

Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего
профессионального образования

«Санкт-Петербургский национальный исследовательский университет
информационных технологий, механики и оптики»

Факультет безопасности информационных технологий

Дисциплина:

«Основы стеганографии»

Отчет по лабораторной работе №1

на тему

«Основы текстовой стеганографии»

Выполнила:

студентка гр. N3453

Чубаркина И.Р. _____

Проверил:

Калабишка М.М.. _____

Санкт-Петербург

2020

Цель работы:

Изучить основные методы текстовой стеганографии, путем их реализации в функционале самописной утилиты. Оценить эффективность методов и провести их сравнение.

Задачи:

1. Ознакомиться с теоретическим материалом;
2. Выбрать любой текст достаточного объема на английском языке для использования в качестве носителя;
3. Написать утилиту, отвечающую требованиям;
4. Использовать утилиту для встраивания произвольного слова на русском языке в стегоконтейнер любым из трех методов, и записать результаты ее работы в отчет;
5. Использовать утилиту для изъятия записанного на предыдущем этапе слова из стегоконтейнера и записать результаты работы в отчет;
6. В отчете оценить объем встраивания выбранного метода и сравнить его с другими методами. Сделать выводы;
7. В отчете провести оценку изъятия информации. Сделать выводы;
8. Сформулировать общий вывод.

Ход выполнения работы:

Были изучены три метода текстовой стеганографии: метод прямой замены, метод использования пробелов, метод использования специальных символов.

Создана консольная утилита (C#), реализующая:

1. Чтение текста-носителя из txt файла;
2. Вывод созданного стегоконтейнера в txt файл;
3. Вывод считанного сообщения в консоль;
4. Утилита умеет использовать 3 метода текстовой стеганографии.

Выбор используемого метода - вручную.

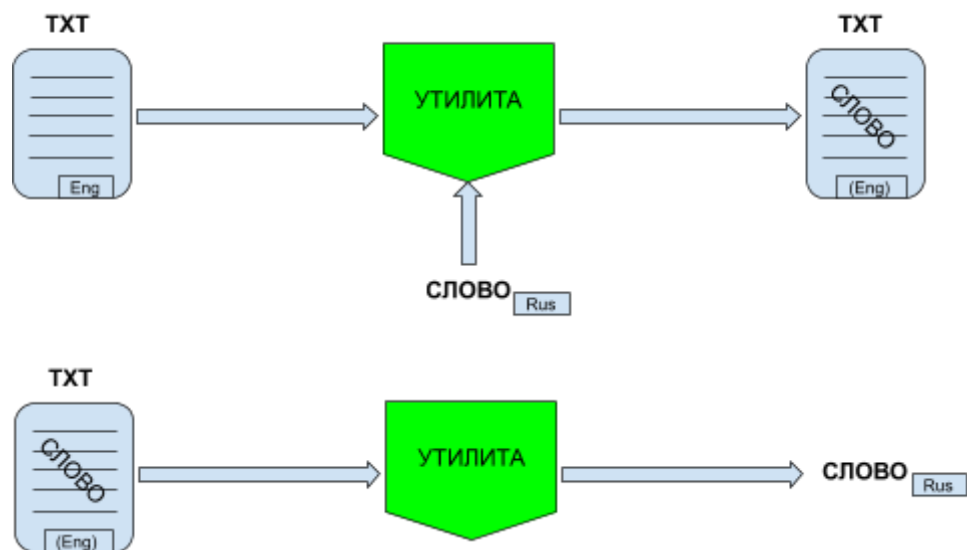


Рис.1 Функции утилиты

Передаваемое русское слово раскладывается на последовательность битов и передается с помощью носителя (исходного английского текста).

Точка входа в приложение - файл, представленные в приложении [1].

1) Метод использования специальных символов. В одной строке можно закодировать один бит передаваемого слова. Файл считывается построчно. Если мы хотим передать ноль - в конце строки специальные символы (`\r\n`) остаются нетронутыми. Если передаем единицу - пропускаем символ `'r'`. Таким образом, мы можем передать количество бит информации, равное числу строк в исходном тексте. Внешний вид текста не меняется. Объем файла уменьшается, так как мы обрезаем часть специальных символов.

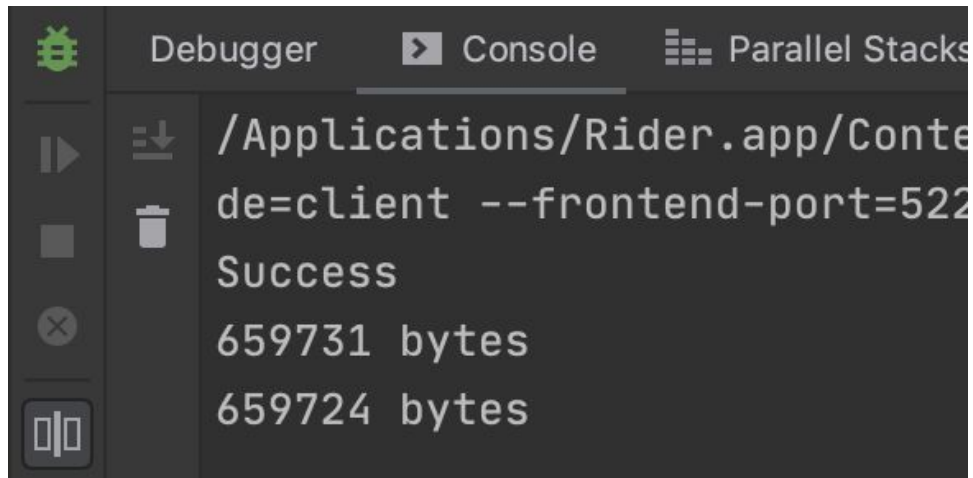
Код, отвечающий за реализацию данного метода, представлен в приложении [5].

Встраивание слова в стегоконтейнер.

Аргументы, переданные утилите:

```
Program arguments: encode 1 test.txt иришка test_result.txt
```

Результат работы:



The screenshot shows a debugger window with the 'Console' tab selected. The console output displays the command path, the command itself, and the results of the encoding process.

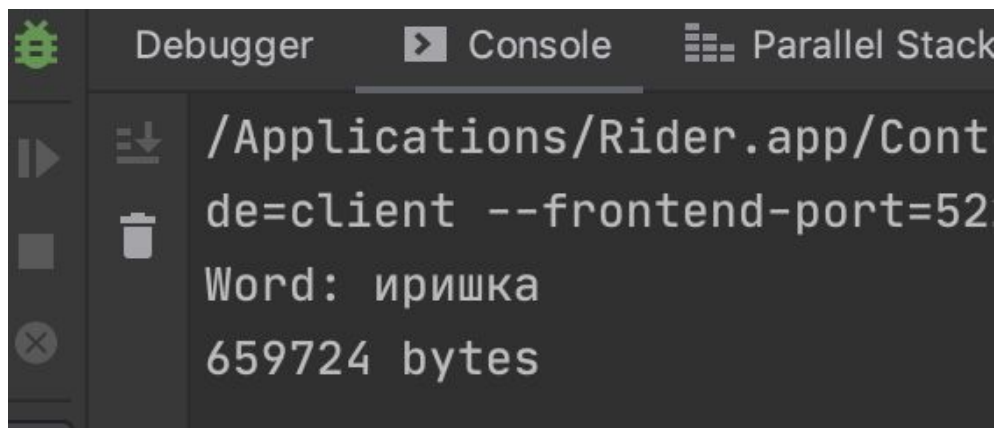
```
/Applications/Rider.app/Contents/MacOS/stealthy.exe  
de=client --frontend-port=5222  
Success  
659731 bytes  
659724 bytes
```

Изъятие информации из стегоконтейнера.

Аргументы, переданные утилите:

```
Program arguments: decode 1 test_result.txt
```

Результат работы:



The screenshot shows a debugger window with the 'Console' tab selected. The console output displays the command path, the command itself, the extracted word, and the file size.

```
/Applications/Rider.app/Contents/MacOS/stealthy.exe  
de=client --frontend-port=5222  
Word: иришка  
659724 bytes
```

2) Метод использования пробелов. В одной строке можно закодировать один бит передаваемого слова. Файл считывается построчно. Если мы хотим передать ноль - мы не добавляем пробел в конце строки. Если передаем единицу - добавляем один пробел. Таким образом, мы можем передать количество бит информации, равное числу строк в исходном тексте. Внешний вид текста не меняется. Объем файла увеличивается, так как мы добавляем пробелы .

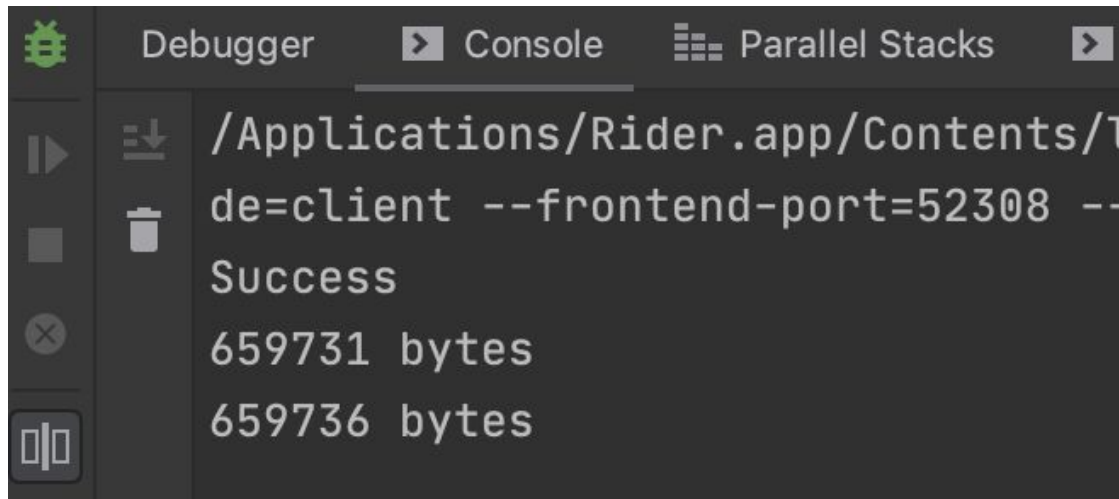
Код, отвечающий за реализацию данного метода, представлен в приложении [3].

Встраивание слова в стегоконтейнер.

Аргументы, переданные утилите:

```
Program arguments: encode 2 test.txt иришка test_result.txt
```

Результат работы:

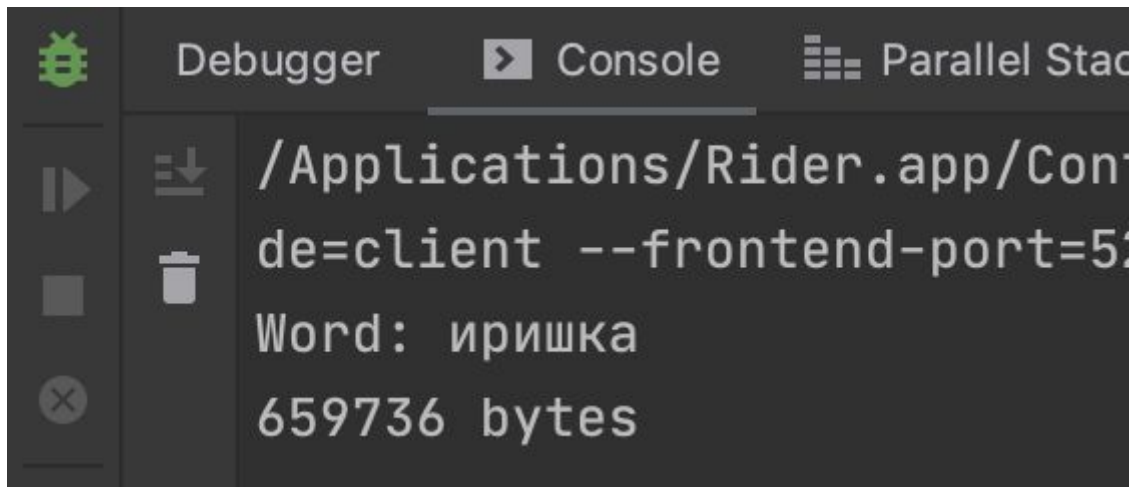


Изъятие:

Аргументы, переданные утилите:

```
Program arguments: decode 2 test_result.txt
```

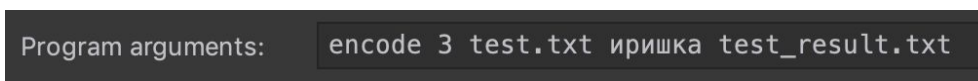
Результат работы:



```
Debugger Console Parallel Stacks  
/Applications/Rider.app/Contents/lib  
de=client --frontend-port=52324 --pl  
Word: иришка  
659736 bytes
```

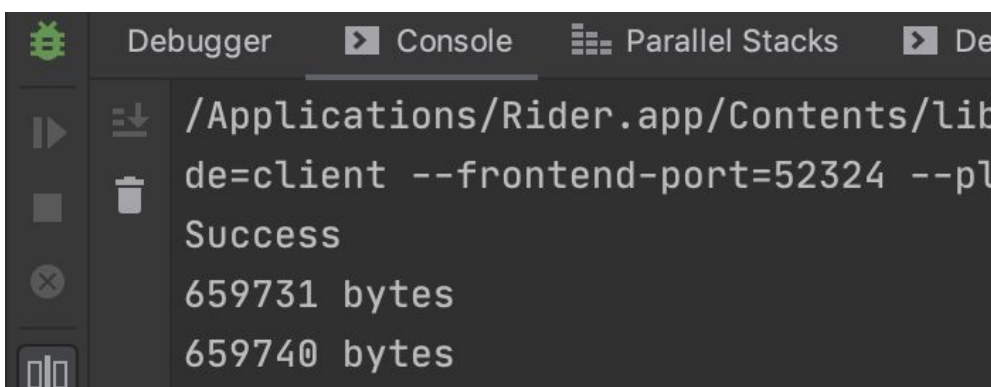
3) Метод замены. Используются буквы одинакового начертания - в русском и английском алфавитах. Если мы хотим передать единицу - мы заменяем английский символ на схожий с ним русский. Если передаем ноль - не заменяем. Также в контейнер передается объем (в двоичном виде) встраиваемого слова - некая служебная последовательность, чтобы было ясно, когда остановиться и перестать считывать символы. Внешний вид текста не меняется. Объем файла увеличивается, так как добавляется служебная последовательность. Код, отвечающий за реализацию данного метода, представлен в приложении [4].

Аргументы, переданные утилите:



```
Program arguments: encode 3 test.txt иришка test_result.txt
```

Результат работы:



```
Debugger Console Parallel Stacks De  
/Applications/Rider.app/Contents/lib  
de=client --frontend-port=52324 --pl  
Success  
659731 bytes  
659740 bytes
```

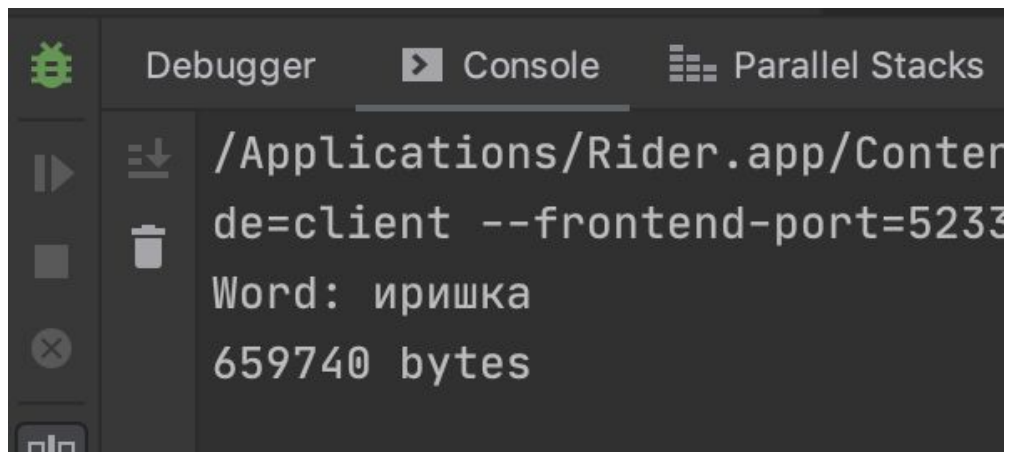
Изъятие:

Аргументы, переданные утилите:

Program arguments:

decode 3 test_result.txt

Результат работы:



The screenshot shows the Rider IDE interface with the 'Console' tab selected. The console displays the output of a command executed in the terminal. The output consists of three lines: a file path, a command, and the results of the command.

```
/Applications/Rider.app/Content/
de=client --frontend-port=5233
Word: иришка
659740 bytes
```

Вывод: в методе пробелов и специальных символов можно кодировать 1 бит на строку. То есть необходимо 5 строк для кодирования одной буквы. Если в строке около 70 символов, то на 350 символов исходного текста приходится одна буква из передаваемого слова. Если текст небольшой, его может не хватить. Третий метод эффективнее - буквы одинакового начертания встречаются в каждой строке по несколько раз и кодируются без пропусков. Минимальное количество символов для успешной передачи слова сокращается.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Шутько Надежда Павловна Алгоритмы реализации методов текстовой стеганографии на основе модификации пространственно-геометрических и цветовых параметров текста // Труды БГТУ. Серия 3: Физико-математические науки и информатика. 2016. №6 (188).
2. Шутько Надежда Павловна, Романенко Дмитрий Михайлович, Урбанович Павел Павлович Математическая модель системы текстовой стеганографии на основе модификации пространственных и цветовых параметров символов текста // Труды БГТУ. Серия 3: Физико-математические науки и информатика. 2015. №6 (179).

Приложение 1

Файл

```
using System;
using System.IO;
using Service.Services;
using Service.Utills;
using Service.Validators;

namespace Service
{
    class Program
    {
        static void Main(string[] args)
        {
            ConsoleValidator.ValidateArgsAndThrow(args);
            var sourceFilePath = ConsoleHelper.GetSourceFilePath(args);
            var method = args[1];

            if (args[0] == "encode")
            {
                var word = args[3];
                var wordAsBits = string.Empty;
                foreach (var character in word)
                {
                    wordAsBits +=
Constants.Constants.RussianCharactersToCodes[character];
                }
                var targetFilePath =
ConsoleHelper.GetTargetFilePath(args);
                HandleEncodeMode(method, sourceFilePath, targetFilePath,
wordAsBits, word);
            }
            else if (args[0] == "decode")
            {
                HandleDecodeMode(method, sourceFilePath);
            }
        }

        private static void HandleDecodeMode(string method, string
sourceFilePath)
        {

```

```

        try
        {
            string word;
            switch (method)
            {
                case "1":
                    word =
SpecialSymbolsService.Decode(sourceFilePath);
                    break;
                case "2":
                    word = EndSpacesService.Decode(sourceFilePath);
                    break;
                case "3":
                    word =
ReplacementService.Decode(sourceFilePath);
                    break;
                default:
                    throw new ArgumentException("Wrong method");
            }

            Console.WriteLine("Word: " + word);
            ConsoleHelper.ShowFileSize(sourceFilePath);
        }
        catch (Exception e)
        {
            Console.WriteLine(e.Message);
        }
    }

    private static void HandleEncodeMode(string method, string
sourceFilePath, string targetFilePath, string wordAsBits, string word)
    {
        try
        {
            switch (method)
            {
                case "1":
                    SpecialSymbolsService.Encode(sourceFilePath,
targetFilePath, wordAsBits);
                    break;
                case "2":

```

```

        EndSpacesService.Encode(targetFilePath,
sourceFilePath, wordAsBits);
        break;
    case "3":
        ReplacementService.Encode(targetFilePath,
sourceFilePath, wordAsBits, word);
        break;
    default:
        throw new ArgumentException("Wrong method");
    }
    Console.WriteLine("Success");
    ConsoleHelper.ShowFileSize(sourceFilePath);
    ConsoleHelper.ShowFileSize(targetFilePath);
}
catch (Exception e)
{
    Console.WriteLine(e.Message);
}
}
}
}

```

Приложение 2

Файл Constants.cs

```
using System.Collections.Generic;

namespace Service.Constants
{
    public static class Constants
    {
        public static readonly Dictionary<char, string>
RussianCharactersToCodes = new Dictionary<char, string>()
        {
            {'а', "00000"},
            {'б', "00001"},
            {'в', "00010"},
            {'г', "00011"},
            {'д', "00100"},
            {'е', "00101"},
            {'ж', "00110"},
            {'з', "00111"},
            {'и', "01000"},
            {'й', "01001"},
            {'к', "01010"},
            {'л', "01011"},
            {'м', "01100"},
            {'н', "01101"},
            {'о', "01110"},
            {'п', "01111"},
            {'р', "10000"},
            {'с', "10001"},
            {'т', "10010"},
            {'у', "10011"},
            {'ф', "10100"},
            {'х', "10101"},
            {'ц', "10110"},
            {'ч', "10111"},
            {'ш', "11000"},
            {'щ', "11001"},
            {'ъ', "11010"},
            {'ы', "11011"},
            {'ь', "11100"},
            {'э', "11101"},
            {'ю', "11110"},
            {'я', "11111"}
        };

        public static readonly Dictionary<char, char>
EnglishToRussianCharacters = new Dictionary<char, char>()
        {
            {'a', 'а'},
            {'e', 'е'},
            {'o', 'о'},
            {'p', 'р'},
            {'c', 'с'},
            {'y', 'у'},
            {'x', 'х'}
        };

        public const int CodeWidth = 5;
        public const string DefaultTargetFilePath =
@"../../../../../TextFiles/result.txt";
    }
}
```

```
        public const string PathToTextFolder = @"../../TextFiles/";  
    }  
}
```

Приложение 3

Файл EndSpaceService.cs

```
using System.IO;
using System.Text;
using Service.Utills;
using Service.Validators;

namespace Service.Services
{
    public static class EndSpacesService
    {
        public static void Encode(string targetFilePath, string
sourceFilePath, string word)
        {
            FileValidator.ValidateByLinesCountAndThrow(targetFilePath,
word);
            var source = File.ReadAllText(sourceFilePath,
Encoding.UTF8);

            using var streamWriter = File.CreateText(targetFilePath);
            var counter = 0;
            for (var i = 0; i < source.Length - 2; i++)
            {
                if (source[i] == ' ' && source[i + 1] == '\r' &&
source[i + 2] == '\n')
                {
                    //remove (skip) all end spaces
                    counter++;
                    continue;
                }
                if (source[i] == '\r')
                {
                    if (counter == word.Length)
                    {
                        //add space for end of word check (_\n instead
of \r\n)
                        streamWriter.Write(" ");
                        counter++;
                        continue;
                    }

                    if (counter < word.Length && word[counter] == '1')
                    {
                        //add space if 1 (do nothing if 0)
                        streamWriter.Write(" ");
                    }
                    counter++;
                }
                streamWriter.Write(source[i]);
            }
        }

        public static string Decode(string pathToFile)
        {
            var source = File.ReadAllText(pathToFile, Encoding.UTF8);
            var word = string.Empty;

            for (var i = 1; i < source.Length; i++)
            {
                //break when end of word
            }
        }
    }
}
```

```

        if (source[i - 1] == ' ' && source[i] == '\n')
            break;

        //detect 1
        if (source[i - 1] == ' ' && source[i] == '\r')
        {
            word += "1";
            continue;
        }

        //detect 0
        if (source[i-1] != ' ' && source[i] == '\r')
            word += "0";
    }

    return
DecodeHelper.DecodeWord(Constants.Constants.RussianCharactersToCodes, word);
}
}
}

```


Приложение 4

Файл ReplacementService.cs

```
using System;
using System.IO;
using System.Linq;
using System.Text;
using System.Text.RegularExpressions;
using Service.Utills;

namespace Service.Services
{
    public static class ReplacementService
    {
        public static void Encode(string targetFilePath, string
sourceFilePath, string bits, string word)
        {
            var source = File.ReadAllText(sourceFilePath,
Encoding.UTF8);
            var wordLength = word.Length;
            var helpCode = Convert.ToString(wordLength,
2).PadLeft(Constants.Constants.CodeWidth, '0');
            var helpCounter = 0;
            var wordCounter = 0;

            using var streamWriter = File.CreateText(targetFilePath);
            foreach (var character in source)
            {
                //write help word
                if
(Constants.Constants.EnglishToRussianCharacters.Keys.Contains(character) &&
helpCounter < Constants.Constants.CodeWidth)
                {
                    //replace if 1 (do nothing if 0)
                    var result = helpCode[helpCounter] == '1'
?
Constants.Constants.EnglishToRussianCharacters[character]
: character;
                    streamWriter.Write(result);
                    helpCounter++;
                    continue;
                }

                //write our word
                if
(Constants.Constants.EnglishToRussianCharacters.Keys.Contains(character) &&
wordCounter < bits.Length)
                {
                    //replace if 1 (do nothing if 0)
                    var result = bits[wordCounter] == '1'
?
Constants.Constants.EnglishToRussianCharacters[character]
: character;
                    streamWriter.Write(result);
                    wordCounter++;
                    continue;
                }
                streamWriter.Write(character);
            }

            public static string Decode(string pathToFile)
```

```

    {
        var source = File.ReadAllText(pathToFile, Encoding.UTF8);
        var word = string.Empty;
        var help = string.Empty;
        var wordCounter = 0;
        var helpCounter = 0;
        var wordLength = 0;

        foreach (var character in source)
        {
            if
(!Constants.Constants.EnglishToRussianCharacters.Keys.Contains(character) &&
!Constants.Constants.EnglishToRussianCharacters.Values.Contains(character))
continue;

            if (helpCounter < Constants.Constants.CodeWidth)
            {
                help += DecodeCharacter(character);
                helpCounter++;
                wordLength = Convert.ToInt32(help, 2) *
Constants.Constants.CodeWidth;
            }
            else if (wordCounter < wordLength && wordLength != 0)
            {
                word += DecodeCharacter(character);
                wordCounter++;
            }
        }

        return
DecodeHelper.DecodeWord(Constants.Constants.RussianCharactersToCodes, word);
    }

    private static string DecodeCharacter(char character)
    {
        //returns 1 if ru character, 0 if en
        const string pattern = @"[а-я]+";
        var match = Regex.Match(character.ToString(), pattern,
RegexOptions.IgnoreCase);
        return match.Success ? "1" : "0";
    }
}

```

Приложение 5

Файл SpecialSymbolsService.cs

```
using System.IO;
using System.Text;
using Service.Utills;
using Service.Validators;

namespace Service.Services
{
    public static class SpecialSymbolsService
    {
        public static void Encode(string sourceFilePath, string
targetFilePath, string word)
        {
            FileValidator.ValidateByLinesCountAndThrow(sourceFilePath,
word);
            var source = File.ReadAllText(sourceFilePath,
Encoding.UTF8);

            using var streamWriter = File.CreateText(targetFilePath);
            var counter = 0;
            foreach (var character in source)
            {
                if (character == '\r')
                {
                    if (counter == word.Length)
                    {
                        //add space for end of word check (_\n instead
of \r\n)
                        streamWriter.Write(" ");
                        counter++;
                        continue;
                    }

                    if (counter < word.Length && word[counter] == 'l')
                    {
                        //skip symbol (just \n instead of \r\n)
                        counter++;
                        continue;
                    }
                    counter++;
                }
                streamWriter.Write(character);
            }
        }

        public static string Decode(string pathToFile)
        {
            var source = File.ReadAllText(pathToFile, Encoding.UTF8);
            var word = string.Empty;

            for (var i = 1; i < source.Length; i++)
            {
                //break when end of word
                if (source[i - 1] == ' ' && source[i] == '\n')
                    break;

                //detect 0
                if (source[i - 1] == '\r' && source[i] == '\n')
                {
                    word += "0";
                }
            }
        }
    }
}
```

```
        continue;
    }

    //detect 1
    if (source[i - 1] != '\r' && source[i] == '\n')
        word += "1";
    }

    return
DecodeHelper.DecodeWord(Constants.Constants.RussianCharactersToCodes, word);
    }
}
```

Приложение 6

Файл ConsoleHelper.cs

```
using System;
using System.IO;

namespace Service.Utils
{
    public static class ConsoleHelper
    {
        //encode 1 test.txt иришка test_result.txt
        public static void ShowFileSize(string pathToFile)
        {
            var sourceFileSize = new FileInfo(pathToFile).Length;
            Console.WriteLine(sourceFileSize + " bytes");
        }

        public static string GetTargetFilePath(string[] args)
        {
            return args.Length == 5
                ? Constants.Constants.PathToTextFolder + args[4]
                : Constants.Constants.DefaultTargetFilePath;
        }

        public static string GetSourceFilePath(string[] args) =>
            Constants.Constants.PathToTextFolder + args[2];
    }
}
```

Приложение 7

Файл DecodeHelper.cs

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

namespace Service.Utills
{
    public static class DecodeHelper
    {
        public static string DecodeWord(Dictionary<char, string>
collection, string word)
        {
            var chars = Enumerable.Range(0, word.Length /
Constants.Constants.CodeWidth)
                .Select(i => word.Substring(i *
Constants.Constants.CodeWidth, Constants.Constants.CodeWidth))
                .Select(code => collection.FirstOrDefault(pair =>
pair.Value == code).Key);
            return string.Concat(chars);
        }
    }
}
```

Приложение 8

Файл ConsoleValidator.cs

```
using System;

namespace Service.Validators
{
    public static class ConsoleValidator
    {
        public static void ValidateArgsAndThrow(string[] args)
        {
            if (args.Length < 3)
            {
                throw new ArgumentException("Please set minimum 3
args");
            }

            if(args[0] == "encode" && args.Length == 3)
            {
                throw new ArgumentException("Please set minimum 4
args");
            }
        }
    }
}
```

Приложение 9

Файл FileValidator.cs

```
using System;
using System.IO;
using System.Linq;

namespace Service.Validators
{
    public static class FileValidator
    {
        public static void ValidateByLinesCountAndThrow(string textPath,
string word)
        {
            var lineCount = File.ReadLines(textPath).Count();
            if (lineCount < word.Length)
            {
                throw new ArgumentException("Too small text for
encoding, please try another method");
            }
        }
    }
}
```