обеспечение требуется);
  - *установка программы* (данный пункт особенно важен для БД, это связано с переносом БД на другой ПК);
  - *запуск программы;*
  - непосредственно *инструкция пользователю* для работы в данном программном продукте.
- **Заключение**.
- **Список литературы**.
- **Приложение** (данный пункт должен содержать текст программного продукта и его электронный вариант [диск]).

При выполнении курсовой работы предусмотреть первоначальную заставку, справку о программе и современный интерфейс для общения с пользователем МЕНЮ с иерархической системой.

Срок преставления курсовой работы **23 декабря 2019 года**.

Старший преподаватель кафедры ВПМ Жулева С.Ю.

Введение

Шифр Цезаря - один из самых простых и широко известных методов шифрования.

Шифр Цезаря — это вид шифра подстановки, в котором каждый символ в открытом тексте заменяется символом, находящимся на некотором постоянном числе позиций левее или правее него в алфавите. Он был так назван в честь Гая Юлия Цезаря, который, по поверию, придумал этот шифр.

Шифр Цезаря также является частью более сложных шифров, таких как шифр Виженера. Однако при всём этом шифр Цезаря легко взламывается и не имеет практического применения на сегодняшний день.

Для раскрытия темы курсовой работы ключевыми являются задачи реализации шифрование, дешифрования и взлома шифра Цезаря. Это возможно реализовать при помощи языка C# и IDE Visual Studio.

1. Анализ предметной области

Основой шифра Цезаря является замена букв в исходном тексте на буквы, идущие в алфавите языка с определённым сдвигом от исходной буквы.

Исходными данными для шифрования и дешифрования являются исходный/зашифрованный текст, сдвиг и язык текста.

Если говорить о взломе, то в качестве исходных данных будет достаточно текста и языка этого текста. Сдвиг не нужен.

В математическом виде шифрование шифром Цезаря выглядит следующим образом:

$$x = y + k$$

, где x – новая буква(зашифрованная), y – исходная буква, а k – сдвиг по алфавиту.

В программе будут очень полезен тип коллекции Dictionary, который реализован в языке C#. Dictionary представляет собой пару ключ-значение, что очень удобно для хранения частоты встречи букв в тексте.

Кроме того активно используется коллекция List, как весьма удобная замена массиву, позволяющая динамически добавлять данные[1].

Я придумал несколько методов решения:

1) Использовать символы UNICODE;
2) Использовать заранее заготовленные List'ы с алфавитами языков[5].

Мною были реализованы алгоритмы на основе обоих этих методов и оценены их показатели при помощи утилиты диагностики StopWatch. В результате мои догадки подтвердились и метод, основанный на символах UNDICODE, имел время выполнения в разы меньше, чем метод, основанный на List'ах. Было принято решение использовать именно его[5].

2. Описание алгоритма программы

Я реализовал шифр Цезаря в отдельной библиотеке классов, что позволит в будущем использовать уже сделанные алгоритмы как в других приложениях, так и на других платформах, например, сайтах[1].

Рассмотрим схему работы шифрования, дешифрования и взлома шифра Цезаря.

Шифрование (рисунок 1):

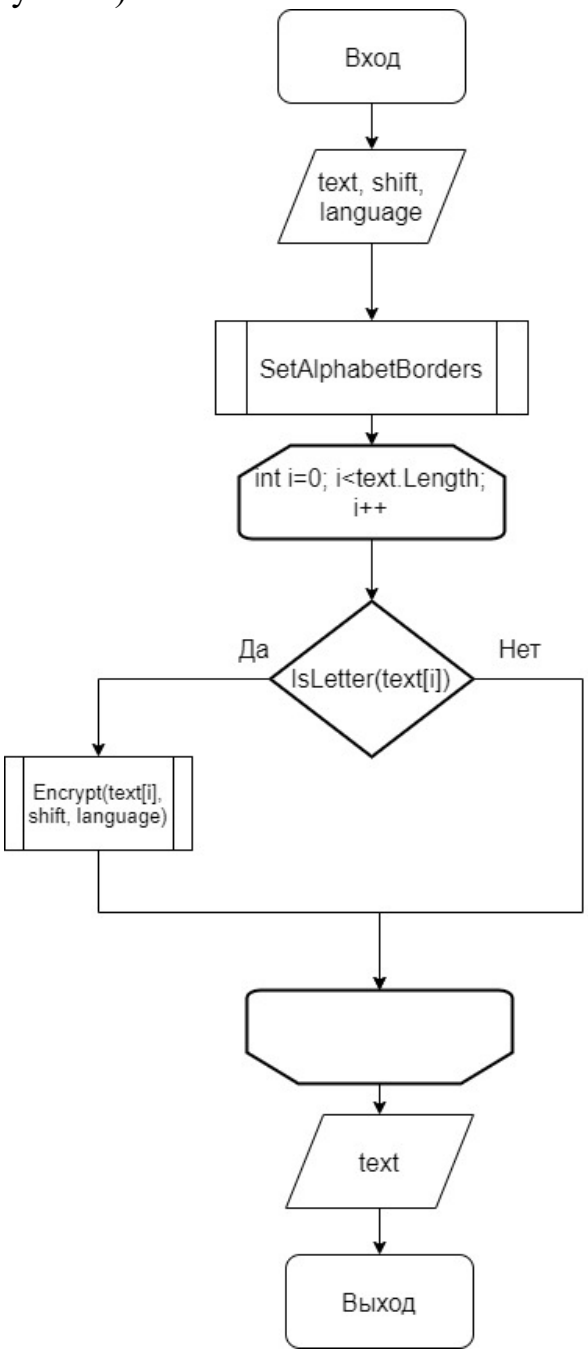

Рисунок 1

Метод Encrypt выглядит следующим образом (рисунок 2):

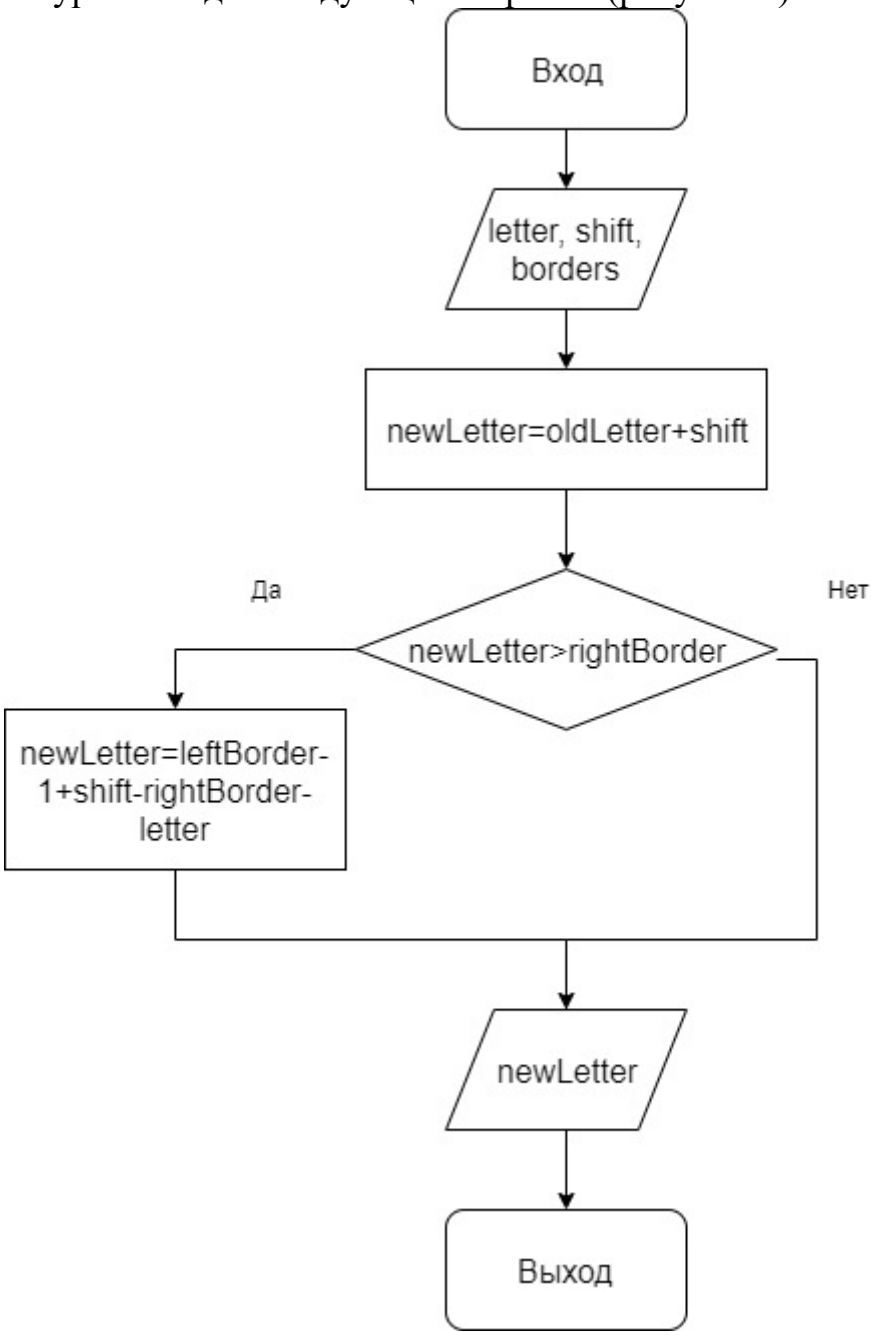

Рисунок 2

Дешифрование (рисунок 3):

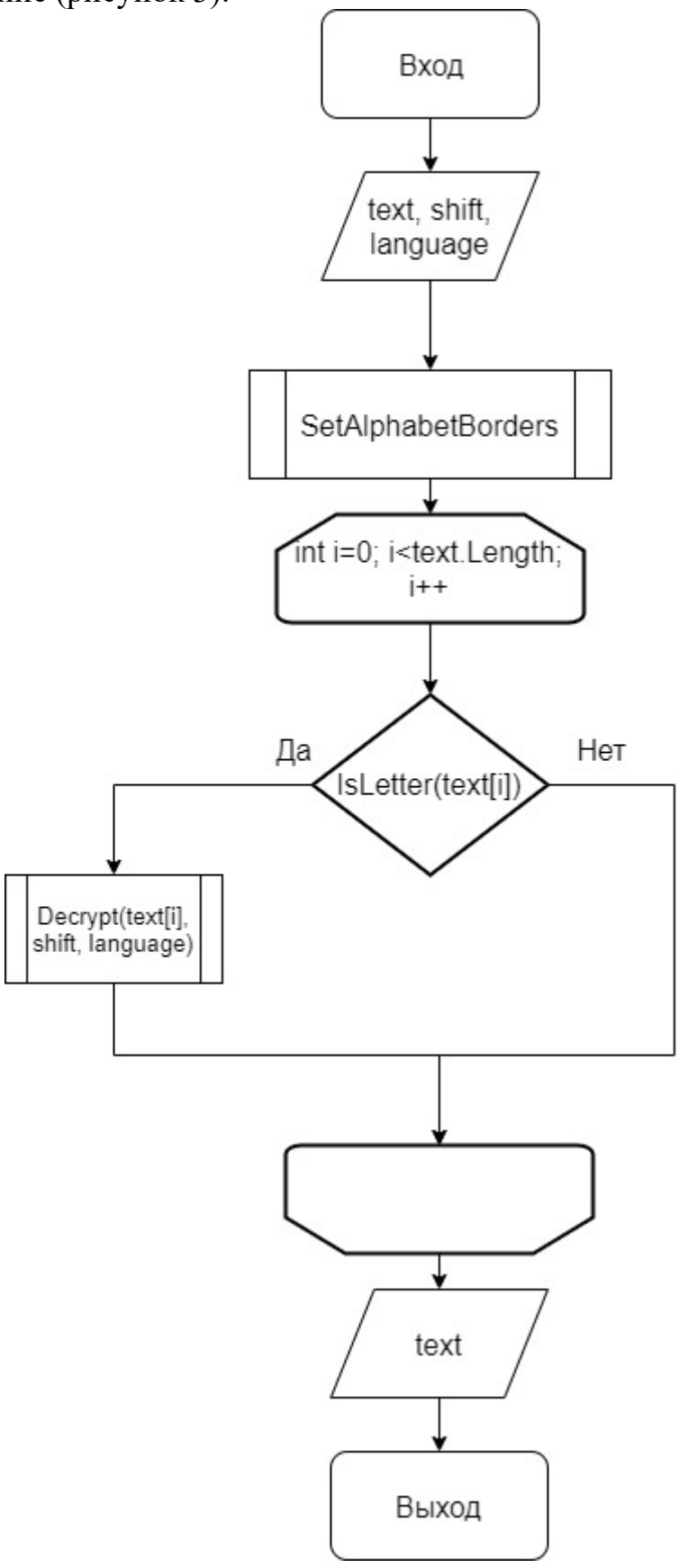

Рисунок 3

Decrypt (рисунок 4):

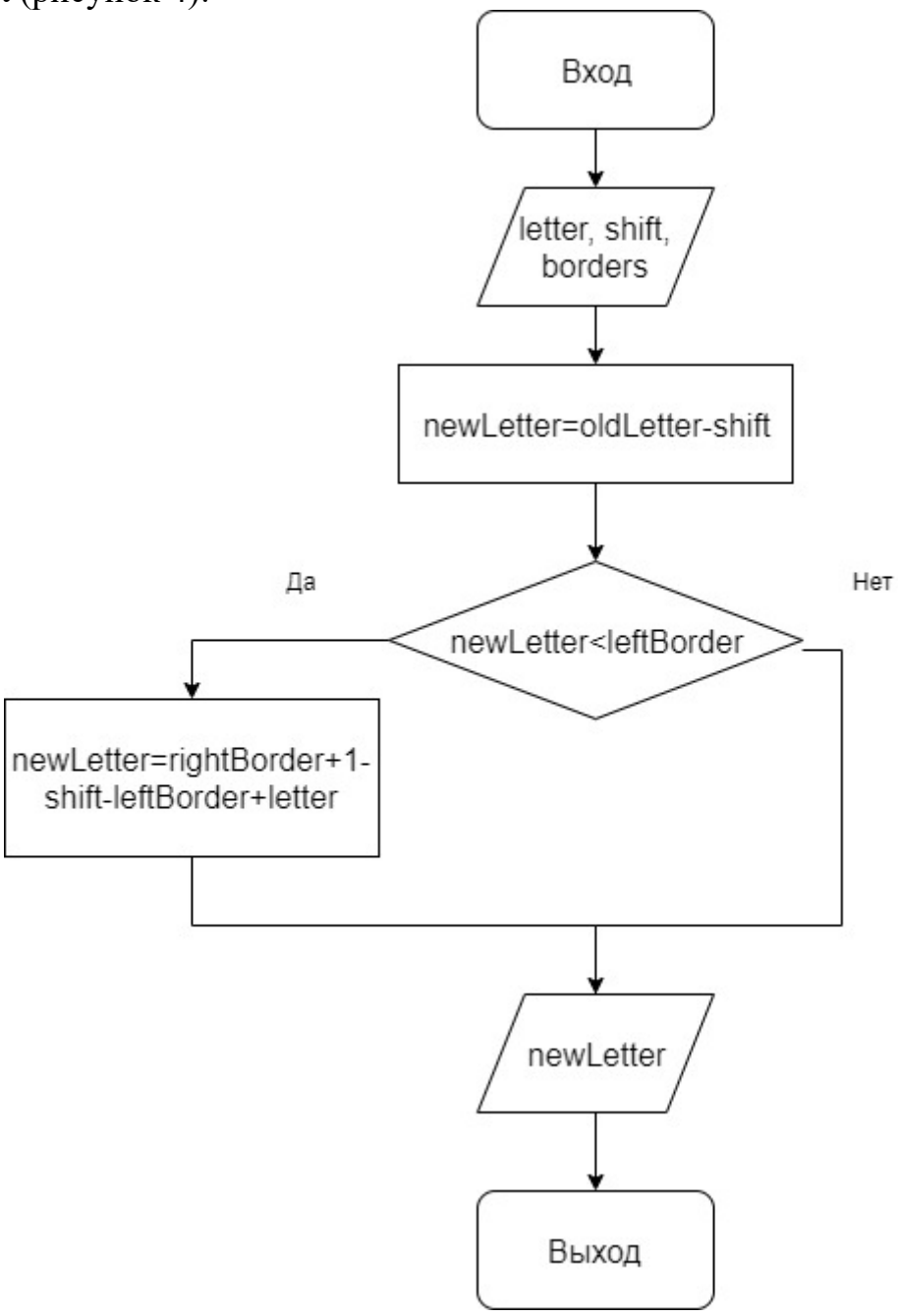

Рисунок 4

3. Обоснование выбора используемого программного обеспечения

Для реализации приложения был использован высокоуровневый строго типизированный объектно-ориентированный язык программирования C#. Я выбрал его, так как он позволяет удобно реализовывать алгоритмы программы с минимальными затратами ресурсов компьютера.

В качестве IDE была выбрана Microsoft Visual Studio 2019.

4. Разработка программы и её описание

В программе реализован интерфейс ICipher, который содержит методы Encrypt, Decrypt, Hack. Это сделано для того, чтобы в дальнейшем иметь возможность расширять приложение (добавлять новые виды шифров) без необходимости менять Presentation Layer. От этого интерфейса наследуется класс CaesarCipher, который реализует эти методы. Кроме того класс CaesarCipher имеет статический конструктор, который вызывается лишь один раз за работу программы – перед первым обращением к какому-либо статическому полю или методу или перед первым созданием объекта класса. В нём происходит заполнение словарей с «идеальными» частотами букв в алфавитах[2].

5. Интерфейсная часть

Программа использует библиотеку MaterialDesign, которая создаёт интерфейс, рекомендуемый компанией Google. Для реализации этого интерфейса форма приложения наследуется от MaterialForm.

На форме находится TabControl, который имеет две вкладки: «Настройки» и «Шифр Цезаря».

На вкладке «Шифр Цезаря» есть два TextBox’а для исходного и обработанного текста, а также RadioButton’ы для выбора настроек действий.

На вкладке «Настройки» находятся RadioButton’ы для выбора темы, цвета и языка интерфейса программы[3].

При запуске программы сразу открывается вкладка «Шифр Цезаря» (рисунок 5).

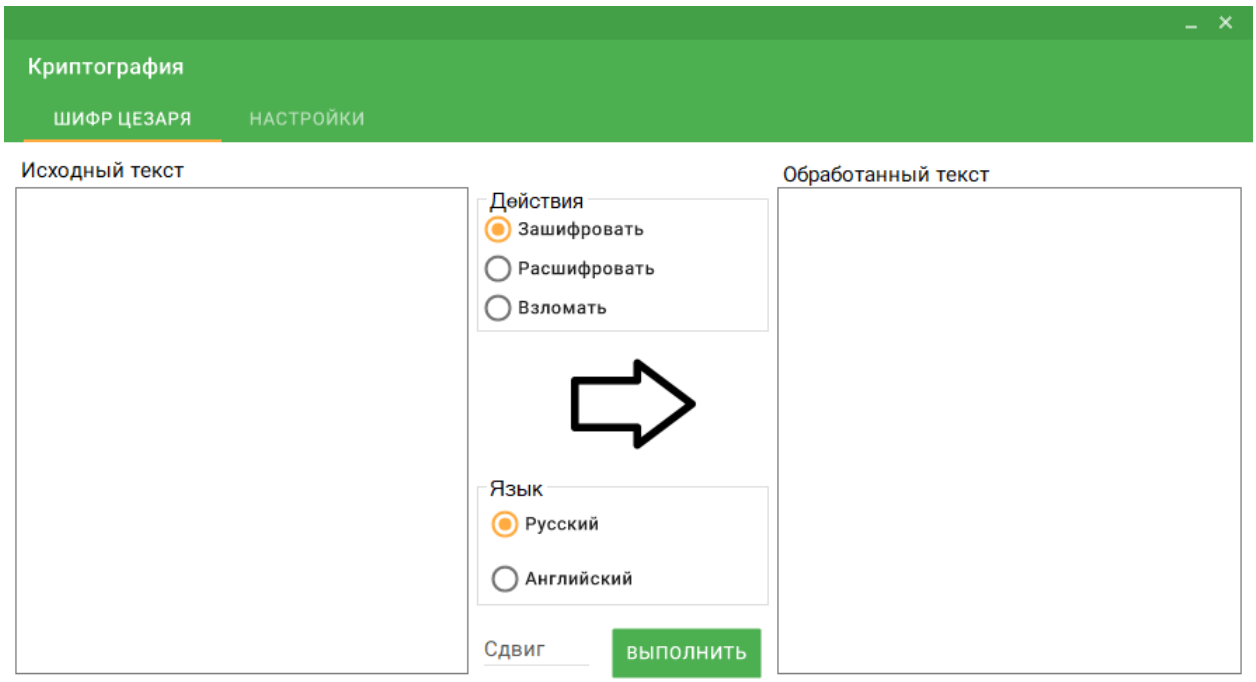

Рисунок 5

После ввода исходного текста и выбора параметров получаем обработанный текст (рисунок 6).

Рисунок 6

Вкладка «Настройки» имеет следующий вид (рисунок 7):

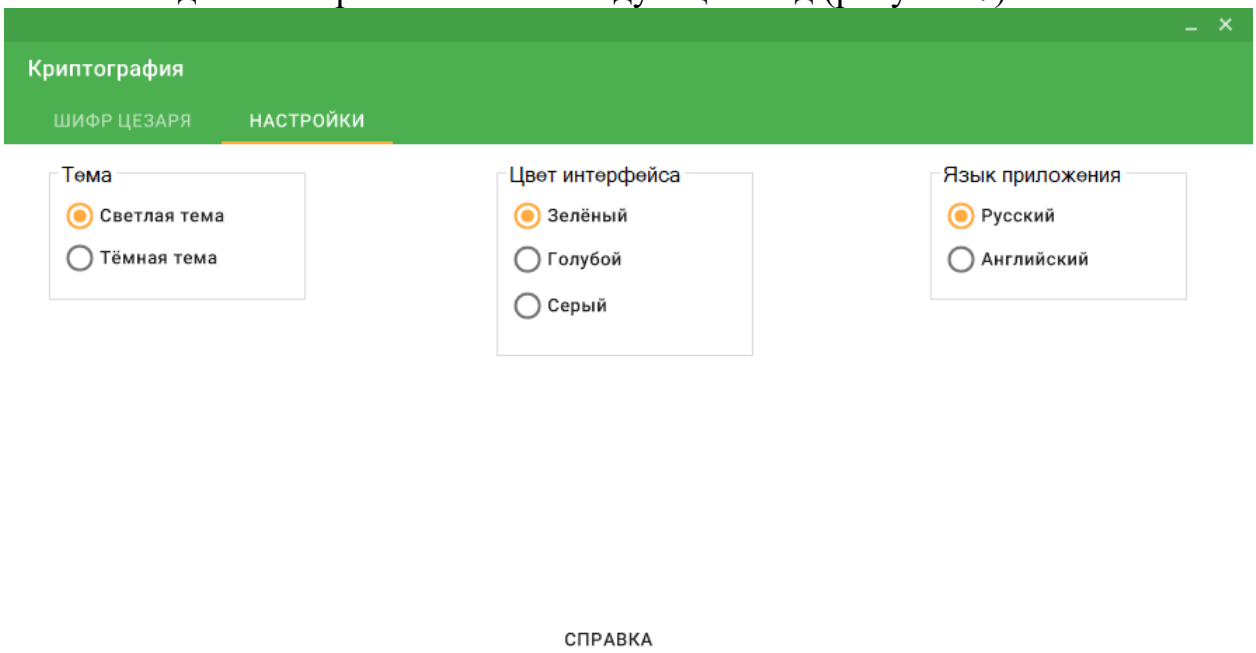

Рисунок 7

Здесь можно выбрать оформление приложения.

Кроме того, в приложении реализована справка для помощи пользователю (рисунок 8).

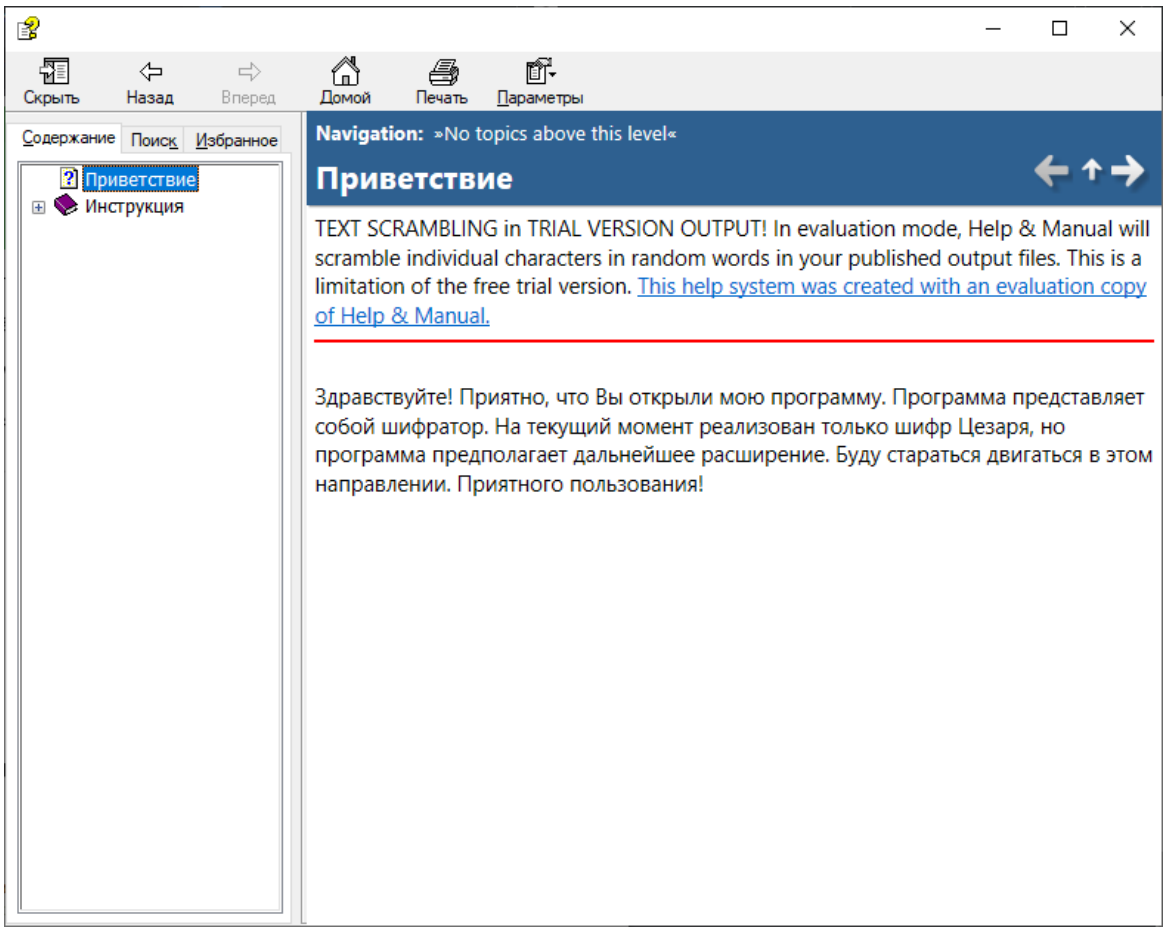
Скрыть
Назад
Вперед
Домой
Печать
Параметры
Содержание
Поиск
Избранное
Приветствие
Инструкция
Navigation: »No topics above this level«
Приветствие
TEXT SCRAMBLING in TRIAL VERSION OUTPUT! In evaluation mode, Help & Manual will scramble individual characters in random words in your published output files. This is a limitation of the free trial version. This help system was created with an evaluation copy of Help & Manual.
Здравствуйте! Приятно, что Вы открыли мою программу. Программа представляет собой шифратор. На текущий момент реализован только шифр Цезаря, но программа предполагает дальнейшее расширение. Буду стараться двигаться в этом направлении. Приятного пользования!

Рисунок 8

6. Тестирование

Цель тестирования, т.е. испытания программы, заключается в выявлении имеющихся в программе ошибок[4].

Содержание	**Ожидаемый результат**	**Статус**
Тестирование шифрования текста с известным сдвигом	При нажатии кнопки «Выполнить» исходный текст преобразуется в правильно зашифрованный текст	Работает
Тестирование дешифрования текста с известным сдвигом	При нажатии кнопки «Выполнить» зашифрованный текст преобразуется в правильно расшифрованный текст	Работает
Тестирование взлома текста без сдвига	При нажатии кнопки «Выполнить» зашифрованный текст преобразуется в правильно расшифрованный текст без указанного сдвига	Работает
Тестирование отображения справки	При нажатии пункта настроек «Справка» отобразиться окно справки	Работает

7. Системные требования

Для запуска приложения необходим только .exe файл программы и операционная система не ниже Windows XP.

Кроме того, обязательно наличие оперативной памяти не менее 55 МБ и места на жёстком диске не менее 40 МБ.

8. Руководство пользователя

Полное руководство пользователя удобно реализовано в справке, которая запускается из настроек программы по клику кнопки «Справка»[4].

9. Заключение

Итак, в ходе данной работы были выполнены такие цели, как

1. анализ задания и постановка задачи;
2. разработка и описание основных алгоритмов;
3. создание программного продукта;
4. отладка программы;
5. написана инструкция по использованию;

Программа отвечает всем заявленным требованиям и выполняет все поставленные задачи. Программа получилась понятной для пользователя.

Таким образом, можно с уверенностью сказать, что цель курсовой работы достигнута, так как её результатом является готовый программный продукт, удовлетворяющий поставленной задаче.

10. Библиографический список

1. Алексей Васильев - Программирование на C# для начинающих. Основные сведения. Москва, Изд-во «Эксмо», 2018. стр. 130 - 171.

2. Курс лекций по дисциплине «Визуальное программирование» за 4-5 семестр.

3. «Киберфорум» [электронный ресурс] – режим доступа: http://www.cyberforum.ru/, свободный. –Загл. С экрана. Дата обращения : 05.12.2019 - 25.12.2019

4. «Stackoverflow» [электронный ресурс] – режим доступа: https://ru.stackoverflow.com/, свободный. –Загл. С экрана. Дата обращения : 05.12.2019 - 25.12.2019

5. Справка Microsoft CSharp [электронный ресурс] – Режим доступа : https://docs.microsoft.com/ru-ru/dotnet/csharp/, свободный. – Загл. с экрана.

11. Приложение

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

namespace Ciphers
{
    /// <summary>
    /// Класс, реализующий шифрование текста при помощи шифра Цезаря, а также расшифрование текста и взлом ключа по тексту,
    /// зашифрованных при помощи шифра Цезаря.
    /// </summary>
    public class CaesarCipher : ICipher
    {
        /// <summary>
        /// Словарь с частотами букв в латинице.
        /// </summary>
        static readonly Dictionary<char, double> latinLettersIdealProbabilities;
        /// <summary>
        /// Словарь с частотами букв в кириллице.
        /// </summary>
        static readonly Dictionary<char, double> cyrillicLettersIdealProbabilities;

        /// <summary>
        /// Статический конструктор, в котором происходит заполнение словарей значениями вероятностей букв для языков.
        /// </summary>
        static CaesarCipher()
        {
            latinLettersIdealProbabilities = new Dictionary<char, double>();
            for (int i = 97; i <= 122; i++)
            {
                latinLettersIdealProbabilities.Add((char)i, 0);
            }
            latinLettersIdealProbabilities['a'] = 0.0817;
            latinLettersIdealProbabilities['b'] = 0.0149;
            latinLettersIdealProbabilities['c'] = 0.0278;
            latinLettersIdealProbabilities['d'] = 0.0425;
            latinLettersIdealProbabilities['e'] = 0.127;
            latinLettersIdealProbabilities['f'] = 0.0223;
            latinLettersIdealProbabilities['g'] = 0.0202;
            latinLettersIdealProbabilities['h'] = 0.0609;
            latinLettersIdealProbabilities['i'] = 0.0697;
            latinLettersIdealProbabilities['j'] = 0.0015;
            latinLettersIdealProbabilities['k'] = 0.0077;
            latinLettersIdealProbabilities['l'] = 0.0403;
            latinLettersIdealProbabilities['m'] = 0.0241;
            latinLettersIdealProbabilities['n'] = 0.0675;
            latinLettersIdealProbabilities['o'] = 0.0751;
            latinLettersIdealProbabilities['p'] = 0.0193;
            latinLettersIdealProbabilities['q'] = 0.001;
            latinLettersIdealProbabilities['r'] = 0.0599;
            latinLettersIdealProbabilities['s'] = 0.0633;
            latinLettersIdealProbabilities['t'] = 0.0906;
            latinLettersIdealProbabilities['u'] = 0.0276;
            latinLettersIdealProbabilities['v'] = 0.0098;
            latinLettersIdealProbabilities['w'] = 0.0236;
            latinLettersIdealProbabilities['x'] = 0.0015;
            latinLettersIdealProbabilities['y'] = 0.0197;
            latinLettersIdealProbabilities['z'] = 0.0005;

            cyrillicLettersIdealProbabilities = new Dictionary<char, double>();
            for (int i = 1072; i <= 1103; i++)
            {
                cyrillicLettersIdealProbabilities.Add((char)i, 0);
            }
            cyrillicLettersIdealProbabilities['а'] = 0.0801;
            cyrillicLettersIdealProbabilities['б'] = 0.0159;
            cyrillicLettersIdealProbabilities['в'] = 0.0454;
            cyrillicLettersIdealProbabilities['г'] = 0.017;
            cyrillicLettersIdealProbabilities['д'] = 0.0298;
            cyrillicLettersIdealProbabilities['е'] = 0.0845;
            cyrillicLettersIdealProbabilities['ж'] = 0.0094;
            cyrillicLettersIdealProbabilities['з'] = 0.0165;
            cyrillicLettersIdealProbabilities['и'] = 0.0735;
            cyrillicLettersIdealProbabilities['й'] = 0.0121;
```

```
            cyrillicLettersIdealProbabilities['к'] = 0.0349;
            cyrillicLettersIdealProbabilities['л'] = 0.0440;
            cyrillicLettersIdealProbabilities['м'] = 0.0321;
            cyrillicLettersIdealProbabilities['н'] = 0.067;
            cyrillicLettersIdealProbabilities['о'] = 0.1097;
            cyrillicLettersIdealProbabilities['п'] = 0.0281;
            cyrillicLettersIdealProbabilities['р'] = 0.0473;
            cyrillicLettersIdealProbabilities['с'] = 0.0547;
            cyrillicLettersIdealProbabilities['т'] = 0.0626;
            cyrillicLettersIdealProbabilities['у'] = 0.0262;
            cyrillicLettersIdealProbabilities['ф'] = 0.0026;
            cyrillicLettersIdealProbabilities['х'] = 0.0097;
            cyrillicLettersIdealProbabilities['ц'] = 0.0048;
            cyrillicLettersIdealProbabilities['ч'] = 0.0144;
            cyrillicLettersIdealProbabilities['ш'] = 0.0073;
            cyrillicLettersIdealProbabilities['щ'] = 0.0036;
            cyrillicLettersIdealProbabilities['ъ'] = 0.0004;
            cyrillicLettersIdealProbabilities['ы'] = 0.019;
            cyrillicLettersIdealProbabilities['ь'] = 0.0174;
            cyrillicLettersIdealProbabilities['э'] = 0.0032;
            cyrillicLettersIdealProbabilities['ю'] = 0.0064;
            cyrillicLettersIdealProbabilities['я'] = 0.0201;
        }

        #region Encrypt

        /// <summary>
        /// Зашифровывает строку.
        /// </summary>
        /// <param name="sourceText">Исходная строка</param>
        /// <param name="shift">Сдвиг вперёд по алфавиту каждого символа в строке</param>
        /// <param name="dictionaryLanguage">Язык алфавита</param>
        /// <returns>Зашифрованная строка</returns>
        public string Encrypt(string sourceText, int shift, string dictionaryLanguage)
        {
            StringBuilder sourceTextSB = new StringBuilder(sourceText);
            SetAlphabetBorders(dictionaryLanguage, out int firstLetterUpperCase, out int lastLetterUpperCase, out int firstLetterLowerCase,
                out int lastLetterLowerCase, out int additionalLetterUpperCase, out int additionalLetterLowerCase);
            if (shift > lastLetterUpperCase - firstLetterUpperCase)//можно также сравнить и по нижнему регистру, но это равнозначно
                throw new Exception("Максимальный сдвиг больше, чем длина алфавита языка минус 1");
            for (int i = 0; i < sourceTextSB.Length; i++)
            {
                if (Char.IsLetter(sourceTextSB[i]) && sourceTextSB[i] != additionalLetterLowerCase && sourceTextSB[i] !=
additionalLetterUpperCase)
                //здесь дополнительно проверям не является ли буква "исключением". Если является, то просто пропускаем её(для кириллицы
- "Ё" и "ё")
                {
                    int leftBorder, rightBorder;
                    if (char.IsUpper(sourceTextSB[i]))
                    {
                        leftBorder = firstLetterUpperCase;
                        rightBorder = lastLetterUpperCase;
                    }
                    else
                    {
                        leftBorder = firstLetterLowerCase;
                        rightBorder = lastLetterLowerCase;
                    }
                    sourceTextSB[i] = EncryptCharShift(sourceTextSB[i], shift, leftBorder, rightBorder);
                }
            }
            return sourceTextSB.ToString();
        }

        /// <summary>
        /// Заменяет символ на другой символ дальше по алфавиту с заданным сдвигом.
        /// </summary>
        /// <param name="letter">Исходный(заменяемый) символ</param>
        /// <param name="shift">Сдвиг - количество символов, на которое нужно сместить исходный</param>
        /// <param name="bordersArray">Границы алфавита в UNICODE, которому принадлежит символ. Массив должен состоять из двух
элементов: первый - левая граница; второй - правая граница</param>
        /// <returns>Новый получившийся символ</returns>
        private char EncryptCharShift(char letter, int shift, params int[] bordersArray)
        {
            int letterNumber = (int)letter;
            int newLetter = letterNumber + shift;
            if (newLetter > bordersArray[1])
```

```
            newLetter = (bordersArray[0] - 1) + (shift - (bordersArray[1] - letterNumber));//"-1" используется для того, чтобы имитировать
                                                                                           //переход с последней буквы алфавита на первую букву алфавита, так как без "-
1" буква преобразуется в следующую за той, которая должна была бы быть
            return (char)newLetter;
        }

        #endregion

        #region Decrypt

        /// <summary>
        /// Расшифровывает строку.
        /// </summary>
        /// <param name="cipherText">Зашифрованная строка</param>
        /// <param name="shift">Сдвиг назад по алфавиту каждого символа в строке</param>
        /// <param name="dictionaryLanguage">Язык алфавита</param>
        /// <returns>Расшифрованная строка</returns>
        public string Decrypt(string cipherText, int shift, string dictionaryLanguage)
        {
            StringBuilder cipherTextSB = new StringBuilder(cipherText);
            SetAlphabetBorders(dictionaryLanguage, out int firstLetterUpperCase, out int lastLetterUpperCase, out int firstLetterLowerCase,
                out int lastLetterLowerCase, out int additionalLetterUpperCase, out int additionalLetterLowerCase);
            if (shift > lastLetterUpperCase - firstLetterUpperCase)
                throw new Exception("Максимальный сдвиг больше, чем длина алфавита языка минус 1");
            for (int i = 0; i < cipherTextSB.Length; i++)
            {
                if (Char.IsLetter(cipherTextSB[i]) && cipherTextSB[i] != additionalLetterLowerCase && cipherTextSB[i] !=
additionalLetterUpperCase)
                //здесь дополнительно проверям не является ли буква "исключением". Если является, то просто пропускаем её(для кириллицы
- "Ё" и "ё")
                {
                    int leftBorder, rightBorder;
                    if (char.IsUpper(cipherTextSB[i]))
                    {
                        leftBorder = firstLetterUpperCase;
                        rightBorder = lastLetterUpperCase;
                    }
                    else
                    {
                        leftBorder = firstLetterLowerCase;
                        rightBorder = lastLetterLowerCase;
                    }
                    cipherTextSB[i] = DecryptCharShift(cipherTextSB[i], shift, leftBorder, rightBorder);
                }
            }
            return cipherTextSB.ToString();
        }

        /// <summary>
        /// Заменяет символ на другой символ раньше по алфавиту с заданным сдвигом.
        /// </summary>
        /// <param name="letter">Заменяемый символ</param>
        /// <param name="shift">Сдвиг - количество символов, на которое нужно сместить исходны</param>
        /// <param name="bordersArray">Границы алфавита в UNICODE, которому принадлежит символ. Массив должен состоять из двух
элементов: первый - левая граница; второй - правая граница</param>
        /// <returns>Новый получившийся символ</returns>
        private char DecryptCharShift(char letter, int shift, params int[] bordersArray)
        {
            int letterNumber = (int)letter;
            int newLetter = letterNumber - shift;
            if (newLetter < bordersArray[0])
                newLetter = (bordersArray[1] + 1) - (shift - (letterNumber - bordersArray[0]));//"+1" используется для того, чтобы имитировать
                                                                                               //переход с первой буквы алфавита на последнюю букву алфавита, так как без
"+1" буква преобразуется в предыдущую той, которая должна была бы быть
            return (char)newLetter;
        }

        #endregion

        #region Hack

        /// <summary>
        /// Расшифровывает строку, взламывая шифр Цезаря.
        /// </summary>
        /// <param name="cipherText">Зашифрованная строка</param>
        /// <param name="dictionaryLanguage">Язык алфавита</param>
        /// <returns>Расшифрованная строка</returns>
```

```
        public string Hack(string cipherText, string dictionaryLanguage)
        {
            SetAlphabetBorders(dictionaryLanguage, firstLetterUpperCase: out _, out _, out int firstLetterLowerCase,
                out int lastLetterLowerCase, out _, out int additionalLetterLowerCase);
            Dictionary<char, double> cipherTextLettersProbabilities = new Dictionary<char, double>(GetCipherTextLettersProbabilities(cipherText,
firstLetterLowerCase, lastLetterLowerCase, additionalLetterLowerCase));
            Dictionary<char, double> languageLettersIdealProbabilities = new Dictionary<char,
double>(GetLanguageIdealProbabilitiesDictionary(dictionaryLanguage));
            List<double> actualAndIdealProbabilitiesDifferenceSumList = new List<double>();
            int lettersCount = lastLetterLowerCase - firstLetterLowerCase + 1;
            //"+1" так как при вычитании количество букв получается на одну меньше реального
            for (int shift = 1; shift < lettersCount; shift++)
            {
                double actualAndIdealProbabilitiesDifferenceSum = 0;
                //выше сумма разницы "идеальной" и настоящей частоты каждой буквы для одного сдвига
                for (int letter = firstLetterLowerCase; letter <= lastLetterLowerCase; letter++)
                {
                    languageLettersIdealProbabilities.TryGetValue((char)letter, out double idealLetterProbability);
                    int newLetter = letter + shift;
                    if (newLetter > lastLetterLowerCase)
                    //проверка на то, уходит ли сдвиг за правую границу алфавита
                    {
                        newLetter = (firstLetterLowerCase - 1) + (shift - (lastLetterLowerCase - letter));
                        //"-1" используется для того, чтобы имитировать
                        //переход с последней буквы алфавита на первую букву алфавита, так как без "-1" буква преобразуется в следующую за
той,
                        //которая должна была бы быть
                    }
                    cipherTextLettersProbabilities.TryGetValue((char)newLetter, out double actualNewLetterProbability);
                    double actualAndIdealProbabilitiesDifference = Math.Abs(idealLetterProbability - actualNewLetterProbability);
                    actualAndIdealProbabilitiesDifferenceSum += actualAndIdealProbabilitiesDifference;
                }
                actualAndIdealProbabilitiesDifferenceSumList.Add(actualAndIdealProbabilitiesDifferenceSum);
            }
            int actualShift = actualAndIdealProbabilitiesDifferenceSumList.IndexOf(actualAndIdealProbabilitiesDifferenceSumList.Min()) + 1;
            //"+1" так как в List<> индексирование начинается с 0, а не 1
            return this.Decrypt(cipherText, actualShift, dictionaryLanguage);
        }

        /// <summary>
        /// Позволяет получить словарь с "идеальными" частотами букв для указанного языка.
        /// </summary>
        /// <param name="dictionaryLanguage">Язык словаря</param>
        /// <returns>Словарь с "идеальными" частотами букв</returns>
        private Dictionary<char, double> GetLanguageIdealProbabilitiesDictionary(string dictionaryLanguage)
        {
            if (string.Compare(dictionaryLanguage, "Cyrillic", StringComparison.OrdinalIgnoreCase) == 0)//если исходный текст на кириллице
            {
                return cyrillicLettersIdealProbabilities;
            }
            else
            {
                if (string.Compare(dictionaryLanguage, "Latin", StringComparison.OrdinalIgnoreCase) == 0)
                {
                    return latinLettersIdealProbabilities;
                }
                else
                {
                    throw new Exception("Несовместимый язык.");
                }
            }
        }

        /// <summary>
        /// Рассчитывает частоту встречи каждой буквы в зашифрованном тексте.
        /// </summary>
        /// <param name="cipherText">Зашифрованный текст</param>
        /// <returns>Словарь, где ключ - буква алфавита, а значение - частота его встречи в зашифрованном тексте</returns>
        private Dictionary<char, double> GetCipherTextLettersProbabilities(string cipherText, int firstLetterLowerCase,
             int lastLetterLowerCase, int additionalLetterLowerCase)
        {
            Dictionary<char, double> cipherTextLettersProbabilities = new Dictionary<char, double>();
            int cipherTextLettersCount = 0;
            foreach (char letter in cipherText.ToLower().ToCharArray())
            {
                if (char.IsLetter(letter) && letter != additionalLetterLowerCase)
                {
```

```
                cipherTextLettersCount++;
                //выше определяю количество букв (!!!) в тексте для последующего вычисления частоты
                if (cipherTextLettersProbabilities.ContainsKey(letter))
                {
                    cipherTextLettersProbabilities.TryGetValue(letter, out double letterCount);
                    cipherTextLettersProbabilities[letter] = (letterCount + 1);
                }
                else
                {
                    cipherTextLettersProbabilities.Add(letter, 1);
                }
            }
        }
        //выше рассчитываю сколько каждый раз каждая буква встретилась в тексте
        for (int letter = firstLetterLowerCase; letter <= lastLetterLowerCase; letter++)
        {
            if (cipherTextLettersProbabilities.TryGetValue((char)letter, out double letterCount))
            {
                double letterProbability = (letterCount / cipherTextLettersCount);
                cipherTextLettersProbabilities[(char)letter] = letterProbability;
            }
        }
        //выше рассчитываю частоту (вероятность) каждой буквы в тексте
        return cipherTextLettersProbabilities;
    }

    #endregion

    /// <summary>
    /// Устанавливает границы алфавита в зависимости от указанного языка.
    /// </summary>
    /// <param name="dictionaryLanguage">Язык алфавита</param>
    /// <param name="firstLetterUpperCase">Номер первой буквы алфавита верхнего регистра в десятичном виде в UNICODE</param>
    /// <param name="lastLetterUpperCase">Номер последней буквы алфавита верхнего регистра в десятичном виде в
UNICODE</param>
    /// <param name="firstLetterLowerCase">Номер первой буквы алфавита нижнего регистра в десятичном виде в UNICODE</param>
    /// <param name="lastLetterLowerCase">Номер последней буквы алфавита нижнего регистра в десятичном виде в
UNICODE</param>
    /// <param name="additionalLetterUpperCase">Дополнительная буква верхнего регистра(если она находится не рядом с основным
алфавитом в UNICODE)</param>
    /// <param name="additionalLetterLowerCase">Дополнительная буква нижнего регистра(если она находится не рядом с основным
алфавитом в UNICODE)</param>
    private void SetAlphabetBorders(string dictionaryLanguage, out int firstLetterUpperCase, out int lastLetterUpperCase,
        out int firstLetterLowerCase, out int lastLetterLowerCase, out int additionalLetterUpperCase, out int additionalLetterLowerCase)
    {
        if (string.Compare(dictionaryLanguage, "Cyrillic", StringComparison.OrdinalIgnoreCase) == 0)//если исходный текст на кириллице
        {
            firstLetterUpperCase = 1040;
            lastLetterUpperCase = 1071;
            firstLetterLowerCase = 1072;
            lastLetterLowerCase = 1103;
            additionalLetterUpperCase = 1025;//для букв "Ё" и "ё", так как они расположены в UNICODE отдельно от остальных букв
алфавита
            additionalLetterLowerCase = 1105;
        }
        else
        {
            if (string.Compare(dictionaryLanguage, "Latin", StringComparison.OrdinalIgnoreCase) == 0)//если исходный текст на латинице
            {
                firstLetterUpperCase = 65;
                lastLetterUpperCase = 90;
                firstLetterLowerCase = 97;
                lastLetterLowerCase = 122;
                additionalLetterUpperCase = default;
                additionalLetterLowerCase = default;
            }
            else
            {
                throw new Exception("Несовместимый язык.");
            }
        }
    }
  }
}
```