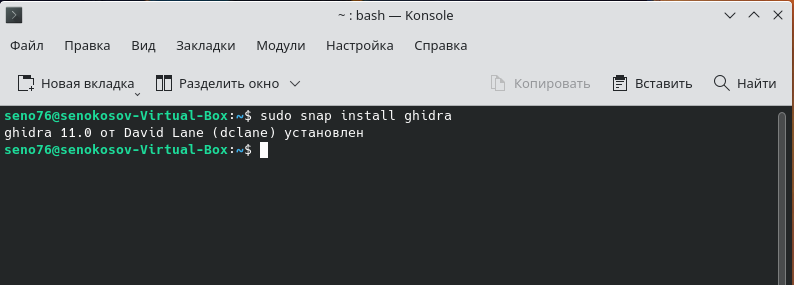
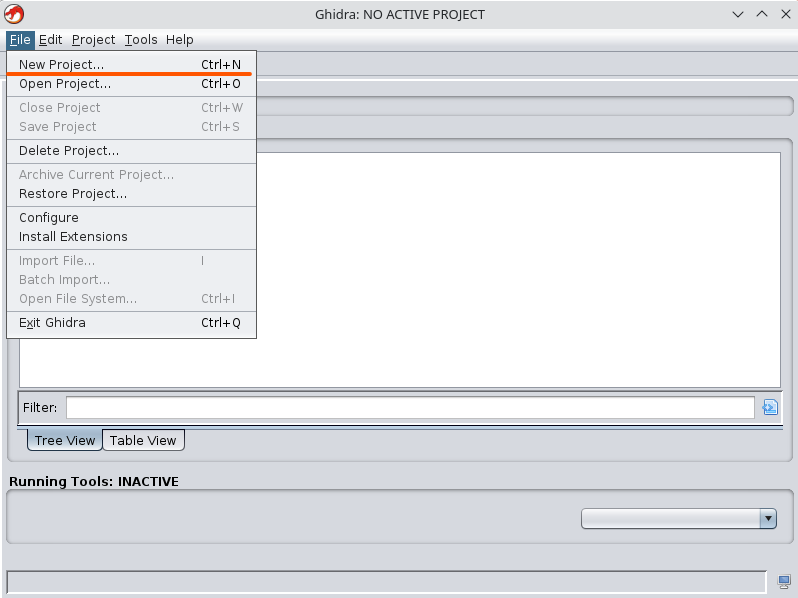
**Лабораторная работа 6**

**Выполнил: Сенокосов Владислав Владимирович 431**

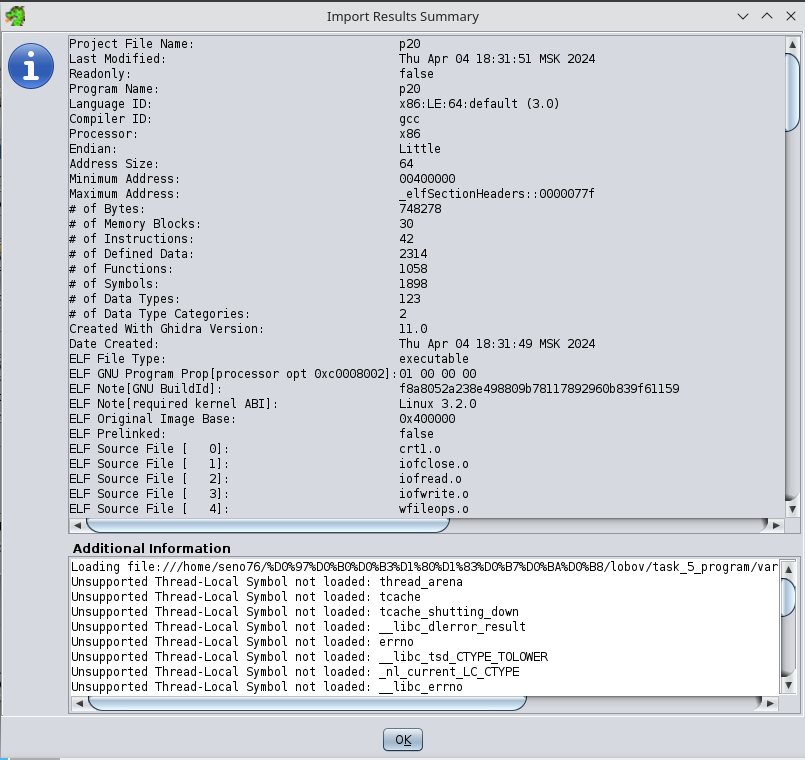
Установили пакет ghidra:



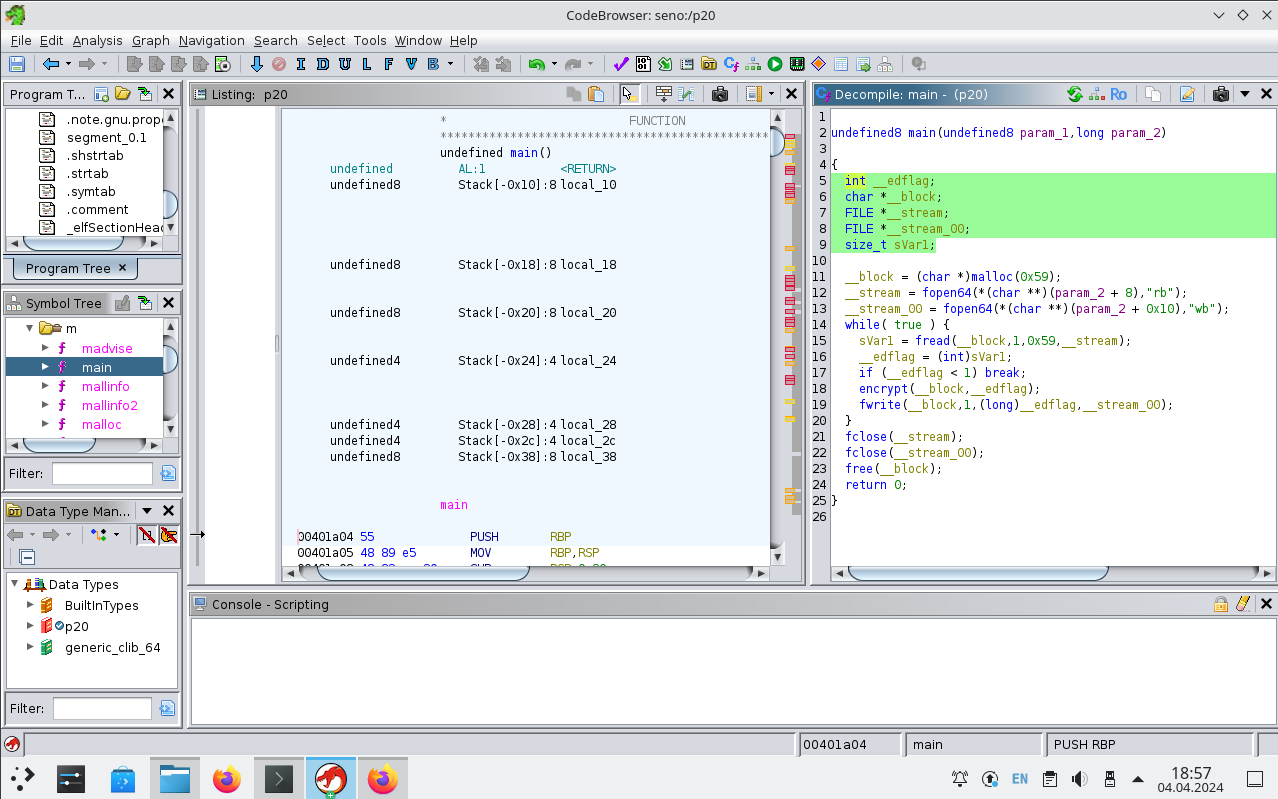
Запускаем и создаем новый проект:



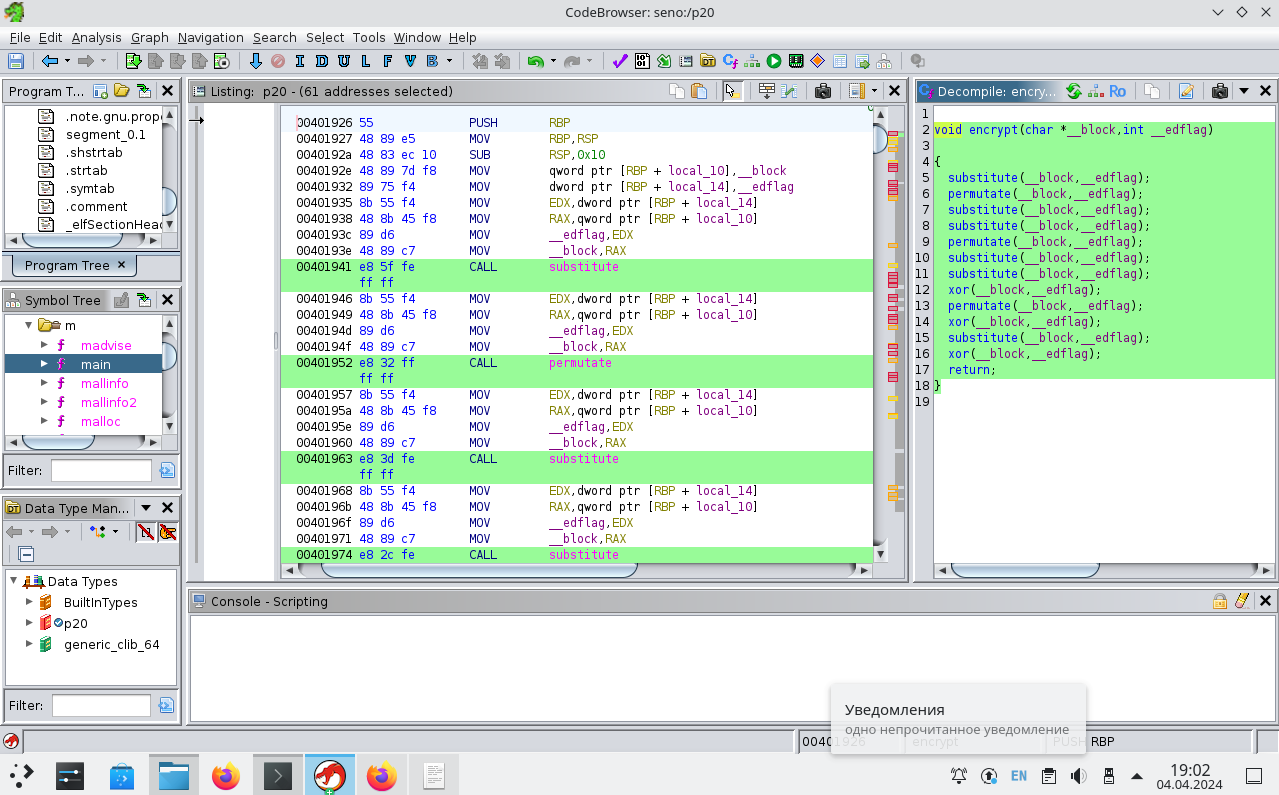
Выбираем в качестве исследования файл, соответствующий порядковому номеру в группе, то есть 20 и запустим в программе:



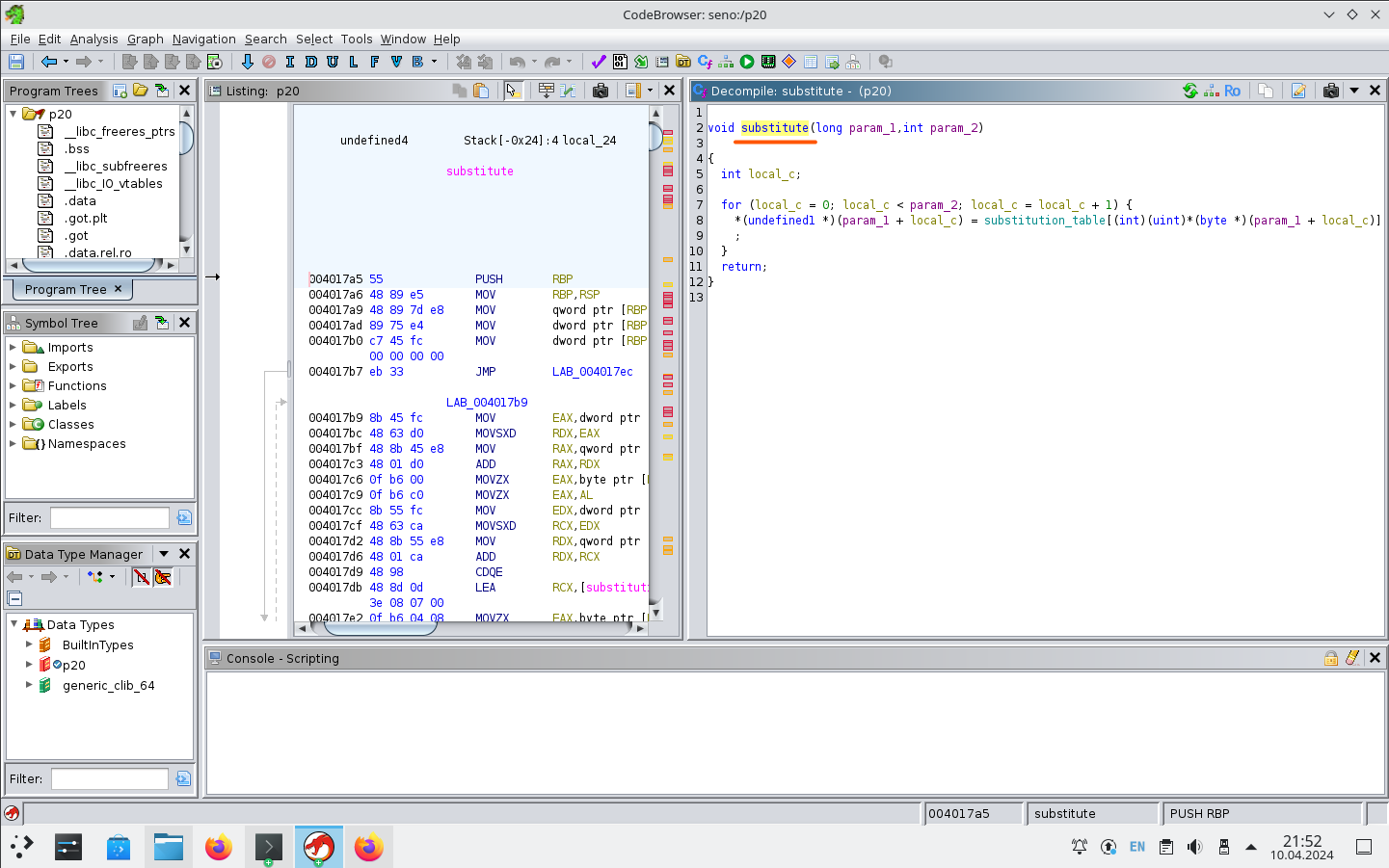
Основная функция main в которой используется шифрование:

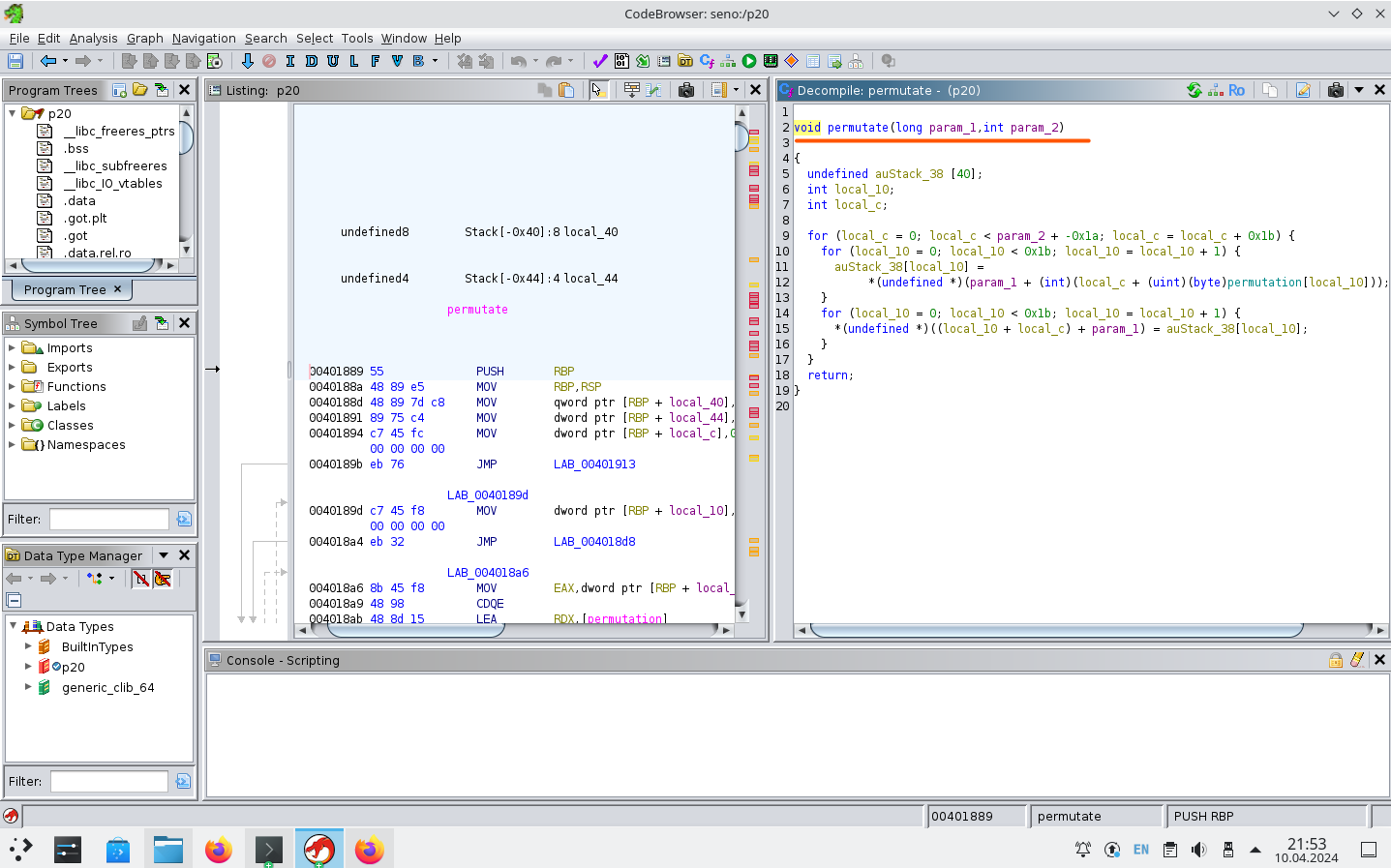


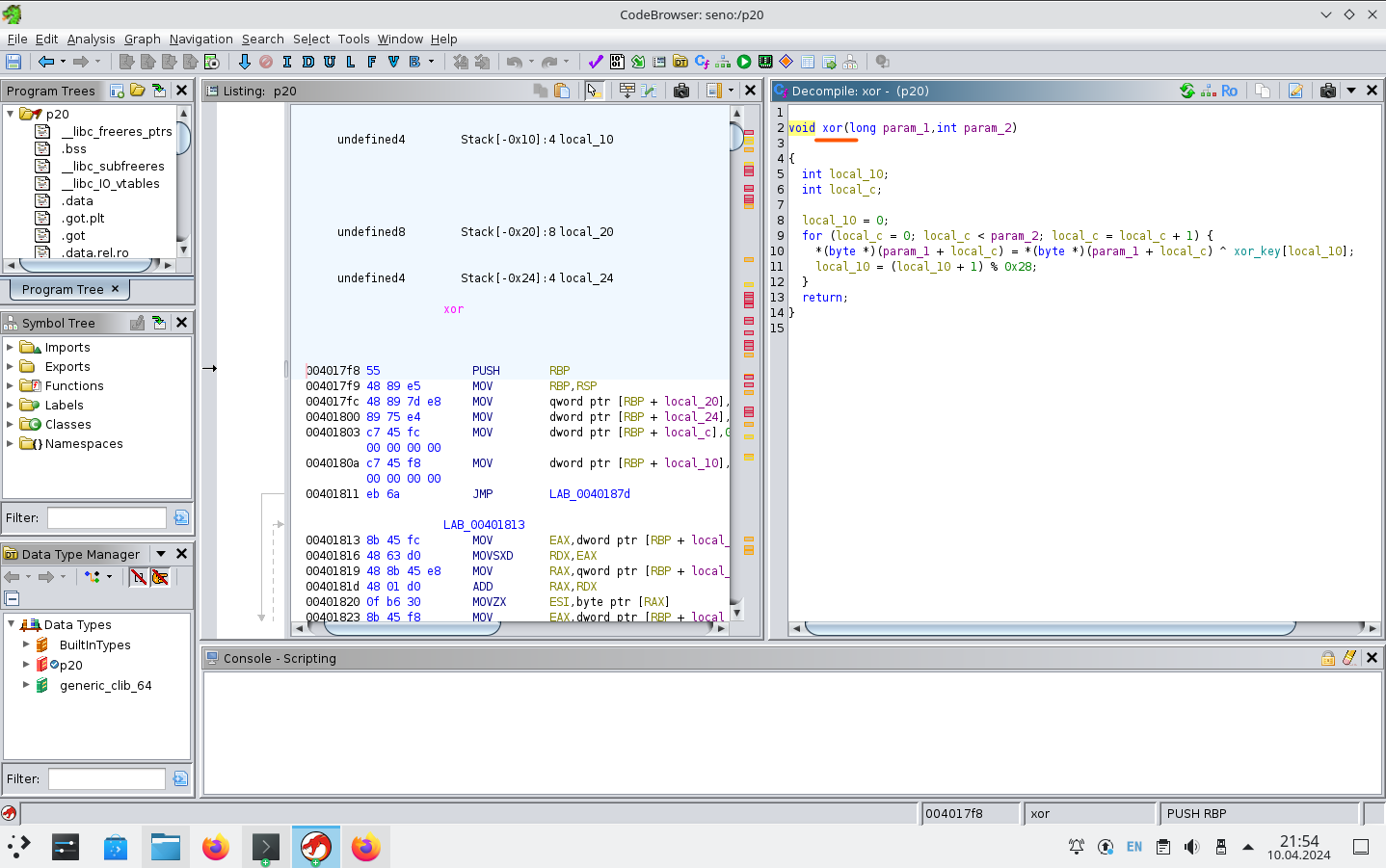
Функция шифрования (состоящая из других вспомогательных функций):



Определим функции, которые необходимо разобрать для написания расшифровщика:







Перепишем функцию кодирования на язык программирования Python:

import data

def main(in\_file, out\_file):

    block = 89

    stream\_2 = open(out\_file, "wb")

    with open(in\_file, "rb") as stream\_1:

        while True:

            chunk = stream\_1.read(block)

            if not chunk:

                break

            encrypted\_chunk = encrypt(chunk, len(chunk))

            stream\_2.write(bytes(encrypted\_chunk))

    stream\_2.close()

def encrypt(\_\_block, \_\_edflag):

    \_\_block = list(\_\_block)

    substitute(\_\_block,\_\_edflag)

    permutate(\_\_block,\_\_edflag)

    substitute(\_\_block,\_\_edflag)

    substitute(\_\_block,\_\_edflag)

    permutate(\_\_block,\_\_edflag)

    substitute(\_\_block,\_\_edflag)

    substitute(\_\_block,\_\_edflag)

    xor(\_\_block,\_\_edflag)

    permutate(\_\_block,\_\_edflag)

    xor(\_\_block,\_\_edflag)

    substitute(\_\_block,\_\_edflag)

    xor(\_\_block,\_\_edflag)

    return \_\_block

def substitute(arr, param\_2):

    for local\_c in range(param\_2):

        arr[local\_c] = data.substitution\_table[arr[local\_c]]

def permutate(arr, param\_2):

    new\_stack = [0] \* 40

    for local\_c in range(0, param\_2 - 26, 27):

        for local\_10 in range(27):

            new\_stack[local\_10] = arr[local\_c + int(data.permutation[local\_10])]

        for local\_10 in range(27):

            arr[local\_10 + local\_c] = new\_stack[local\_10]

def xor(arr, param\_2):

    local\_10 = 0

    for local\_c in range(param\_2):

        arr[local\_c] = arr[local\_c] ^ data.xor\_key[local\_10]

        local\_10 = (local\_10 + 1) % 40

print(main("text.txt", "encode.txt"))

Использовали данные из таблиц следующего вида:

substitution\_table = [

    0x53, 0x9b, 0x17, 0x20, 0x22, 0x56, 0x83,

0x08, 0x52, 0xb9, 0x26, 0x50, 0x03, 0x02,

    0x1b, 0x49, 0x2e, 0xd8, 0x9e, 0x48, 0x42,

0x8f, 0x2b, 0xf3, 0x94, 0x4f, 0x9a, 0xe7,

    0x05, 0x66, 0x16, 0x6b, 0xc1, 0x9c, 0xf0,

0xf9, 0x19, 0x5d, 0xd9, 0xed, 0xd7, 0x07,

    0xb4, 0xdc, 0xcc, 0x64, 0xf7, 0x98, 0xb5,

0x95, 0xfe, 0xbc, 0xbd, 0xe6, 0x10, 0x72,

    0x5a, 0x57, 0xcb, 0x4c, 0x3b, 0x8d, 0xf8,

0x5c, 0xf1, 0x34, 0xb1, 0x90, 0x76, 0xca,

    0x23, 0xff, 0x0b, 0xeb, 0x29, 0x2a, 0x5b,

0x92, 0x41, 0x6f, 0x43, 0x91, 0x33, 0x3a,

    0x3f, 0xd6, 0x5f, 0x74, 0x1d, 0xd0, 0x8e,

0xcd, 0xfa, 0xa2, 0xc5, 0xf2, 0x6d, 0x24,

    0x4d, 0xa6, 0x97, 0xb6, 0x80, 0xc4, 0xbe,

0xdf, 0x55, 0xaa, 0x1e, 0x1a, 0x63, 0x1c,

    0xc7, 0x6e, 0x44, 0x09, 0x1f, 0x7f, 0x77,

0x89, 0xd4, 0xae, 0xa0, 0x73, 0x68, 0xbf,

    0x75, 0xd5, 0x81, 0x4e, 0xac, 0xb0, 0x61,

0xd2, 0x0c, 0xe0, 0xe1, 0xba, 0x79, 0x82,

    0xb3, 0xfb, 0xcf, 0x7a, 0x11, 0x30, 0xd3,

0x93, 0xdb, 0xf6, 0x39, 0x38, 0x69, 0x7e,

    0x27, 0xa9, 0xb7, 0x36, 0x8a, 0x65, 0x35,

0x3d, 0x87, 0xfc, 0x21, 0x5e, 0xee, 0x70,

    0x47, 0x32, 0xb2, 0xfd, 0xc8, 0xc6, 0x3e,

0xa1, 0xe8, 0xde, 0x62, 0x00, 0x12, 0x2f,

    0xe5, 0x0f, 0x85, 0x06, 0xea, 0x6c, 0xec,

0x60, 0xc3, 0x8b, 0x46, 0x86, 0xe4, 0xc9,

    0xce, 0xc2, 0x0e, 0x18, 0x01, 0x58, 0xe3,

0x04, 0x67, 0xad, 0x78, 0x6a, 0xd1, 0xb8,

    0x7c, 0x15, 0x54, 0x99, 0x7d, 0xaf, 0x7b,

0xf4, 0x84, 0xa3, 0x8c, 0x88, 0x4b, 0x45,

    0xa8, 0xe9, 0x51, 0xa5, 0xef, 0x71, 0x0a,

0x4a, 0xbb, 0x0d, 0x2c, 0xc0, 0x25, 0xa7,

    0xdd, 0x2d, 0xf5, 0xab, 0xe2, 0x14, 0x13,

0x28, 0x40, 0x37, 0x59, 0x9f, 0x3c, 0xa4,

    0x31, 0x9d, 0xda, 0x96

]

permutation = [

    0x19, 0x04, 0x09, 0x02, 0x0f, 0x14, 0x10, 0x0c, 0x0a, 0x17,

    0x13, 0x00, 0x0e, 0x11, 0x0d, 0x05, 0x12, 0x01, 0x1a, 0x03,

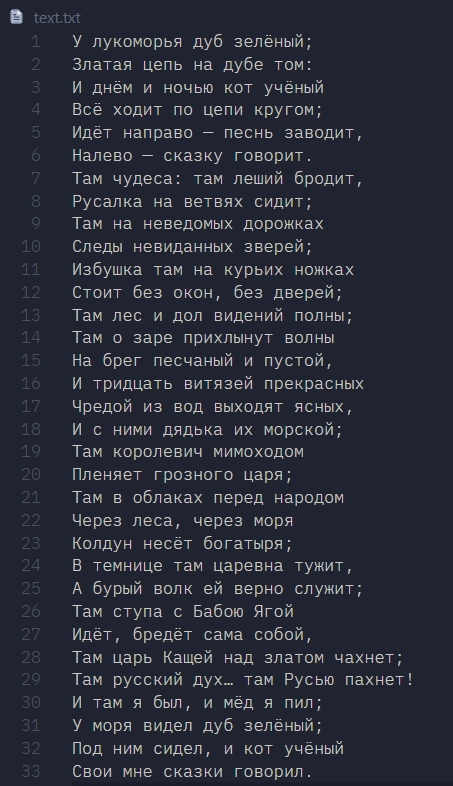
    0x06, 0x15, 0x0b, 0x08, 0x18, 0x16, 0x07

]

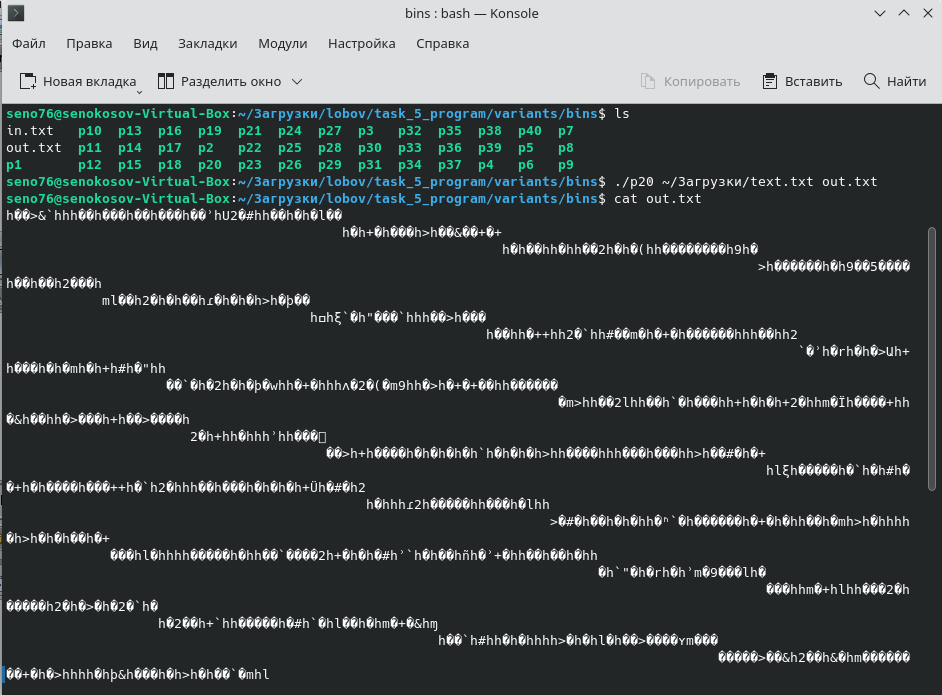
xor\_key = [0] \* 40

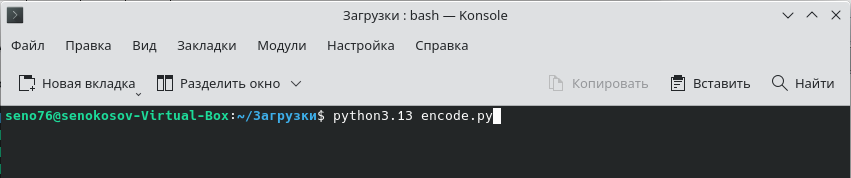
Проверим правильно ли происходит шифрования по нашему реализованному алгоритму и представленному на исследование, путем шифрования одного и того же текста и проверки равенства файлов с помощью утилиты diff в linux.

Исходный текст:



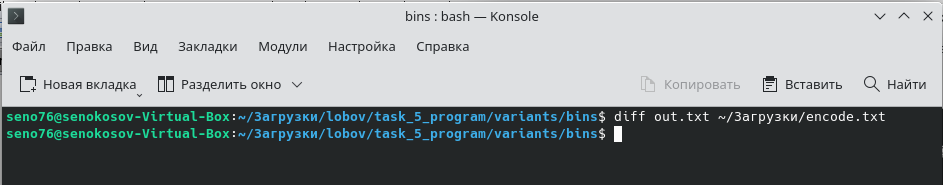
Шифруем программой p20 и собственной реализацией encode.py:







В результате сравниваем полученные файлы c помощью diff и видим, что никаких ошибок не появляется, что говорит о том, что данные идентичные:



Теперь напишем функцию дешифратора:

import data

def main(in\_file, out\_file):

    block = 89

    stream\_2 = open(out\_file, "wb")

    with open(in\_file, "rb") as stream\_1:

        while True:

            chunk = stream\_1.read(block)

            if not chunk:

                break

            decrypted\_chunk = decrypt(chunk, len(chunk))

            stream\_2.write(bytes(decrypted\_chunk))

    stream\_2.close()

def decrypt(\_\_block, \_\_edflag):

    \_\_block = list(\_\_block)

    xor(\_\_block,\_\_edflag)

    substitute(\_\_block,\_\_edflag)

    xor(\_\_block,\_\_edflag)

    permutate(\_\_block,\_\_edflag)

    xor(\_\_block,\_\_edflag)

    substitute(\_\_block,\_\_edflag)

    substitute(\_\_block,\_\_edflag);

    permutate(\_\_block,\_\_edflag);

    substitute(\_\_block,\_\_edflag);

    substitute(\_\_block,\_\_edflag);

    permutate(\_\_block,\_\_edflag);

    substitute(\_\_block,\_\_edflag);

    return \_\_block

def substitute(arr, param\_2):

    for local\_c in range(param\_2):

        arr[local\_c] = data.substitution\_table.index(arr[local\_c])

def xor(arr, param\_2):

    local\_10 = 0

    for local\_c in range(param\_2):

        arr[local\_c] = arr[local\_c] ^ data.xor\_key[local\_10]

        local\_10 = (local\_10 + 1) % 40

def permutate(arr, param\_2):

    new\_stack = [0] \* 40

    for local\_c in range(0, param\_2 - 26, 27):

        for local\_10 in range(27):

            new\_stack[local\_10] = arr[local\_c + data.permutation.index(local\_10)]

        for local\_10 in range(27):

            arr[local\_10 + local\_c] = new\_stack[local\_10]

    print(arr)

print(main("t20.txt", "decoded\_text.txt"))

Запустим этот код от тестового файла t20.txt и убедимся, что был получен расшифрованный текст:

