1. Министерство образования и науки Российской Федерации
2. Санкт-Петербургский Политехнический Университет Петра Великого
3. —
4. Институт компьютерных наук и кибербезопасности
5. **Кафедра «Информационная безопасность компьютерных систем»**

**Лабораторная работа №10**

**Отчёт**

1. «Методы надежного хранения и передачи информации»
2. по дисциплине «Основы информационной безопасности»
3. Выполнил

студенты гр. 5151001/40001 Кириллов.Д.А

1. <*подпись*>

Преподаватель Орел Е.М.

1. <*подпись*>
2. Санкт-Петербург
3. 2025
4. **Цель работы:**

Ознакомление с методом Хемминга помехоустойчивого кодирования, позволяющим обнаруживать и автоматически исправлять ошибки, возникающие при хранении и передаче информации в ненадежных средах.

1. **Постановка задачи:**

Необходимо научиться кодировать и декодировать произвольный текст методом Хемминга, разработать программу, осуществляющую это, и проверить ее работоспособность на разных кодируемых блоках с измененными битами, сравнить размеры исходного файла и закодированного в зависимости от длины блоков.

1. **Ход работы:**
2. Пример расчета кода Хемминга для 16-разрядного числа:

Возьмем для примера 1891541473225918, переведем его в двоичное и получим

110101110000101100011010001100101101110110010111110. Далее разобьем на блоки по 8 информационных бит, получим:

[1, 1, 0, 1, 0, 1, 1, 1]

[0, 0, 0, 0, 1, 0, 1, 1]

[0, 0, 0, 1, 1, 0, 1, 0]

[0, 0, 1, 1, 0, 0, 1, 0]

[1, 1, 0, 1, 1, 1, 0, 1]

[1, 0, 0, 1, 0, 1, 1, 1]

[1, 1, 0, 0, 0, 0, 0, 0]

Рассчитаем количество контрольных битов, чтоб оно удовлетворяло неравенству

2^L = L + m + 1, где m – количество информационных битов. Получим L = 4.

Далее рассчитаем код Хемминга для каждого блока: информационные биты будут стоять на позициях, равных степеням двойки, т.е 1,2,4,8. Каждый контрольный бит будет проверять все биты, у которых в двоичном представлении позиции есть 1 в разряде X, где X равен двоичному логарифму от позиции контрольного бита.

Значение контрольного бита будет равно XOR от всех его подконтрольных битов.

Таким образом, в первом блоке контрольный бит 1 будет проверять позиции 3, 5, 7,9,11 и будет равен 0, контрольный бит 2 – 3,6,7,10,12 и равен 0, контрольный бит 4 – 5,6,7,12 и равен 1, контрольный бит 8 – позиции с 8 по 12 и равен 1. Проделав эти преобразования со всеми блоками, получим:

[0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1]

[0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1]

[1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0]

[0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0]

[0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1]

[1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1]

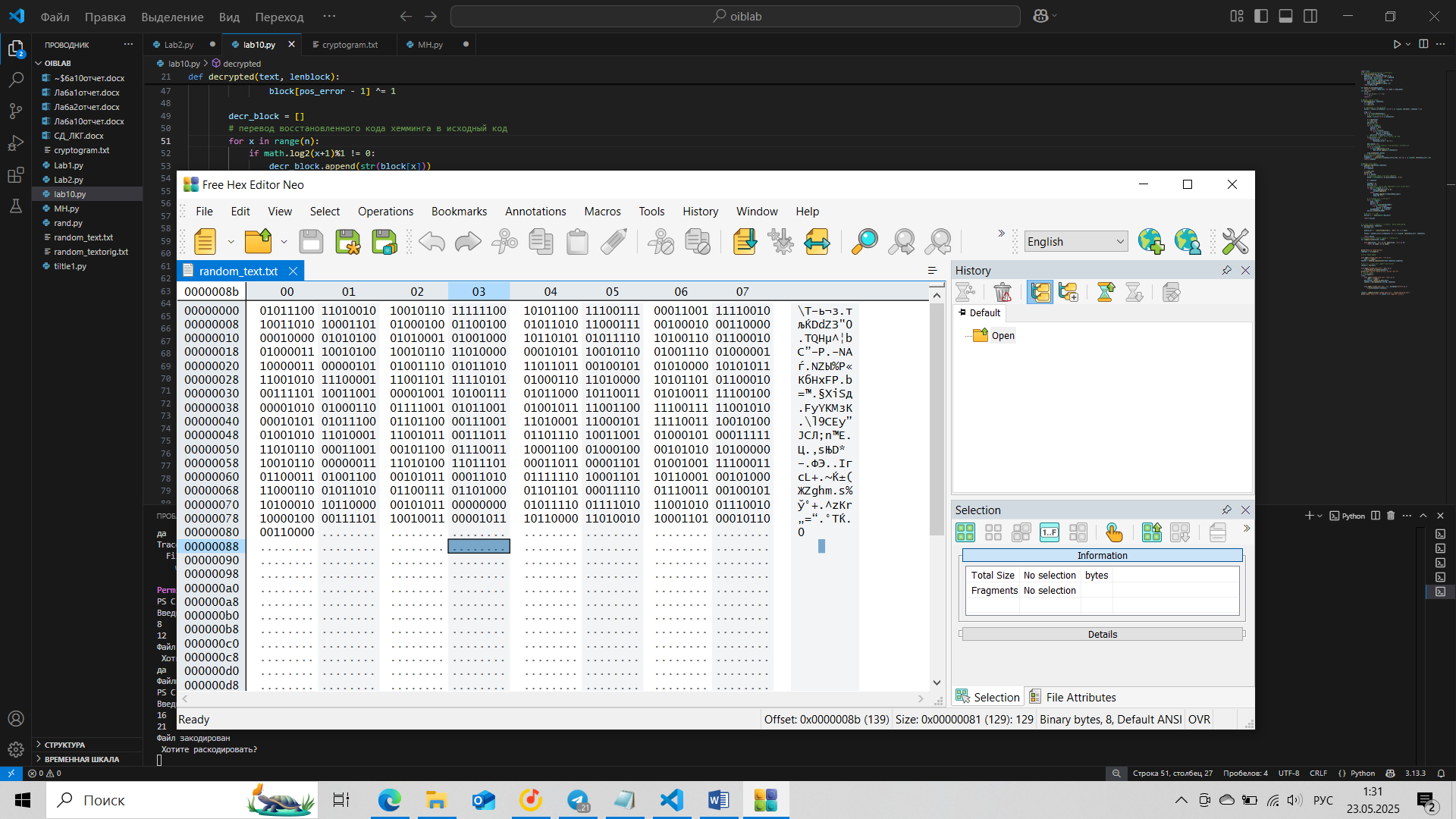
[0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0]

1. Блок-схема алгоритма вычисления помехоустойчивого кода:

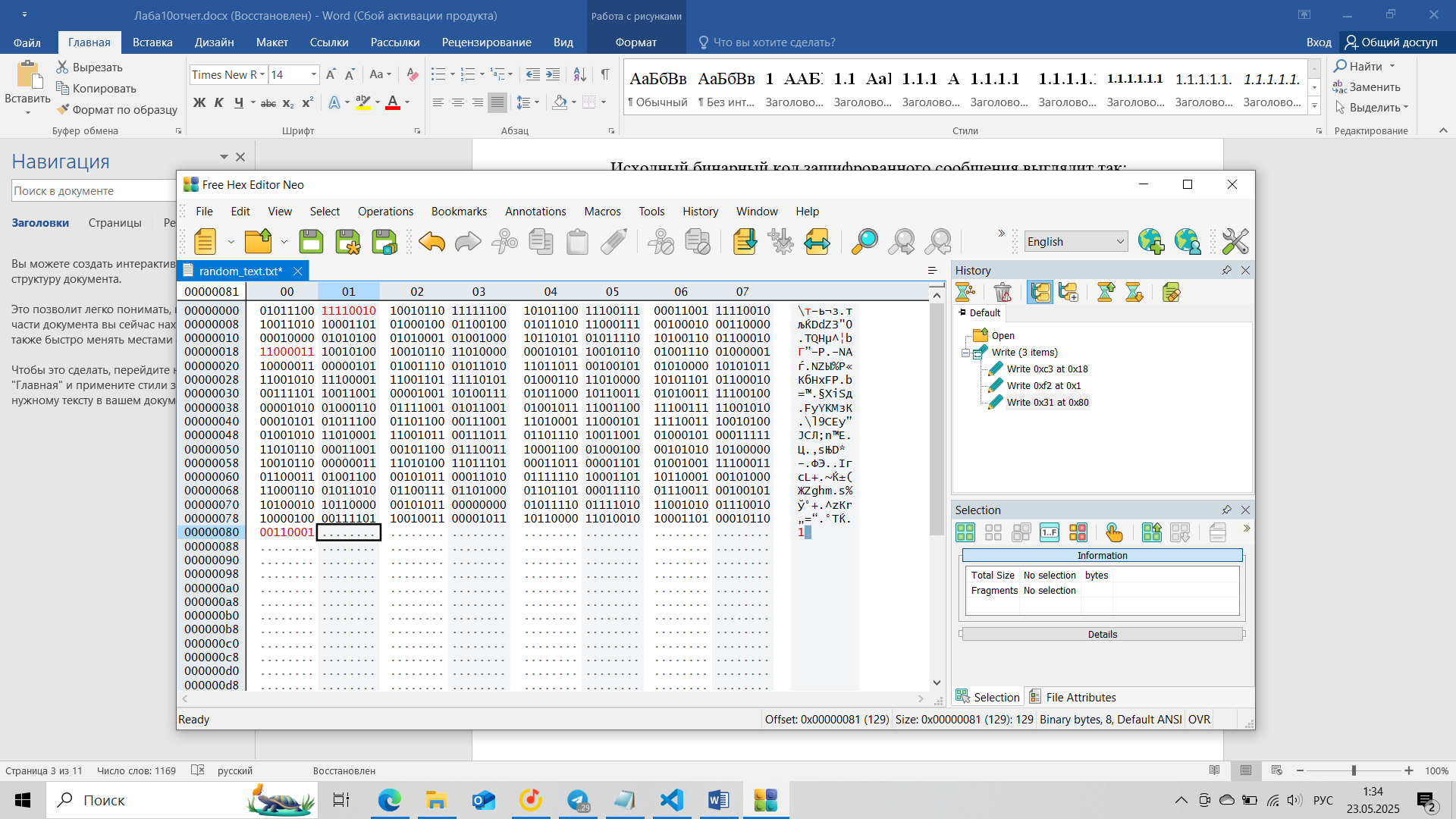
Приложение 1, 2.

1. Описание однобитных и групповых ошибок, внесенных в файл:

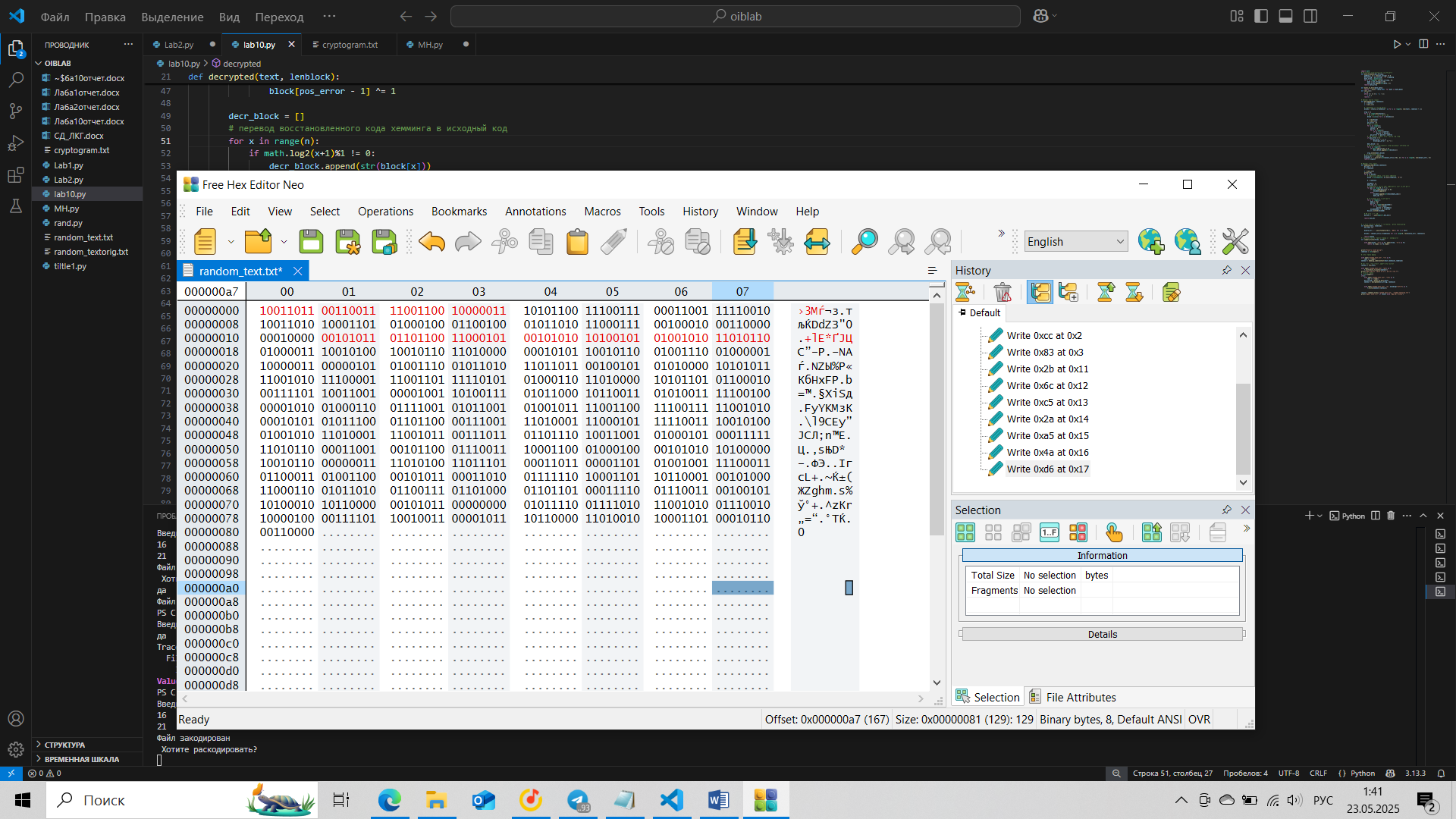
Исходный бинарный код зашифрованного сообщения выглядит так:



*Рисунок 1. Исходный двоичный код.*

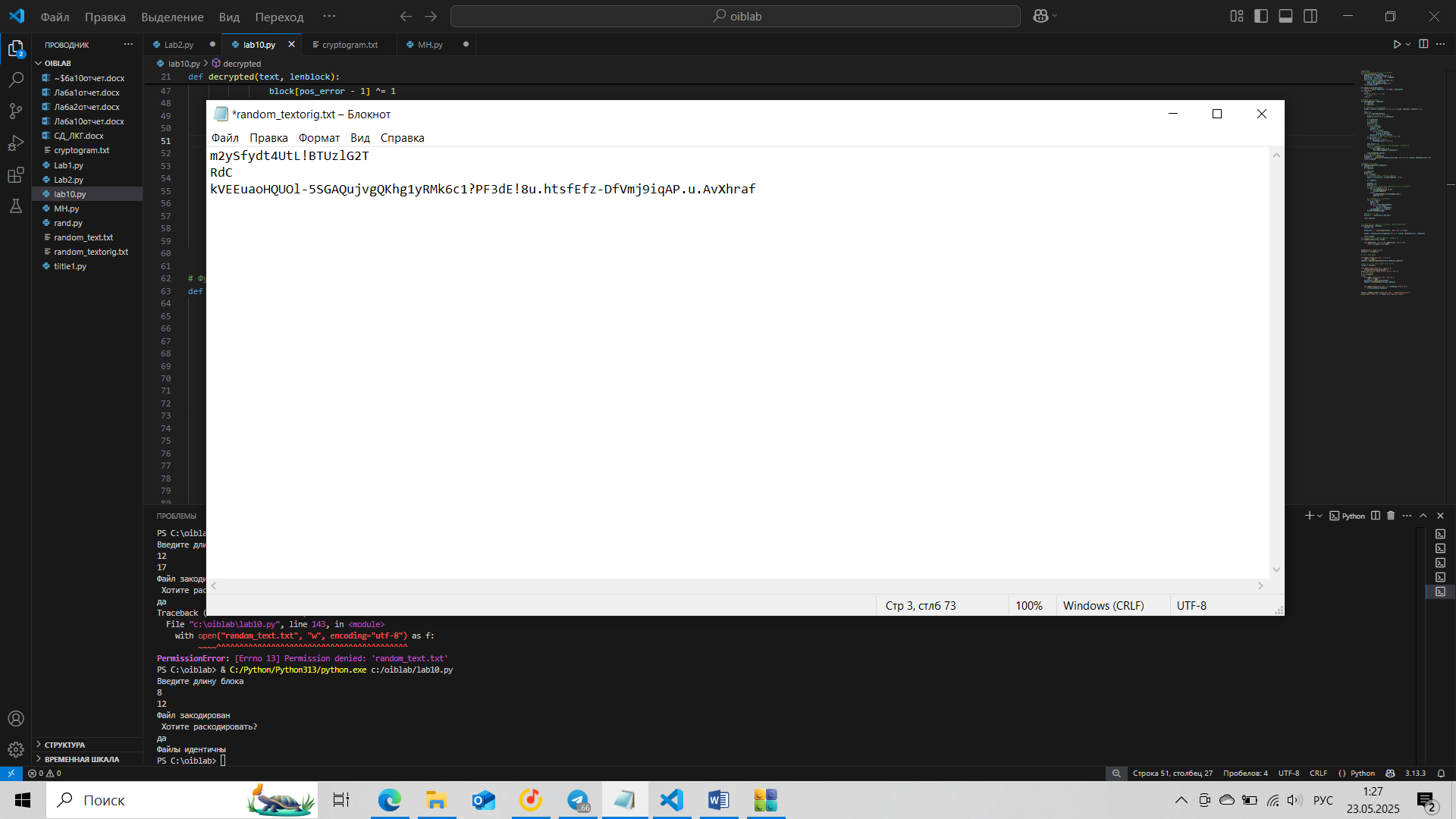


*Рисунок 2. Одиночные ошибки.*



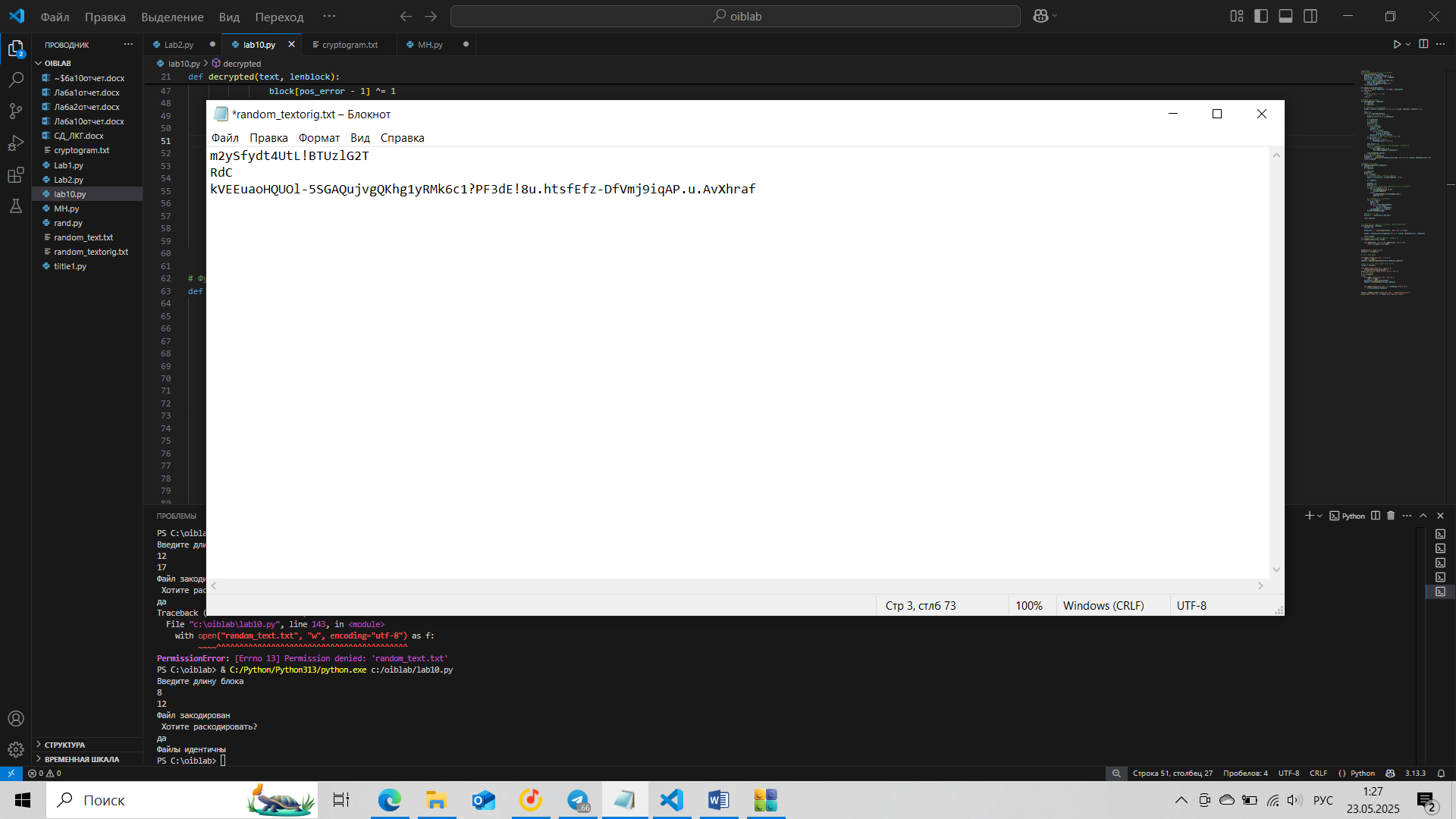
*Рисунок 3. Групповые ошибки.*

1. Результаты восстановления:



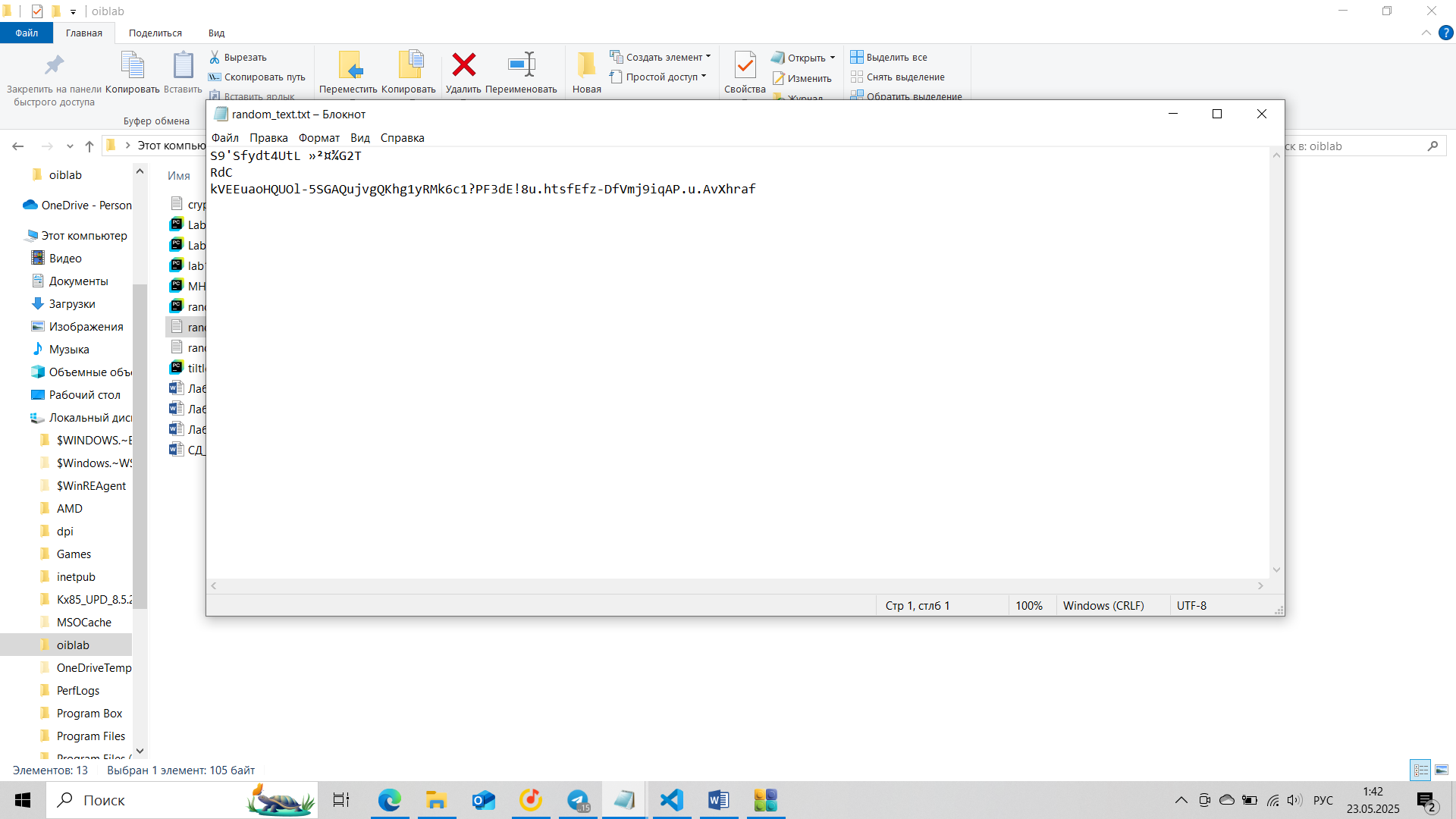
*Рисунок 4. Исходный текст.*

Как видно, при одиночных ошибках код восстановился корректно



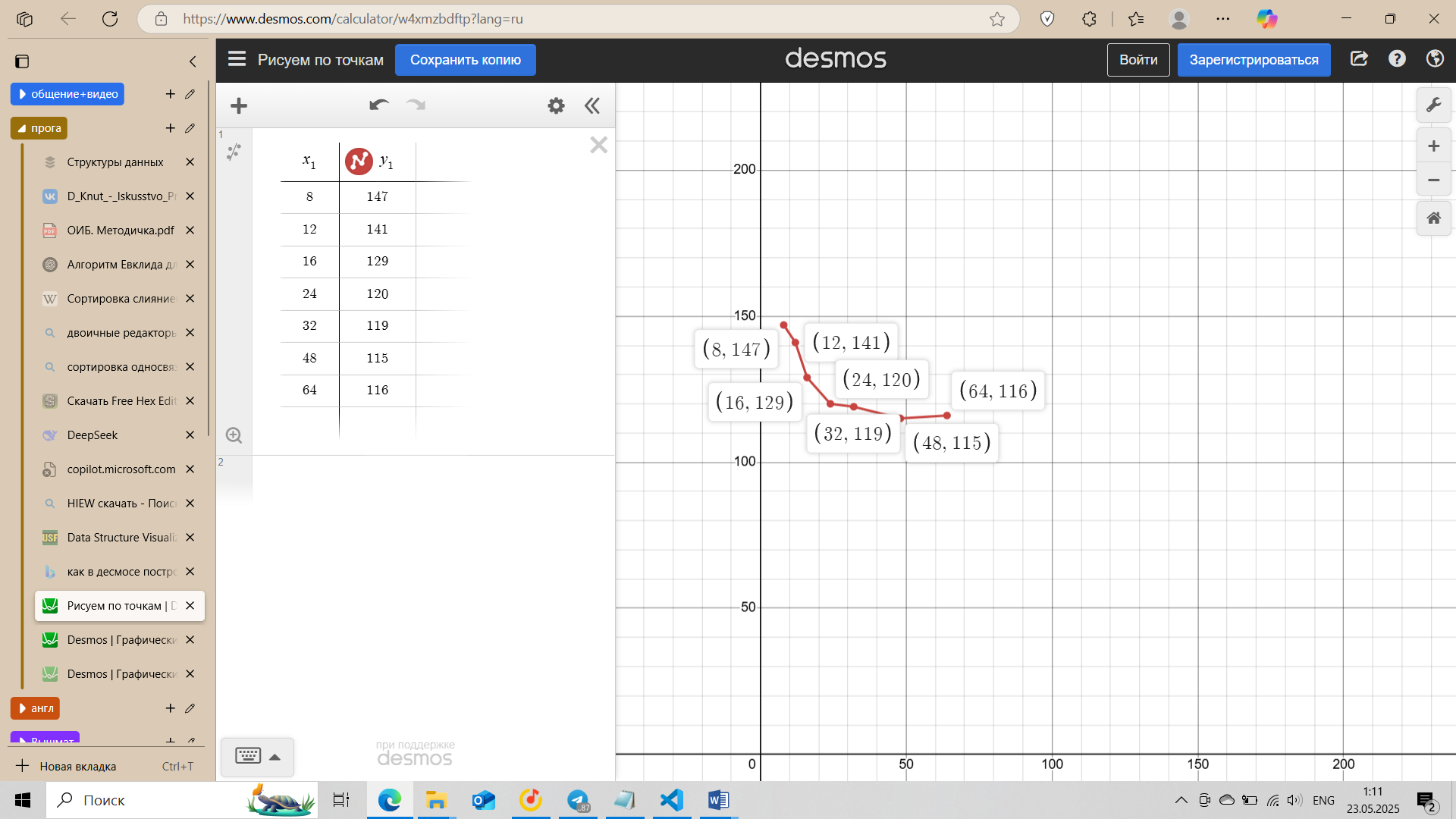
*Рисунок 5. Результат восстановления одиночных ошибок.*

При множественных групповых ошибках видны неточности в расшифровке символов



*Рисунок 6. Результат восстановления групповых ошибок.*

1. График зависимости изменения объема закодированный информации от размера кодируемых блоков:



*Рисунок 7. График зависимости объема информации от кодируемых блоков.*

1. Ответы на контрольные вопросы:
2. Расстояние Хемминга – минимальное число разрядов с неодинаковым значением кодовых слов.
3. В функцию мы передаем n = количество символов и k = количество информационных символов. Далее для них мы представляем данные как многочлен d1 + d2x + d3x^2 и тд. После в поле Галуа мы вычисляем значения этих многочленов в n разных точках и записываем полученные значения в блок. При декодировании можно исправить до (n-k)/2 ошибок блоке.
4. Можно сделать так, чтоб 8 бит контролировал все биты с 1 по 7.
5. Размер кодируемого блока влияет на количество контрольных битов, общую длину кодового слова и на избыточность.
6. Схема размещения контрольных и информационных битов в блоке размером 16 разрядов:

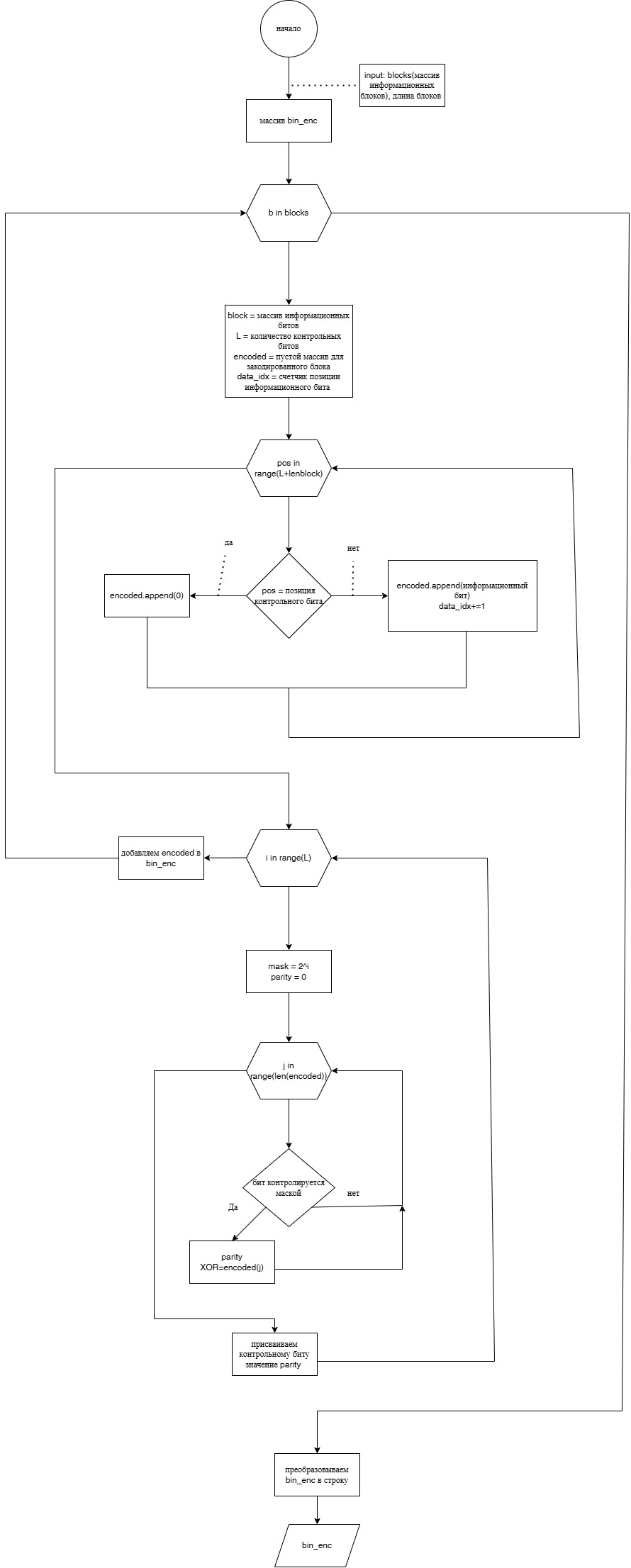
|  |  |  |  |
| --- | --- | --- | --- |
| Позиция бита | Тип бита | Контролируется битами | Что контролирует |
| 1 | P1 |  | 3,5,7,9,11,13,15,17,19,21 |
| 2 | P2 |  | 3,6,7,10,11,14,15,18,19 |
| 3 | D1 | P1, P2 | информационный |
| 4 | P4 |  | 4-7,12-15,20-21 |
| 5 | D2 | P1, P4 | информационный |
| 6 | D3 | P2, P4 | информационный |
| 7 | D4 | P1, P2, P4 | информационный |
| 8 | P8 |  | 8-15 |
| 9 | D5 | P1, P8 | информационный |
| 10 | D6 | P2, P8 | информационный |
| 11 | D7 | P1, P2, P8 | информационный |
| 12 | D8 | P4, P8 | информационный |
| 13 | D9 | P1, P4, P8 | информационный |
| 14 | D10 | P2, P4, P8 | информационный |
| 15 | D11 | P1,P2,P4,P8 | информационный |
| 16 | P16 |  | 17-21 |
| 17 | D12 | P1, P16 | информационный |
| 18 | D13 | P2, P16 | информационный |
| 19 | D14 | P1,P2,P16 | информационный |
| 20 | D15 | P4,P16 | информационный |
| 21 | D16 | P1,P4,P16 | информационный |

1. **Выводы:**

Был изучен метод кодирования и декодирования методом Хемминга, разработана программа осуществляющая эти действия и проверяющая, изменился ли файл после создания помех или нет. Также были зафиксированы изменения размера файла в зависимости от длины кодируемых блоков и составлен график этого измерения.

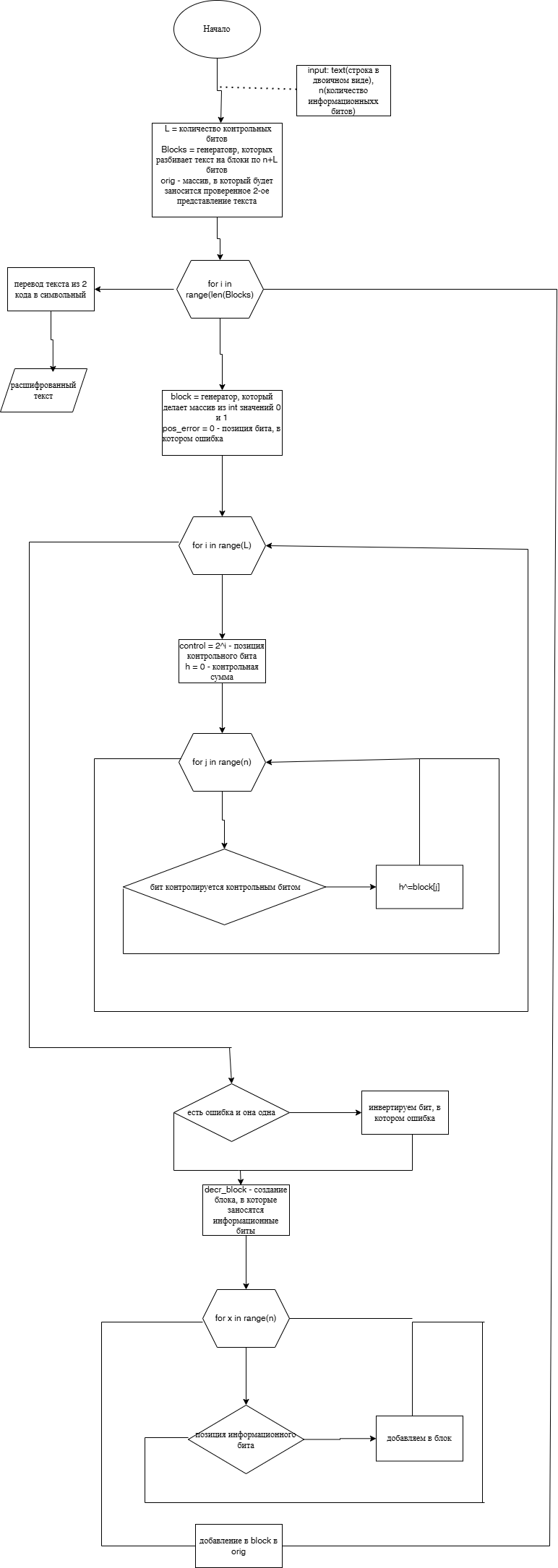
**Приложение 1**

Блок схема кодирования методом хемминга

****

Приложение 2.

Блок схема декодирования кода хемминга



**Приложение 3**

import math

# Вычисление количества контрольных битов

def calc\_L(k):

    L = 0

    while k > (2 \*\* L - L - 1):

        L += 1

    return L

# функция декодирования

def decrypted(text, lenblock):

    n = lenblock

    L = calc\_L(n)

    # разделение строки на блоки

    blocks = [text[i:i+lenblock + L] for i in range(0, len(text), lenblock + L)]

    orig = []

    for i in range(len(blocks)):

        # перевод строки в массив 01

        block = [int(c) for c in blocks[i]]

        n = len(block)

        L = calc\_L(n)

        pos\_error = 0

        # поиск ошибки

        for i in range(L):

            control = 2\*\*i

            parity = 0

            for j in range(n):

                if (j + 1) & control:

                    parity ^= block[j]

            pos\_error += parity \* control

        # исправление ошибки, если в блоке она одна

        if pos\_error > 0:

            if pos\_error - 1 < n:

                block[pos\_error - 1] ^= 1

        decr\_block = []

        # перевод восстановленного кода хемминга в исходный код

        for x in range(n):

            if math.log2(x+1)%1 != 0:

                decr\_block.append(str(block[x]))

        orig.extend(decr\_block)

    # восстановление файла

    binary\_str = ''.join(orig)

    origtext = ''.join(chr(int(binary\_str[i:i+8], 2)) for i in range(0, len(binary\_str), 8))

    return origtext

# Функция кодирования

def hamming\_code(blocks,lenblock):

    bin\_enc = []

    for b in blocks:

        # добавление нулей, если блок неполный

        block = list(map(int, b.ljust(lenblock, '0')))

        k = lenblock

        L = calc\_L(k)

        encoded = []

        data\_idx = 0

        # добавление в массив информационных и контрольных битов

        for pos in range(0, k + L):

            if (math.log2(pos+1)%1 == 0):

                encoded.append(0)

            else:

                encoded.append(int(block[data\_idx]))

                data\_idx += 1

        # вычисление контрольных битов

        for i in range(L):

            mask = 2\*\*i

            parity = 0

            for j in range(len(encoded)):

                if (j + 1) & mask:

                    parity ^= encoded[j]

            encoded[mask - 1] = parity

        bin\_enc.extend(encoded)

    # перевод в строку

    bin\_enc = ''.join(map(str,bin\_enc))

    return bin\_enc

# Перевод текста в массив строк длиной, равной длине блока

def bintext(text, lenblock):

    enc\_text =[]

    binary\_str = ''.join(format(ord(c), '08b') for c in text)

    blocks = [binary\_str[i:i+lenblock] for i in range(0, len(binary\_str), lenblock)]

    return blocks

# Сравнение декодированного файла и оригинального

def compare\_files(file1, file2):

    with open(file1, 'rb') as f1, open(file2, 'rb') as f2:

        return f1.read() == f2.read()

print("Введите длину блока")

lenblock = int(input())

# кодирование файла

with open("random\_text.txt", "r") as f:

    text = f.read()

newtext = hamming\_code(bintext(text,lenblock),lenblock)

# для того, чтоб отсеять избыточные символы

lentext = len(text)

with open("random\_text.txt", "w", encoding="utf-8") as f:

    f.write(newtext)

print("Файл закодирован\n Хотите раскодировать?")

# раскодирование

if(input()):

    with open("random\_text.txt", "r", encoding="utf-8") as f:

        text = f.read()

    newtext = decrypted(text, lenblock)

    with open("random\_text.txt", "w", encoding="utf-8") as f:

        f.write(newtext[:lentext])

result = compare\_files('random\_text.txt', 'random\_textorig.txt')

print("Файлы идентичны" if result else "Файл поврежден")