МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное автономное образовательное учреждение
высшего образования «Санкт-Петербургский политехнический
университет Петра Великого»

Институт компьютерных наук и кибербезопасности

Высшая школа технологий искусственного интеллекта

Направление: 02.03.01 Математика и компьютерные науки

Отчет о выполнении практической работы №3
по дисциплине «Функциональное программирование»
Обработка файлов в Haskell
Вариант №2

Студент,
группа 5130201/20101 _____ Богданова Е.М.

Преподаватель _____ Моторин Д.Е.

«_____»_____ 2024 г.

Санкт-Петербург, 2024

Содержание

1 Введение

В данном отчете представлено описание практической работы, в которой реализуется программа на языке Haskell. Haskell — стандартизированный чистый функциональный язык программирования общего назначения. Работа выполнена с помощью утилиты Haskell Stack 3.1.1 - это инструмент для управления проектами на языке Haskell. Он упрощает сборку, зависимые библиотеки и создание пакетов, обеспечивая воспроизводимость окружения разработки.

2 Постановка задачи

Создать проект в stack. Все чистые функции записать в библиотеку Lib.hs и ограничить доступ к вспомогательным функциям. Использовать do-нотацию для работы с внешними файлами. Найти портрет указанного человека:

Ляпунов, Александр Михайлович

Перевести изображение в формат .bmp (24-разрядный), при необходимости изменить ширину и высоту изображения без искажений. Сохранить в файл формата .txt фрагмент биографии (не менее 1000 символов без пробелов, текст не должен обрываться на середине слова или предложения). Закодировать текст в изображение методом:

Шифром Цезаря. Смещение задается пользователем

Ключ к шифру записывается в имя файла. Написать функцию расшифровывающую текст из изображения используя ключ из имени файла и сохраняющую результат в отдельный текстовый файл.

Создать функции шифрующие текст в последний бит каждого байта, последние два бита каждого байта, ..., все биты в байте. В отчете привести примеры искажений изображения.

3 Математическое описание

3.1 Формат BMP

BMP (от англ. Bitmap Picture) — это формат хранения растровой графики, который представляет изображение в виде двумерного массива пикселей, разработанный компанией Microsoft. Файлы формата BMP могут иметь расширения .bmp, .dib и .rle..

1. Изображение представлено как массив пикселей P:

   P: R x C → (R, G, B) , где:

   - R — количество строк (высота изображения),

   - C — количество столбцов (ширина изображения),

   - (R, G, B) — цвет пикселя, определяемый значениями красного, зеленого и синего каналов.

2. Заголовок файла BMP содержит метаданные, такие как:

   - Размер изображения в байтах,

   - Разрешение (ширину и высоту),

   - Количество бит на пиксель (обычно 24 или 32 бита). В случае лабораторной работы используется 24-разрядный формат.

3. Изображение хранится в построчном порядке, начиная с нижней строки (модели координат).

Это позволяет дескриптору BMP восстанавливать и отображать изображение на экране.

3.2 Стеганография

Стеганография (от греч. [стеганос] «скрытый» + [графо] «пишу»; букв. «тайнопись») — способ передачи или хранения информации с учётом сохранения в тайне самого факта такой передачи (хранения). В отличие от криптографии, которая скрывает содержимое тайного сообщения, стеганография скрывает сам факт его существования. Как правило, сообщение будет выглядеть как что-либо иное, например, как изображение, статья, список покупок, письмо или судоку. Стеганографию обычно используют совместно с методами криптографии, таким образом, дополняя её.

Стеганографию можно описать следующим образом:

Исходный сигнал:
Пусть S - это исходный сигнал (например, изображение, звук), который можно представить как набор данных $S = s_1, s_2, \ldots, s_n$, где каждый s_i - это значение сигнала в i-й точке.

Скрытый сигнал:
Пусть H - это скрытый сигнал (например, текст), который можно представить как

набор данных $H = h_1, h_2, \ldots, h_m$, где каждый h_i - это значение скрытого сигнала в i-й точке.

Встраивание:
Встраивание скрытого сигнала в исходный сигнал можно описать как функцию $f(S, H) = S'$, где S' - это модифицированный исходный сигнал, содержащий скрытую информацию.

Извлечение:
Извлечение скрытого сигнала из модифицированного исходного сигнала можно описать как функцию $g(S') = H'$, где H' - это извлеченная скрытая информация.

Основные методы стеганографии:

LSB (Least Significant Bit): скрытая информация встраивается в младшие биты пикселей изображения, не вызывая заметных визуальных изменений.

DCT (Discrete Cosine Transform): скрытая информация встраивается в коэффициенты DCT изображения, которые менее заметны для человеческого глаза.

Spread Spectrum: скрытая информация распространяется по всему исходному сигналу, делая ее менее заметной.

3.3 Шифр Цезаря

Шифр Цезаря (лат. Notae Caesarianae), также известный как шифр сдвига или код Цезаря — разновидность шифра подстановки, в котором каждый символ в открытом тексте заменяется символом, находящимся на некотором постоянном числе позиций левее или правее него в алфавите (так, в шифре со сдвигом вправо на 3, А была бы заменена на Г, Б станет Д, и так далее). Шифр был назван в честь римского полководца Гая Юлия Цезаря, использовавшего его для секретной переписки со своими военачальниками.

Шаг шифрования, выполняемый шифром Цезаря, часто включается как часть более сложных схем, таких как шифр Виженера. Как и все моноалфавитные шифры, шифр Цезаря легко взламывается и не имеет почти никакого применения на практике. Тем не менее, он считается одним из самых простых и наиболее широко известных методов шифрования.

4 Особенности реализации

4.1 Ход программы

Реализация программы подразумевает несколько смысловых частей:

1. Считывание текста биографии из файла с расширением .txt и изображения деятеля в формате .bmp.
2. Шифрование текста с помощью шифра Цезаря со смещением, введенным пользователем.
3. Встраивание зашифрованного текста в изображение формата .bmp и именование файла ключом, который является сдвигом в шифре Цезаря.
4. Извлечение текста из файла изображения.
5. Расшифровка текста с помощью обратного шифра Цезаря со смещением, считанным из названия файла.
6. Сравнение результатов для проверки корректности работы программы.

4.2 Функции

4.2.1 Основная функция работы программы

- main :: IO ()
  Эта функция является основной и управляет ходом работы всей программы. В первую очередь считываются аргументы командной строки: inputTextFile - путь к файлу с текстом, который нужно зашифровать; inputBmpFile - путь к файлу с изображением, в которое будет встроена зашифрованная информация; shiftStr - строка, представляющая сдвиг для шифра Цезаря; bitsPByte - строка, представляющая количество бит на байт, которое будет использоваться для встраивания информации в изображение. Затем проверяется корректность аргументов, что bitsPerByte находится в диапазоне от 1 до 8 (так как в одном байте изображения 8 бит) и что shift находится в диапазоне от 1 до 5. Далее считывается содержимое файла inputTextFile и вызывается функция encryptByCaesar, которая будет описана далее, которая шифрует текст шифром Цезаря с заданным сдвигом shiftStr. Затем зашифрованный текст записывается в файл. После этого вызывается функция embedTextInBMP, которая встраивает зашифрованный текст в изображение inputBmpFile, изображение сохраняется в новый файл. После этого вызывается функция decodeTextFromBMP, которая извлекает текст из изображения и декодирует его обратным шифром Цезаря. Расшифрованная строка записывается в новый файл. Последний шаг - сравнение первоначального текста и расшифрованного из изображения, если результаты совпадают, то программа сработала корректно, если не совпадают - не корректно.

Код приведен в «Приложение А».

4.2.2 Шифр Цезаря

- encryptByCaesar :: Int -> String -> String
  Эта функция для шифрования строки, принимает сдвиг shift и строку str. Применяет с помощью map к каждому символу функцию caesarCipher и возвращает зашифрованную строку

- caesarCipher :: Int -> Char -> Char
  Эта функция реализует алгоритм шифра Цезаря. Принимает - смещение и символ, который нужно зашифровать, возвращает - зашифрованный символ. Если символ между 'a' и 'z' или 'A' и 'Z', то сдвигаем символ: fromEnum c - возвращает целое число, соответствующее этому символу в таблице ASCII, из него вычитается ASCII-код символа 'a'/'A', чтобы считать от нуля, затем прибавляется сдвиг shift, после этого берется остаток от деления на 26 (количество букв английского алфавита), чтобы не выйти за пределы алфавита, затем прибавляется ASCII-код символа 'a'/'A', с помощью функции toEnum возвращается символ по итоговому ASCII-коду.

Код приведен в «Приложение Б».

4.2.3 Кодирование текста в изображение

- embedTextInBMP :: FilePath -> String -> Int -> Int -> IO ()
  Эта функция для встраивания текста в Bitmap файл. Принимает: имя входного файла; текст, который нужно встроить; количество бит на байт, ключ - смещение в шифре Цезаря. С помощью функции openBinaryFile открываем файл в бинарном режиме, принимая путь к файлу inputFile и режим чтения ReadMode. Если файл успешно открыт, openBinaryFile возвращает дескриптор файла, который затем присваивается переменной handle. Затем с помощью функции B.hGetContents (работаем с Data.ByteString под псевдонимом B) из handle считываем все содержимое файла в строку contents. Устанавливаем размер заголовка bmpHeaderSize, с помощью B.drop отбрасываем от contents заголовок и передаем в bmpData, преобразовываем bmpData в список байтов bmpBytes. Вызываем ф-ю textToBits и преобразовываем текст (который нужно встроить) в список битов. Вызываем ф-ю embedBits и встраиваем биты в байты, записываем результат в modifiedBytes. Задаем имя нового файла, название файла формата - <ключ.bmp>. Записываем в новый файл измененные байты.

- textToBits :: String -> [Bool]
  Функция для преобразования текста в список битов. Сначала вызывается fromEnum, которая преобразует символ в его ASCII-код, а затем результат передается в intToBits, которая преобразует это числовое значение в список битов, затем добавляем список из 32 [True] к списку, полученному от concatMap, для обозначения конца последовательности битов. concatMap - функция высшего порядка, которая принимает: функцию и список, применяет функцию к каждому элементу списка и затем конкатенирует все полученные списки в один. Оператор . позволяет комбинировать две функции так, чтобы результат одной функции передавался в качестве аргумента другой функции.

- embedBits :: [Word8] -> [Bool] -> Int -> [Word8]
  Ф-я для встраивания битов текста в массив байтов файла. Принимает: список байтов, в который будут встроены биты; список битов, которые нужно встроить; количество битов, которое будет встроено в каждый байт и возвращает список байтов после встраивания битов. Вычисляется длина последовательности битов bitsLength, вычисляется количество байтов availableBytes, доступных для встраивания с помощью функции length. Вычисляется максимальное количество битов maxBits, которые можно встроить в доступные байты как количество байтов на количество битов на байт. Затем берется не более maxBits битов из bits. Если bits короче maxBits, последовательность embeddedBits дополняется значениями False до длины maxBits, это необходимо для заполнения оставшегося пространства в байтах, куда не поместилась исходная информация, предотвращая искажение данных. Затем с помощью функции chunksOf разбиваем список битов на подсписки заданного размера bitsPerByte, каждый подсписок будет встраиваться в отдельный байт с помощью функции embedByte. zipWith - функция, которая принимает функцию embedByte, два списка в качестве параметров список подсписков bits и список bytes и bitsPerByte, она объединяет эти два списка, применяя embedByte между соответствующими элементами.

- embedByte :: [Bool] -> Int -> Word8 -> Word8
  Функция принимает кусок битов bitChunk, количество битов на байт bitsPerByte и исходный байт byte. Она встраивает эти биты в байт, модифицируя его, в соответствии с порядком битов в байте (от младшего к старшему) и результатом сравнения paddedBits, таким образом модифицируя исходный байт. Функция возвращает измененный байт.

- chunksOf :: Int -> [a] -> [[a]]
  Функция для разделения списка на подсписки фиксированной длины, если список пустой, то возвращает пустой список, иначе рекурсивно: take - берет первые n элементов из списка xs и вызывается функция chunksOf.

  Код приведен в «Приложение Б».

4.2.4 Декодирование текста из изображения

- decodeTextFromBMP :: FilePath -> Int -> IO ()
  Эта функция предназначена для извлечения зашифрованного текста из файла формата .bmp. Функция принимает путь к входному BMP файлу и количество бит на байт, файл открывается в бинарном режиме для чтения, содержимое файла читается в виде байтового массива. Затем первые 54 байта - заголовок - отбрасываются, так как они не содержат закодированного текста. Остальные байты распаковываются в список байтов. Затем вызывается функция extractBits, которая извлекает отдельные подсписки битов размера указанного количества бит на байт. Затем извлекаем текстовые биты extractTextBits до маркера завершения текста. С помощью функции bitsToText извлеченные текстовые биты преобразуются в строку. Также считывается сдвиг для шифра Цезаря, который берется из имени файла изображения. Полученный текст расшифровывается с помощью функции decryptByCaesar, используя вычисленный сдвиг. Расшифрованная строка записывается в новый файл.

- extractBits :: [Word8] -> Int -> [Bool]
  Функция преобразует список байтов в список логических значений. Принимает: список байтов bytes, из которого будут извлекаться биты; количество бит bitsPerByte, которые нужно извлечь из каждого байта. Делим список байтов на подсписки длины bitsPerByte и формируем булевые значения для каждого бита.

- extractTextBits :: [Bool] -> [Bool] Функция извлекает биты из списка, пока не встретит 32 последовательных True. Как только это происходит, она возвращает список собранных битов в порядке их появления. Принимает список булевых значений bits и вызывает вспомогательную функцию go, передавая ей этот список и пустой список []. Внутри go происходит: если список xs пустой, то функция возвращает reverse acc, то есть список, перевернутый для получения значений в исходном порядке. Если xs не пустой: проверяется, равен ли подсписок, состоящий из первых 32 элементов xs, списку, состоящему из 32 True. Если это так, функция завершает выполнение и возвращает перевернутый acc. Если нет, go рекурсивно вызывается снова, с пропущенным первым элементом списка (через tail xs) и добавлением первого элемента xs (через head xs) в acc.

- bitsToText :: [Bool] -> Int-> String Функция принимает список битов и разбивает его на байты, преобразуя каждый байт в символ. Результатом будет строка, представляющая текстовое представление данного списка битов.

- decryptByCaesar :: Int -> String -> String Функция предназначена для расшифровки строки, зашифрованной шифром Цезаря. Вызывается функция encryptByCaesar с отрицательным смещением для каждого символа строки.

Код приведен в «Приложение Б».

5 Результаты работы

Входные параметры:

- входной текстовый файл - D:/LabHaskell/lab2/bio.txt
- входной файл изображения - D:/LabHaskell/lab2/img.bmp
- сдвиг - 3;
- количество бит на байт - 5.

Исходный файл текста представлен на Рис. 1.

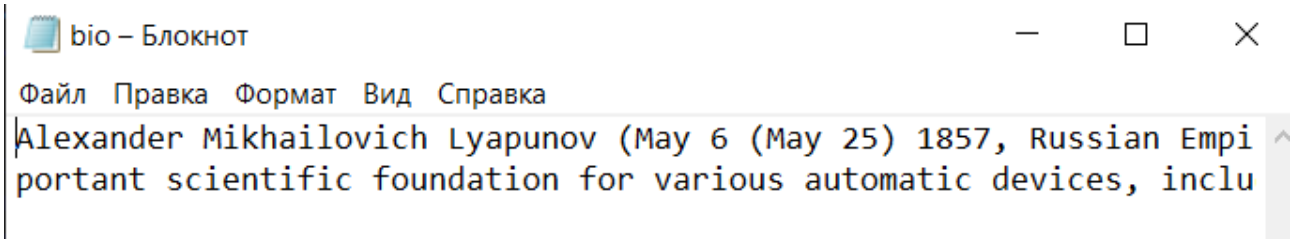

Рис. 1. Исходный файл текста

Зашифрованный текстовый файл представлен на Рис. 2.

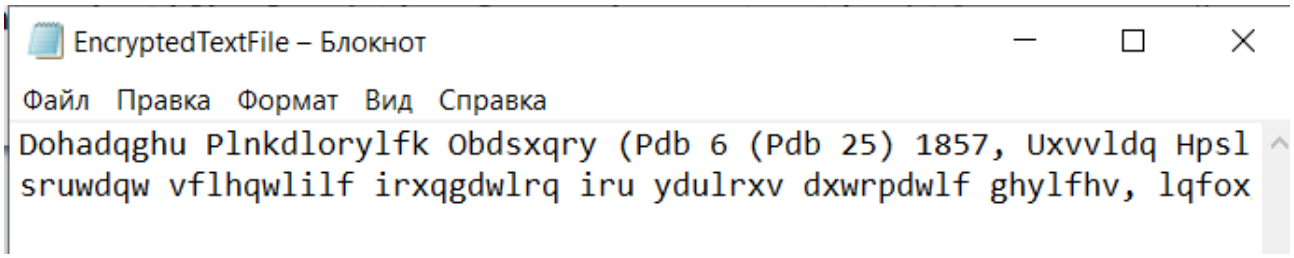

Рис. 2. Зашифрованный текстовый файл

Расшифрованный текстовый файл представлен на Рис. 3.

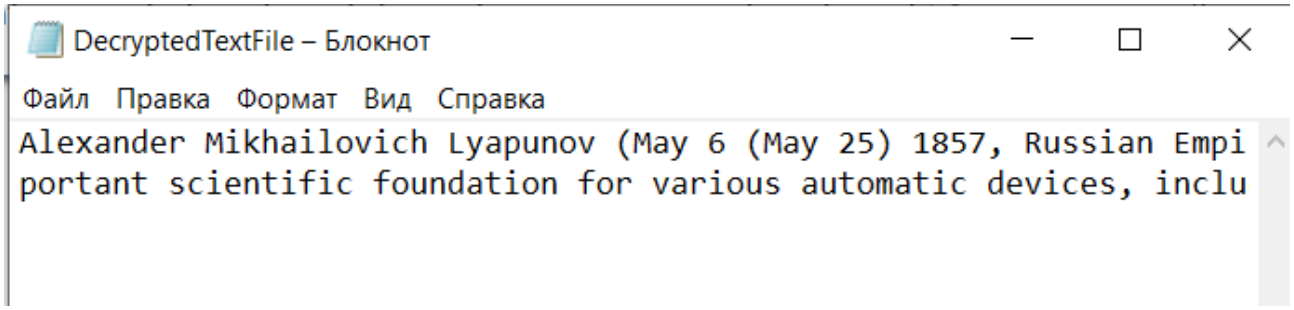

Рис. 3. Расшифрованный текстовый файл

Оригинальное изображение представлено на Рис. 4.

Рис. 4. Оригинальное изображение

Изображение при встраивании текста в 1 бит представлено на Рис. 5.

Рис. 5. Изображение при встраивании текста в 1 бит

Изображение при встраивании текста в 2 бита представлено на Рис. 6.

Рис. 6. Изображение при встраивании текста в 2 бита

Изображение при встраивании текста в 3 бита представлено на Рис. 7.

Рис. 7. Изображение при встраивании текста в 3 бита

Изображение при встраивании текста в 4 бита представлено на Рис. 8.

Рис. 8. Изображение при встраивании текста в 4 бита

Изображение при встраивании текста в 5 бит представлено на Рис. 9.

Рис. 9. Изображение при встраивании текста в 5 бит

Изображение при встраивании текста в 6 бит представлено на Рис. 10.

Рис. 10. Изображение при встраивании текста в 6 бит

Изображение при встраивании текста в 7 бит представлено на Рис. 11.

Рис. 11. Изображение при встраивании текста в 7 бит

Изображение при встраивании текста в 8 бит представлено на Рис. 12.

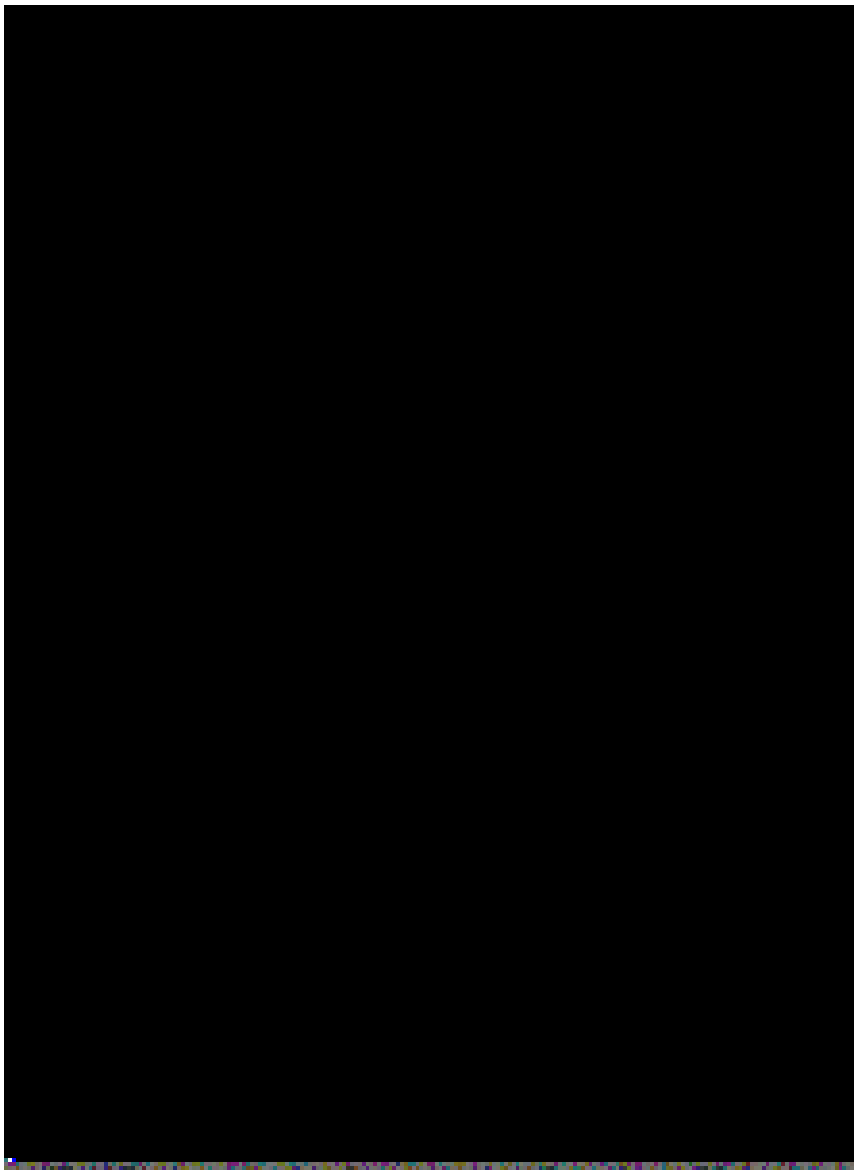

Рис. 12. Изображение при встраивании текста в 8 бит

Был проведен эксперимент искажения изображения для определенных входных данных. Как видно из примеров выше, при увеличении количества бит, отводимых под текст, качество изображения ухудшается.

Заключение

В ходе выполнения практической работы была реализована программа на Haskell. Был создан проект в Haskell Stack. Все функции записаны в библиотеку Lib.hs. Был найден портрет Ляпунова Александра Михайловича. Изображение было переведено в формат .bmp (24-разрядный). В файл формата .txt был сохранен фрагмент биографии. Текст был закодирован шифром Цезаря в изображение. Ключ к шифру был записан в имя нового файла. Написана функция расшифровывающая текст из изображения, использован ключ из имени файл, сохраняющая результат в отдельный текстовый файл. Созданы функции шифрующие текст в последний бит каждого байта, последние два бита каждого байта, ..., все биты в байте.

Приложение A. Main

```
module Main (main) where
import System.Environment (getArgs)
import System.Exit (exitFailure)
import Control.Monad
import Lib

main :: IO ()
main = do
    args <- getArgs
    case args of
        [inputTextFile, inputBmpFile, shiftStr, bitsPByte] -> do
            let shift = read shiftStr :: Int  -- Преобразуем строку в целое число
            let bitsPerByte = read bitsPByte :: Int  -- Преобразуем строку в целое число
            -- Проверка значений
            when (bitsPerByte < 1 || bitsPerByte > 8) $ do
                putStrLn "Ошибка: bitsPerByte должно быть от 1 до 8."
                exitFailure
            when (shift < 1 || shift > 5) $ do
                putStrLn "Ошибка: shift должно быть от 1 до 5."
                exitFailure
            content <- readFile inputTextFile
            let encryptedContent = encryptByCaesar shift content
            let outputEncryptedTextFile = "D:/LabHaskell/lab2/EncryptedTextFile.txt"
            writeFile outputEncryptedTextFile encryptedContent
            putStrLn "Файл успешно зашифрован шифром Цезаря"
            embedTextInBMP inputBmpFile encryptedContent bitsPerByte shift
            putStrLn "Текст зашифрован в изображение"
            decodeTextFromBMP ("D:/LabHaskell/lab2/" ++ (show shift) ++ ".bmp")
                bitsPerByte
            putStrLn "Файл успешно расшифрован!"
            decryptedStr <- readFile "D:/LabHaskell/lab2/DecryptedTextFile.txt"
            --let encryptedStr = readFile "D:/LabHaskell/lab2/EncryptedTextFile.txt"
            if content == decryptedStr
                then putStrLn "Файлы до шифровки и после расшифровки совпадают!"
                else putStrLn "Файлы до шифровки и после расшифровки не совпадают!"
        _ -> putStrLn "Аргументы некорректные, формат: программа <входной текстовый файл>
             <входной файл изображения> <сдвиг> <кол-во бит на байт>"
```

Приложение Б. Lib

```haskell
module Lib where

import System.IO
import Data.Bits
import Data.Char
import Data.Word
import qualified Data.ByteString as B
import Data.Word (Word8)
import Numeric (readInt)
import System.FilePath (takeBaseName)

decryptByCaesar :: Int -> String -> String
decryptByCaesar shift str = map (caesarCipher (-shift)) str

bitsToInt :: [Bool] -> Int
bitsToInt bits = foldl (\acc b -> acc * 2 + if b then 1 else 0) 0 bits

bitsToText :: [Bool] -> Int-> String
bitsToText bits bitsPerByte = map (toEnum . bitsToInt) (chunksOf 8 bits)

extractBits :: [Word8] -> Int -> [Bool]
extractBits bytes bitsPerByte =
     concatMap (\byte ->  [testBit byte i | i <- reverse[0 .. (bitsPerByte - 1)]]) bytes

extractTextBits :: [Bool] -> [Bool]
extractTextBits bits = go bits []
  where
    go [] acc = reverse acc
    go xs acc
      | take 32 xs == replicate 32 True = reverse acc  -- Если нашли 8 нулей, завершаем
      | otherwise = go (tail xs) (head xs : acc)

decodeTextFromBMP :: FilePath -> Int -> IO ()
decodeTextFromBMP inputFile bitsPerByte = do
    handle <- openBinaryFile inputFile ReadMode
    contents <- B.hGetContents handle
    let bmpHeaderSize = 54
        bmpData = B.drop bmpHeaderSize contents
        bmpBytes = B.unpack bmpData
        extractedBits = extractBits bmpBytes bitsPerByte
        textBits = extractTextBits extractedBits  -- Извлекаем текстовые биты до маркера завершения
        decodedText = bitsToText textBits bitsPerByte
        shift = read (takeBaseName inputFile) :: Int
        decodedCaesarStr = decryptByCaesar shift decodedText
    let outputDecryptedFile = "D:/LabHaskell/lab2/DecryptedTextFile.txt"  -- Формируем имя выходного файла
    writeFile outputDecryptedFile decodedCaesarStr  -- Записываем строку в файл
    hClose handle

caesarCipher :: Int -> Char -> Char
caesarCipher shift c
    | 'a' <= c && c <= 'z' = toEnum ((fromEnum c - fromEnum 'a' + shift) `mod` 26 + fromEnum 'a')
    | 'A' <= c && c <= 'Z' = toEnum ((fromEnum c - fromEnum 'A' + shift) `mod` 26 + fromEnum 'A')
    | otherwise = c

encryptByCaesar :: Int -> String -> String
encryptByCaesar shift str = map (caesarCipher shift) str

intToBits :: Int -> [Bool]
intToBits n = reverse [testBit n i | i <- [0..7]]

textToBits :: String -> [Bool]
textToBits text = concatMap (intToBits . fromEnum) text ++ replicate 32 True  -- Добавляем False как окончательный маркер

```

```haskell
chunksOf :: Int -> [a] -> [[a]]
chunksOf _ [] = []
chunksOf n xs = take n xs : chunksOf n (drop n xs)

embedByte :: [Bool] -> Int -> Word8 -> Word8
embedByte bitChunk bitsPerByte byte = foldl (\b (bit, pos) -> if bit then setBit b pos
    else clearBit b pos) byte (zip paddedBits positions)
  where
    positions = reverse [0 .. bitsPerByte - 1]
    -- Если bitChunk меньше, чем bitsPerByte, добавляем нули
    paddedBits = take bitsPerByte (bitChunk ++ repeat False)

embedBits :: [Word8] -> [Bool] -> Int -> [Word8]
embedBits bytes bits bitsPerByte =
    let bitsLength = length bits
        availableBytes = length bytes
        maxBits = availableBytes * bitsPerByte
        embeddedBits = take maxBits bits ++ repeat False  -- Заполнение нулями
        modifiedBytes = zipWith3 embedByte (chunksOf bitsPerByte embeddedBits) (take
            availableBytes (repeat bitsPerByte)) bytes
    in modifiedBytes

embedTextInBMP :: FilePath -> String -> Int -> Int -> IO ()
embedTextInBMP inputFile text bitsPerByte shift = do
    handle <- openBinaryFile inputFile ReadMode
    contents <- B.hGetContents handle
    let bmpHeaderSize = 54
        bmpData = B.drop bmpHeaderSize contents
        bmpBytes = B.unpack bmpData -- Преобразуем ByteString в список байтов
        bits = textToBits text
        modifiedBytes = embedBits bmpBytes bits bitsPerByte
        restBytes = drop (length modifiedBytes) bmpBytes
    let newOutputFile = "D:/LabHaskell/lab2/" ++ show shift ++ ".bmp"
   -- Открываем файл для записи в бинарном режиме
    outputHandle <- openBinaryFile newOutputFile WriteMode
    -- Записываем заголовок
    B.hPut outputHandle (B.take bmpHeaderSize contents)
    -- Записываем измененные данные
    B.hPut outputHandle (B.pack modifiedBytes)
    hClose handle
    hClose outputHandle
```

Список использованной литературы

[1] Уилл Курт Программируй на Haskell / пер. с англ. Я.О. Касюлевича, А.А. Романовского и С.Д. Степаненко; под ред. В.Н. Брагилевского. – М.: ДМК Пресс, 2019. — 648 с.