

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ  
ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО

Институт компьютерных наук и кибербезопасности

Высшая школа технологий искусственного интеллекта

Направление: 02.03.01 Математика и компьютерные науки

Отчет по практическому заданию №3

по дисциплине: «Функциональное программирование»

Студент:  
группы 5130201/20102  
\_\_\_\_\_ Салимли А.

Преподаватель:  
к.т.н.  
\_\_\_\_\_ Моторин Д.Е.

«\_\_\_\_ » 2024 г.

Санкт-Петербург, 2024 г.

# **Содержание**

<b><u>Введение</u></b>	3
1. <b><u>Постановка задачи</u></b>	4
2. <b><u>Практическое задание 3</u></b>	5
2.1 <u>Особенности реализации</u>	5
2.2 <u>Шифрование с использованием кодового слова</u>	6
2.3 <u>Декодирование</u>	6
2.4 <u>Код программы</u>	7
3. <b><u>Результаты</u></b>	12
4. <b><u>Заключение</u></b>	15
5. <b><u>Список литературы</u></b>	16
6. <b><u>Приложение А</u></b>	17

## **Введение**

В данном отчете, описан результат выполнения практического задания №3, в котором необходимо создать проект в stack, закодировать текст биографии предоставленного ученого, в изображение, а затем реализовать декодирование текста из изображения.

В ходе выполнения практической работы, реализован на языке Haskell:

Проект в stack в котором разработана программа кодирующая текст в изображение, и декодирующее текст из изображения. Которая принимает на вход изображение, текстовый файл, кодовое слово, количество желаемых битов кодировки. Декодирование принимает на вход закодированное изображение.

Проект был реализован в интегрируемой среде разработки - Visual Studio Code 1.94, на языке Haskell 9.4.8. Расширение основных программ - .hs, структуры проекта: .cabal, .yaml.

## **1 Постановка задачи**

В ходе прохождения практического задания №3, необходимо реализовать проект в stack на языке Haskell.

1. Принимает 24-разрядное изображение формата .bmp, сохраняет файл формата .txt с фрагментом биографии, из не менее 1000 символов без пробелов, текст так же не должен обрываться на середине слова или предложения. Текст закодировать в изображение предоставленным методом.
2. Создать функцию шифрующую текст в последний бит каждого байта, два последних бита каждого байта ... все биты в байте.
3. Создать функцию декодирующую текст из изображения используя ключ из имени файла и сохраняющую результат в отдельный .txt файл.

**Ограничения:** Все чистые функции в Lib.hs, использовать do-нотацию для работы с внешними файлами.

Ученный на фото: Лобачевский Николай Иванович.

Метод шифрования: Шифр с использованием кодового слова (Задается пользователем)

## 2 Практическое задание

### 2.1 Особенности реализации

Для начала работы необходимо найти изображение портрета Лобачевского Николая

Ивановича. По заданию изображение должно быть в .bmp формате (24-разрядный).

Необходимо так же написать биографию Лобачевского Н.И. длиною не менее 1000 символов без учета пробелов.

Ниже представлены текст и изображение.



Nikolai Ivanovich Lobachevsky was born on December 1, 1792, in Makaryev, Nizhny Novgorod province, Russia. He was an outstanding Russian mathematician and is often regarded as the creator of non Euclidean geometry. His work and ideas laid the foundation for hyperbolic geometry, which profoundly impacted mathematics and science. In 1807, he enrolled at Kazan University, where he studied under Professor Martin Bartels, initially focusing on mathematics and later on physics. In 1826, he published his groundbreaking work, *On the Principles of Geometry*, which outlined the fundamentals of a new geometric system. Lobachevsky proposed that alternative geometries could exist without following Euclid's parallel postulate, which revolutionized the mathematical world. His contributions gained recognition only after his death, and today, Lobachevsky's name is associated with one of the most influential theories in mathematics. Besides his theoretical work, Lobachevsky served as rector at Kazan University, where he actively promoted education and science in Russia. His ideas inspired significant advances in physics and mathematics, helping form modern concepts of space. Lobachevsky passed away in 1856, leaving behind a rich scientific legacy that continues to inspire scholars worldwide.

Рис.1 Изображение Лобачевского Н.И.  
формат .bmp

Биография длиною не менее 1000 символов

Идея решения поставленной задачи:

1. Для начало нужно обрезать текст до 1000 символов, а затем найти первую встречающуюся точку. Для того что бы текст не обрывался на слове/ предложении.
2. Шифрование текста будет проходить побитно значит каждый символ нужно конвертировать в 8 бит они будут размещаться в пикселях изображения поочередно в R,G,B каналы.
3. Декодирование будет считывать побитно данные из пикселей изображения. Мы читаем максимум 1000 символов (+ символы до точки) вместе с ключем. Когда ключ будет найден, в конце текста расшифрованный текст ограничится этим значением.

## **2.2 Шифрование с использованием кодового слова**

Идея такого метода в том, что пользователь задает кодовое слово, которое записывается в имя файла (нового). Перед шифрованием текст объединяется с ключем таким образом шифруемая информация состоит из текста после которого идет ключ. Каждый символ текста (+ ключ) можно преобразовать в набор из 8 бит. Идея заключается в том что выбирается кодовое слово, которое пишется впереди, затем выписываются остальные буквы алфавита в своем порядке. После чего можно вернуть в качестве списка битов для символа.

То есть мы обработаем изображение. записывая биты текста и ключа в каналы R,G,B каждого пикселя. Количество битов задаст пользователь. он будет записываться в один цветовой канал.

Нужно так же учесть маскировку и установку битов. С помощью битовых операций, можно очистить значения старых каналов на нужных позициях (n-битов).

После чего можно применить операции для каждого пикселя и получить модифицированные пиксели, с помощью которых легко можно создать изображение. То есть теперь наше изображение будет хранить скрытую информацию.

## **2.2 Декодирование**

Декодирование можно реализовать так:

Будет считывать новое изображение (зашифрованное), его путь хранит в себе кодовое слово в формате: newImage\_КодовоеСлово.bmp. После чего это слово можно обрезать, благодаря символам ( \_ ).

Что бы не считать лишнюю информацию, можно задать ограничение на 1000 символов (+ текст до точки) \* 8 \* длину ключа.

После чего мы преобразуем 8 бит в байт Char который будет хранить в себе ASCII символ.

То есть мы читаем ключ, удаляем его из конкатинированного текста с ключем, затем берем по 8 бит, и преобразуем их в ASCII символ.

## 2.3 Код программы

Код состоит из двух важных файлов, Lib.hs и Main.hs.

В файле Main мы используем все функции которые хотим экспортить из Lib.hs.

После объединяем их логику и выводим приветственные сообщения.

Ниже приведены коды для Lib.hs и Main.hs соответственно.

### Lib.hs:

```
module Lib (encryptTextInImage, decryptTextFromImage, saveBiographyFragment) where

import Codec.Picture
import Data.Bits
import Data.Char (ord, chr)
import Data.List (foldl', nub)
import Data.List.Split (splitOn)
import System.FilePath (takeBaseName, takeFileName)
import qualified Data.Vector.Unboxed as VU

saveBiographyFragment :: FilePath -> String -> IO ()
saveBiographyFragment path content = do
    let fragment = take 1000 content
        extendedFragment = if length fragment < length content
            then takeWhile (/= '.') (drop (length fragment) content) ++ "."
            else ""
    finalFragment = fragment ++ extendedFragment
    if length finalFragment < 1000
        then putStrLn "Ошибка: текст для биографии < 1000"
        else writeFile path finalFragment
originalAlphabet :: [Char]
originalAlphabet = ['a'..'z'] ++ ['A'..'Z'] ++ ['0'..'9']
hasDuplicates :: Eq a => [a] -> Bool
hasDuplicates xs = length xs /= length (nub xs)
generateSubstitutionAlphabet :: String -> Either String [Char]
generateSubstitutionAlphabet codeWord =
    if hasDuplicates codeWord
        then Left "Ошибка!! Символы должны отличаться(((("
        else Right (codeWord ++ [c | c <- originalAlphabet, c `notElem` codeWord])

encodeText :: String -> String -> Either String String
encodeText codeWord text = do
    subAlphabet <- generateSubstitutionAlphabet codeWord
    let alphabet = originalAlphabet
        Right [encodeChar c alphabet subAlphabet | c <- text]

encodeChar :: Char -> [Char] -> [Char] -> Char
encodeChar c alphabet subAlphabet =
    case lookup c (zip alphabet subAlphabet) of
        Just c' -> c'
        Nothing -> c

encodeBits :: Char -> [Int]
encodeBits c = [ if testBit (ord c) i then 1 else 0 | i <- [7,6..0] ]
bitsToInt :: [Int] -> Int
bitsToInt bits = foldl' (\acc bit -> (acc `shiftL` 1) .|. bit) 0
setMask :: Int -> Pixel8
setMask n = foldl' setBit 0 [0..(n - 1)]
clearMask :: Int -> Pixel8
clearMask n = complement (setMask n)
modifyBits :: Int -> Pixel8 -> Int -> Pixel8
modifyBits n originalByte bits =
    (originalByte .&. clearMask n) .|. (fromIntegral bits .&. setMask n)
encodeLength :: Int -> [Int]
encodeLength len = [ if testBit len i then 1 else 0 | i <- [31,30..0] ]
```

```

encryptBitsToImage :: Int -> Image PixelRGB8 -> [Int] -> Image PixelRGB8
encryptBitsToImage n img bitsData = generateImage encoder width height
  where
    width = imageWidth img
    height = imageHeight img
    totalBits = width * height * 3 * n
    bitsPaddedList = take totalBits (bitsData ++ repeat 0)
    bitsPadded = VU.fromList bitsPaddedList
    encoder x y =
      let index = x + y * width
          startPos = index * 3 * n
          pixelBits = VU.slice startPos (3 * n) bitsPadded
          bitsList = VU.toList pixelBits
          bitsR = bitsToInt $ take n bitsList
          bitsG = bitsToInt $ take n $ drop n bitsList
          bitsB = bitsToInt $ take n $ drop (2 * n) bitsList
          PixelRGB8 r g b = pixelAt img x y
          newR = modifyBits n r bitsR
          newG = modifyBits n g bitsG
          newB = modifyBits n b bitsB
      in PixelRGB8 newR newG newB

encryptTextInImage :: FilePath -> String -> String -> Int -> IO ()
encryptTextInImage imgPath text key bits = do
  putStrLn $ "Попытка чтения изображения: " ++ imgPath
  imgResult <- readImage imgPath
  case imgResult of
    Left err -> putStrLn $ "Ошибка изображения: " ++ err
    Right dynImg -> do
      putStrLn "Изображение прочитано"
      case encodeText key text of
        Left errMsg -> putStrLn errMsg
        Right encodedText -> do
          let outputTextPath = "encrypted_image_text_" ++ key ++ ".txt"
          writeFile outputTextPath encodedText
          putStrLn $ "Зашифрованный текст сохранен в " ++ outputTextPath
          let img = convertRGB8 dynImg
              width = imageWidth img
              height = imageHeight img
              totalPixels = width * height
              totalBitsAvailable = totalPixels * 3 * bits
              maxChars = (totalBitsAvailable - 32) `div` 8
          if length encodedText > maxChars
            then putStrLn $ "Ошибка: текст слишком длинный (максимум " ++ show maxChars ++ "
СИМВОЛОВ)"
            else do
              let messageLength = length encodedText
              let lengthBits = encodeLength messageLength
              let messageBits = concatMap encodeBits encodedText
              let allBits = lengthBits ++ messageBits
              let encodedImg = encryptBitsToImage bits img allBits
              let outputImgPath = takeBaseName imgPath ++ "_encrypted_" ++ key ++ "_" ++ show
bits ++ "bits.bmp"
              putStrLn $ "Сохранение зашифрованного bmp: " ++ outputImgPath
              saveBmpImage outputImgPath (ImageRGB8 encodedImg)
              putStrLn "Зашифрованный файл сохранен"

extractBitsFromPixel :: Int -> PixelRGB8 -> [Int]
extractBitsFromPixel n (PixelRGB8 r g b) =
  let bitsR = [ if testBit r i then 1 else 0 | i <- [(n - 1),(n - 2)..0] ]
      bitsG = [ if testBit g i then 1 else 0 | i <- [(n - 1),(n - 2)..0] ]
      bitsB = [ if testBit b i then 1 else 0 | i <- [(n - 1),(n - 2)..0] ]
  in bitsR ++ bitsG ++ bitsB

decodeText :: String -> String -> Either String String
decodeText codeWord text = do
  subAlphabet <- generateSubstitutionAlphabet codeWord
  let alphabet = originalAlphabet
  Right [decodeChar c alphabet subAlphabet | c <- text]
decodeChar :: Char -> [Char] -> [Char] -> Char
decodeChar c alphabet subAlphabet =
  case lookup c (zip subAlphabet alphabet) of

```

```

Just c' -> c'
Nothing -> c

decodeCharFromBits :: [Int] -> Char
decodeCharFromBits bits = chr $ bitsToInt bits
bitsToChars :: [Int] -> String
bitsToChars bits = [ decodeCharFromBits (take 8 (drop (i * 8) bits)) | i <- [0 .. (length bits `div` 8) - 1]
]

decodeTextFromImage :: Image PixelRGB8 -> Int -> String
decodeTextFromImage img bits = decodedMessage
where
    width = imageWidth img
    height = imageHeight img
    pixels = [ pixelAt img x y | y <- [0..height -1], x <- [0..width -1] ]
    extractedBits = concatMap (extractBitsFromPixel bits) pixels
    (lengthBits, restBits) = splitAt 32 extractedBits
    messageLength = bitsToInt lengthBits
    messageBits = take (messageLength * 8) restBits
    decodedMessage = bitsToChars messageBits

decryptTextFromImage :: FilePath -> IO ()
decryptTextFromImage imgPath = do
    let fileName = takeFileName imgPath
    parts = splitOn "_" fileName
    key = if length parts >= 3 then parts !! 2 else error "Ключ не найден в имени файла"
    bitsStr = if length parts >= 4 then parts !! 3 else error "Количество битов не указано"
    bits = read (filter (`elem` ['0'..'9']) bitsStr) :: Int
    putStrLn $ "Ключ: " ++ key
    putStrLn $ "Количество битов: " ++ show bits
    imgResult <- readImage imgPath
    case imgResult of
        Left err -> putStrLn $ "Ошибка изображения: " ++ err
        Right dynImg -> do
            putStrLn "Изображение готово"
            let img = convertRGB8 dynImg
            decodedText = decodeTextFromImage img bits
            case decodeText key decodedText of
                Left errMsg -> putStrLn errMsg
                Right finalText -> do
                    let outputPath = takeBaseName imgPath ++ "_decrypted.txt"
                    writeFile outputPath finalText
                    putStrLn $ "Расшифровка готова, текст сохранен в " ++ outputPath

```

## Main.hs:

```

module Main where
import System.Environment (getArgs)
import Lib (encryptTextInImage, decryptTextFromImage,
            saveBiographyFragment)

main :: IO ()
main = do
    args <- getArgs
    case args of
        ["encrypt", imgPath, textField, key, bitsStr] -> do
            text <- readFile textField
            saveBiographyFragment "biography1000.txt" text
            let bits = read bitsStr :: Int
            encryptTextInImage imgPath text key bits
            putStrLn "Шифрование готово"
        ["decrypt", imgPath] -> do
            decryptTextFromImage imgPath
            putStrLn "Расшифровка завершена!!!"
        _ -> putStrLn "Выберите: \n\
                        \ Для шифрования: encrypt \
                        \ \n\
                        \ Для расшифровки: decrypt"

```

## **Пояснение:**

Далее я подробно опишу, как функции кода реализуют пункты 2.2 и 2.3:

### **Lib.hs:**

#### **Функция saveBiographyFragment**

Эта функция принимает путь path для сохранения файла и текст content. Она обрезает текст до 1000 символов, добавляя дополнительный фрагмент до ближайшей точки, если текст больше 1000 символов. Если итоговый фрагмент меньше 1000 символов, выводится ошибка. Если длина подходит, результат записывается в файл.

#### **Функция encodeBits**

Функция encodeBits принимает символ с и преобразует его в список из 8 бит, представляющих двоичный код символа. Используется для получения битов текста, который будет зашифрован в изображении.

#### **Функция bitsToInt**

Эта функция преобразует список битов в целое число, складывая каждый бит, сдвинутый на нужное количество позиций. Используется при декодировании битов обратно в символы.

#### **Функция setMask и clearMask**

Функция setMask устанавливает маску на n младших битов (делает их равными 1). clearMask создаёт обратную маску, очищая младшие n битов.

#### **Функция modifyBits**

Эта функция принимает байт originalByte и заменяет его младшие n бит на заданное значение bits, используя маску, которая очищает нужные биты и затем вставляет новые.

#### **Функция encryptTextWithBits**

Эта функция отвечает за запись битов текста и ключа в изображение img, распределяя их по n битов на канал RGB каждого пикселя. Она генерирует новое изображение, где каждый пиксель изменён с учётом добавленных битов.

#### **Функция extractBitsFromPixel**

Функция извлекает n младших битов из каждого канала RGB для одного пикселя. Эти биты потом собираются в битовый поток, который используется при декодировании.

#### **Функция decodeChars**

Эта функция принимает поток битов, по 8 бит декодирует в символы, собирает их в строку, и завершает декодирование, когда ключ найден в конце текста. Если ключ найден, текст обрезается на ближайшей точке после 1000 символов.

#### **Функция findLastFullStop**

Находит первую точку после 1000 символов текста и возвращает её индекс. Это используется для обрезки текста после окончания предложения.

#### **Main.hs:**

Состоит из 4 пунктов: Обработка аргументов, шифрование, декодирование, вывод сообщения.

#### **Обработка аргументов:**

Функция получает аргументы командной строки и определяет, какой процесс выполнять:

"encrypt" — для шифрования текста в изображении.

"decrypt" — для дешифрования текста из изображения.

В случае неправильного ввода команда выведет сообщение с инструкциями.

#### **Шифрование:**

При получении команды encrypt функция читает текст из файла textField, сохраняет первые 1000 символов и ближайшую точку в отдельный файл с помощью saveBiographyFragment.

Затем, определяя количество битов для шифрования из строки bitsStr, функция вызывает encryptTextInImage для выполнения шифрования текста в изображении по указанному пути imgPath.

#### **Дешифрование:**

При получении команды decrypt функция вызывает decryptTextFromImage, которая извлекает текст из зашифрованного изображения и сохраняет его в текстовый файл.

#### **Вывод сообщений:**

При успешном выполнении шифрования/ дешифрования выводятся сообщения о завершении.

#### **Ограничения по шифрованию:**

- Текст  $\leq$  1000 символов + символы до первой точки (конец предложения).
- Что бы избежать переполнения - максимальное количество = Ширина \* Высота изображения \* Длина ключа \* 3 (R, G, B).
- Что бы избежать недобора битов - остальная часть заполняется нулями.

### 3. Результаты

Что бы запустить шифрование необходимо прописать:

`stack run encrypt «изображение» «файл_биографии» «кодовое слово» n,`

где n - количество желаемых битов.

Для дешифровки:

`stack decrypt «зашифрованное_изображение»`

Ниже приведены результаты программы ( для n = 1, n = 2 ... n = 8), так же будет приведен текст, начальный, обрезанный, расшифрованный.

Кодовое слово во всех случаях = «ABC»:



Рис.2 n = 1 бит.



Рис.3 n = 2 бит.



Рис.4 n = 3 бит.



Рис.5 n = 4 бит.



Рис.6 n = 5 бит.



Рис.7 n = 6 бит.



Рис.8 n = 7 бит.



Рис.9 n = 8 бит.

Nikolai Ivanovich Lobachevsky was born on December 1, 1792, in Makaryev, Nizhny Novgorod province, Russia. He was an outstanding Russian mathematician and is often regarded as the creator of non Euclidean geometry. His work and ideas laid the foundation for hyperbolic geometry, which profoundly impacted mathematics and science. In 1807, he enrolled at Kazan University, where he studied under Professor Martin Bartels, initially focusing on mathematics and later on physics. In 1826, he published his groundbreaking work, *On the Principles of Geometry*, which outlined the fundamentals of a new geometric system. Lobachevsky proposed that alternative geometries could exist without following Euclid's parallel postulate, which revolutionized the mathematical world. His contributions gained recognition only after his death, and today, Lobachevskys name is associated with one of the most influential theories in mathematics. Besides his theoretical work, Lobachevsky served as rector at Kazan University, where he actively promoted education and science in Russia. His ideas inspired significant advances in physics and mathematics, helping form modern concepts of space. Lobachevsky passed away in 1856, leaving behind a rich scientific legacy that continues to inspire scholars worldwide.

#### Оригинальный текст

Nikolai Ivanovich Lobachevsky was born on December 1, 1792, in Makaryev, Nizhny Novgorod province, Russia. He was an outstanding Russian mathematician and is often regarded as the creator of non Euclidean geometry. His work and ideas laid the foundation for hyperbolic geometry, which profoundly impacted mathematics and science. In 1807, he enrolled at Kazan University, where he studied under Professor Martin Bartels, initially focusing on mathematics and later on physics. In 1826, he published his groundbreaking work, *On the Principles of Geometry*, which outlined the fundamentals of a new geometric system. Lobachevsky proposed that alternative geometries could exist without following Euclid's parallel postulate, which revolutionized the mathematical world. His contributions gained recognition only after his death, and today, Lobachevskys name is associated with one of the most influential theories in mathematics. Besides his theoretical work, Lobachevsky served as rector at Kazan University, where he actively promoted education and science in Russia.

#### Обрезанный текст

NfhliAf IsAklsfCe LIBACebsphv tAp Blok lk DbCbjBbo 1, 1792, fk MAhAovbs, Nfwekv Nlsdlola molsfkCb, RrppfA. Hb tAp Ak lrqpqAkafkd RrppfAk jAqebjAqfCfAk Aka fp lcqbk obdAoaba Ap qeb CobAqlo lc klk ErCifabAk dbljbqv. Hfp tloh Aka fabAp iAfa qeb clarkaAqflk clo evmboBlifC dbljbqv, tefCe molclrkaiv fjmACqba jAqebjAqfCp Aka pCfbkCb. Ik 1807, eb bkoliiba Aq KAwAk Ukfsbopfqv, tebob eb pqrafba rkabo Polcbpplo MAoqfk yAoqbip, fkfjfAiiv clCrpfkd lk jAqebjAqfCp Aka iAqbo lk mevpfCp. Ik 1826, eb mrBifpeba efp dolrkaBobAhfkdk tloh, Ok qeb PofkCfmibp lc Gbljbqv, tefCe lrqifkba qeb crkaAjbkqAip lc A kbt dbljbqofC pvpqbj. LIBACebsphv molmlpba qeAq AjqbokAqfsb dbljbqofbp Clria bufpq tfqelrq clliitfkdk ErCifa'p mAoAiibi mlpqriAqb, tefCe obslirqflkfwbq qeb jAqebjAqfCAi tloia. Hfp ClkqofBrqflkp dAfkbk obCldkfqflk lkiv Acqbo efp abAqe, Aka qlaAv, LIBACebsphv kAjb fp ApplCfAqba tfqe lkb lc qeb jlpo fkcirbkqfAi qeblofbp fk jAqebjAqfCp. ybpfabp efp qeblobqfCAi tloh, LIBACebsphv pbosba Ap obCqlo Aq KAwAk Ukfsbopfqv, tebob eb ACqfsbiv moljlqba barCAqflk Aka pCfbkCb fk RrppfA.

Зашифрованный текст (n = 5 бит).

Nikolai Ivanovich Lobachevsky was born on December 1, 1792, in Makaryev, Nizhny Novgorod province, Russia. He was an outstanding Russian mathematician and is often regarded as the creator of non Euclidean geometry. His work and ideas laid the foundation for hyperbolic geometry, which profoundly impacted mathematics and science. In 1807, he enrolled at Kazan University, where he studied under Professor Martin Bartels, initially focusing on mathematics and later on physics. In 1826, he published his groundbreaking work, On the Principles of Geometry, which outlined the fundamentals of a new geometric system. Lobachevsky proposed that alternative geometries could exist without following Euclid's parallel postulate, which revolutionized the mathematical world. His contributions gained recognition only after his death, and today, Lobachevskys name is associated with one of the most influential theories in mathematics. Besides his theoretical work, Lobachevsky served as rector at Kazan University, where he actively promoted education and science in Russia.

Расшифрованный текст (n = 1...8 бит).

При всех количествах бит (от 1 ... 8 последних бит каждого байта), результат расшифрованного текста одинаковый. Что подтверждает верную расшифровку изображения.

## **4. Заключение**

В результате выполнения практического задания №3, Был создан проект в stack, в котором реализована логика шифрования текста методом кодового слова, и расшифровка по кодовому слову из названия файла. Программа:

- Берет из текстового файла биографии 1000 символов, находит ближайшую точку для конца предложения.
- Шифрует полученный текстовый файл в изображение портрет Лобачевского Николая Ивановича, формата .bmp (24-разрядный).
- Сохраняет новое измененное изображение.
- Декодирует зашифрованное изображение.
- Сохраняет файл с расшифрованным текстом.

В результате:

- Можно закодировать любую информацию.
- Хранить/ передавать сообщение в любых других изображениях .bmp форматы.
- Декодировать изображения по кодовому слову.

Все программы (за исключением файлов структуры проекта таких как: .cabal, .yaml, .md, .sqlite3, .yaml.lock), имеют формат (.hs), программы написаны на языке Haskell, в интегрируемой среде разработки Visual Studio Code 1.94.

## **5. Список литературы**

- [1] W. Kurt. Get Programming with Haskell. Москва: ДМК, 2019. 481 с.
- [2] В. Брагилевский. Haskell in Depth. Москва: ДМК, 2019. 340 с.

## **6. Приложение А**

- Ссылка на репозиторий с проектом: <https://github.com/MathematicLove/Haskellove.git>