

МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение  
высшего образования

**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ  
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н.Г. ЧЕРНЫШЕВСКОГО»**

Кафедра теоретических основ  
компьютерной безопасности и  
криптографии

## **ТЕОРИЯ ПСЕВДОСЛУЧАЙНЫХ ГЕНЕРАТОРОВ**

### **ОТЧЕТ ПО ПРАКТИЧЕСКОМУ КУРСУ**

студентки 4 курса 431 группы

специальности 10.05.01 «Компьютерная безопасность»

факультета компьютерных наук и информационных технологий

Кайдышевой Дарьи Сергеевны

Преподаватель

Ст. преподаватель

\_\_\_\_\_

И. И. Слеповичев

Саратов 2024

## СОДЕРЖАНИЕ

<b>1</b>	<b>Задание 1 Генератор псевдослучайных чисел.....</b>	<b>3</b>
1.1	Линейный конгруэнтный метод.....	4
1.2	Аддитивный метод .....	6
1.3	Пятипараметрический метод.....	7
1.4	Регистр сдвига с обратной связью (РСЛОС) .....	9
1.5	Нелинейная комбинация РСЛОС.....	11
1.6	Вихрь Мерсенна.....	13
1.7	RC4.....	17
1.8	ГПСЧ на основе RSA .....	20
1.9	Алгоритм Блюма-Блюма-Шуба .....	22
<b>2</b>	<b>Задание 2 Преобразование ПСЧ к заданному распределению .....</b>	<b>24</b>
2.1	Стандартное равномерное с заданным интервалом.....	25
2.2	Треугольное распределение .....	27
2.3	Общее экспоненциальное распределение .....	29
2.4	Нормальное распределение .....	31
2.5	Гамма распределение (алгоритм для $c = k$ ( $k$ – целое число)) .....	33
2.6	Логнормальное распределение .....	35
2.7	Логистическое распределение.....	37
2.8	Биномиальное распределение .....	39
	<b>ПРИЛОЖЕНИЕ А.....</b>	<b>41</b>
	<b>ПРИЛОЖЕНИЕ Б.....</b>	<b>47</b>

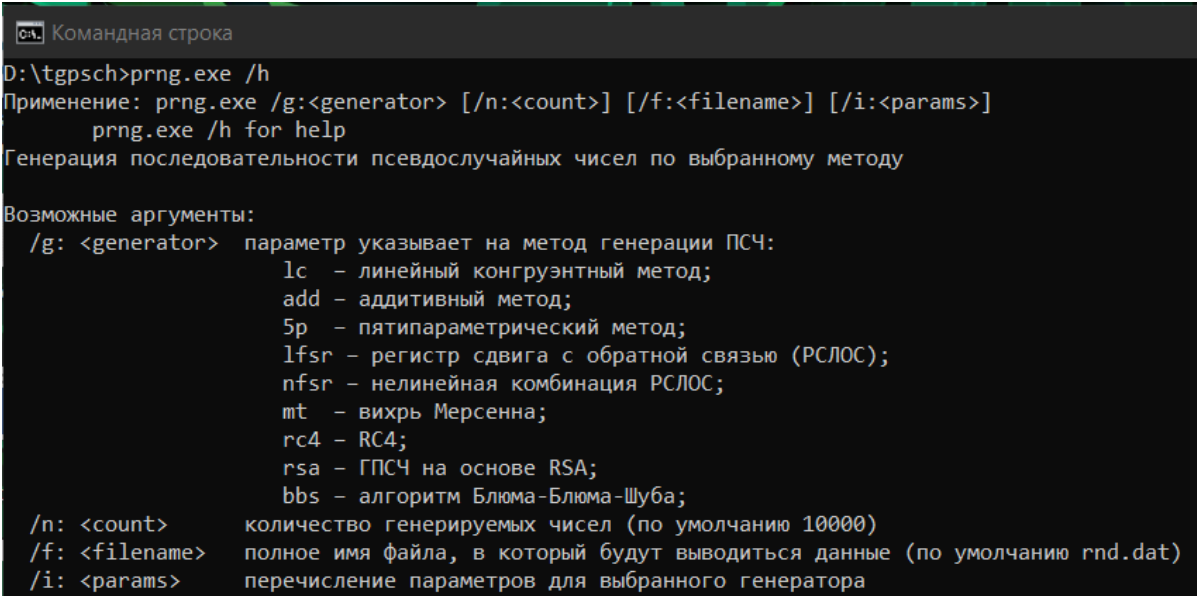
## 1 Задание 1 Генератор псевдослучайных чисел

Создайте программу для генерации псевдослучайных величин следующими алгоритмами:

- a. Линейный конгруэнтный метод;
- b. Аддитивный метод;
- c. Пятипараметрический метод;
- d. Регистр сдвига с обратной связью (РСЛОС);
- e. Нелинейная комбинация РСЛОС;
- f. Вихрь Мерсенна;
- g. RC4;
- h. ГПСЧ на основе RSA;
- i. Алгоритм Блюма-Блюма-Шуба;

Название программы: prng.exe

Для управления приложением предлагается следующий формат параметров командной строки:



```
Командная строка
D:\tgpsch>prng.exe /h
Применение: prng.exe /g:<generator> [/n:<count>] [/f:<filename>] [/i:<params>]
prng.exe /h for help
Генерация последовательности псевдослучайных чисел по выбранному методу

Возможные аргументы:
/g: <generator> параметр указывает на метод генерации ПСЧ:
                  lc - линейный конгруэнтный метод;
                  add - аддитивный метод;
                  5p - пятипараметрический метод;
                  lfsr - регистр сдвига с обратной связью (РСЛОС);
                  nfsr - нелинейная комбинация РСЛОС;
                  mt - вихрь Мерсенна;
                  rc4 - RC4;
                  rsa - ГПСЧ на основе RSA;
                  bbs - алгоритм Блюма-Блюма-Шуба;
/n: <count>      количество генерируемых чисел (по умолчанию 10000)
/f: <filename>    полное имя файла, в который будут выводиться данные (по умолчанию rnd.dat)
/i: <params>      перечисление параметров для выбранного генератора
```

## Рисунок 1

Таблица 1. Порядок элементов вектора параметров (/i:)

Метод	Описание параметров
lc	Модуль, множитель, приращение, начальное значение
add	Модуль, младший индекс, старший индекс, последовательность начальных значений
5p	p, q1, q2, q3, w (см. лекции п. 3.5.2), начальное значение
lfsr	Двоичное представление вектора коэффициентов, начальное значение регистра
nfsr	В алгоритме использовать три РСЛОС R1, R2, R3, скомбинированных функцией $R1 \wedge R2 + R2 \wedge R3 + R3$ . Параметры – двоичное представление векторов коэффициентов для R1, R2, R3, w, x1, x2, x3. w – длина слова, x1, x2, x3 – десятичное представление начальных состояний регистров R1, R2, R3.
mt	Модуль, начальное значение x
rc4	256 начальных значений
rsa	Модуль n, число e, w, начальное значение x. e удовлетворяет условиям: $1 < e < (p-1)(q-1)$ , $\text{НОД}(e, (p-1)(q-1)) = 1$ , где $p \cdot q = n$ . x из интервала [1,n] w – длина слова.
bbs	Начальное значение x (взаимно простое с n). При генерации использовать параметры: $p = 127, q = 131, n = p \cdot q = 16637$

### 1.1 Линейный конгруэнтный метод

Последовательность ПСЧ, получается по формуле:

$$X_{n+1} = (aX_n + c) \bmod m, n \geq 1.$$

В его основе лежит выбор четырех ключевых параметров:

- $m > 0$ , модуль;
- $0 \leq a \leq m$ , множитель;
- $0 \leq c \leq m$ , приращение (инкремент);
- $0 \leq X_0 \leq m$ , начальное значение.

```
D:\tgpsch>prng.exe /g:lc /i:1024,171,513,577 /n:1200 /f:lc.dat
Числа сгенерированы и сохранены в файл lc.dat!

D:\tgpsch>_
```

## Рисунок 2 – Ввод

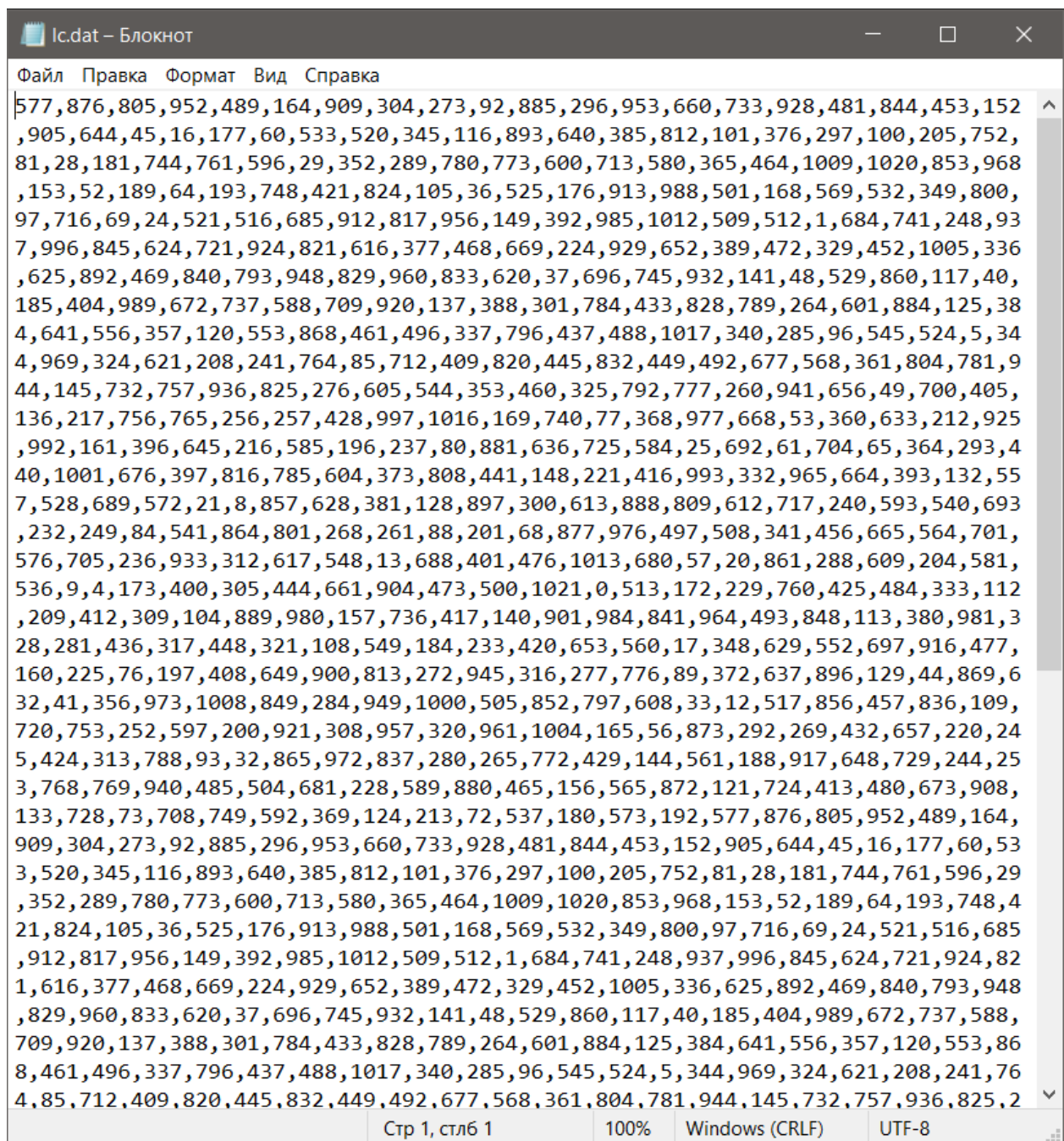


Рисунок 3 – Вывод

```

"""линейный конгруэнтный метод"""
def lc(m, a, c, x0, n_ = 10000):
    x = []
    x.append(x0);
    for i in range(1, n_ - 1):
        x.append(((a * x[i - 1] + c) % m) % 2**10)
    return x

```

Рисунок 4 – Алгоритм

## 1.2 Аддитивный метод

Последовательность ПСЧ, получается по формуле:

$$X_{n+1} = (X_{n-k} + X_{n-j}) \bmod m, \quad j > k \geq 1.$$

В его основе лежит выбор четырех ключевых параметров:

- $m > 0$ , модуль;
- $k$ , младший индекс;
- $j$ , старший индекс;
- последовательность из  $j$  начальных значений.

```
D:\tgpsch>prng.exe /g:add /i:1024,7,11,654,234,654,546,345,875,334,
345,765,546,998,34,756,987,544 /n:1200 /f:add.dat
Числа сгенерированы и сохранены в файл add.dat!

D:\tgpsch>
```

Рисунок 5 – Ввод

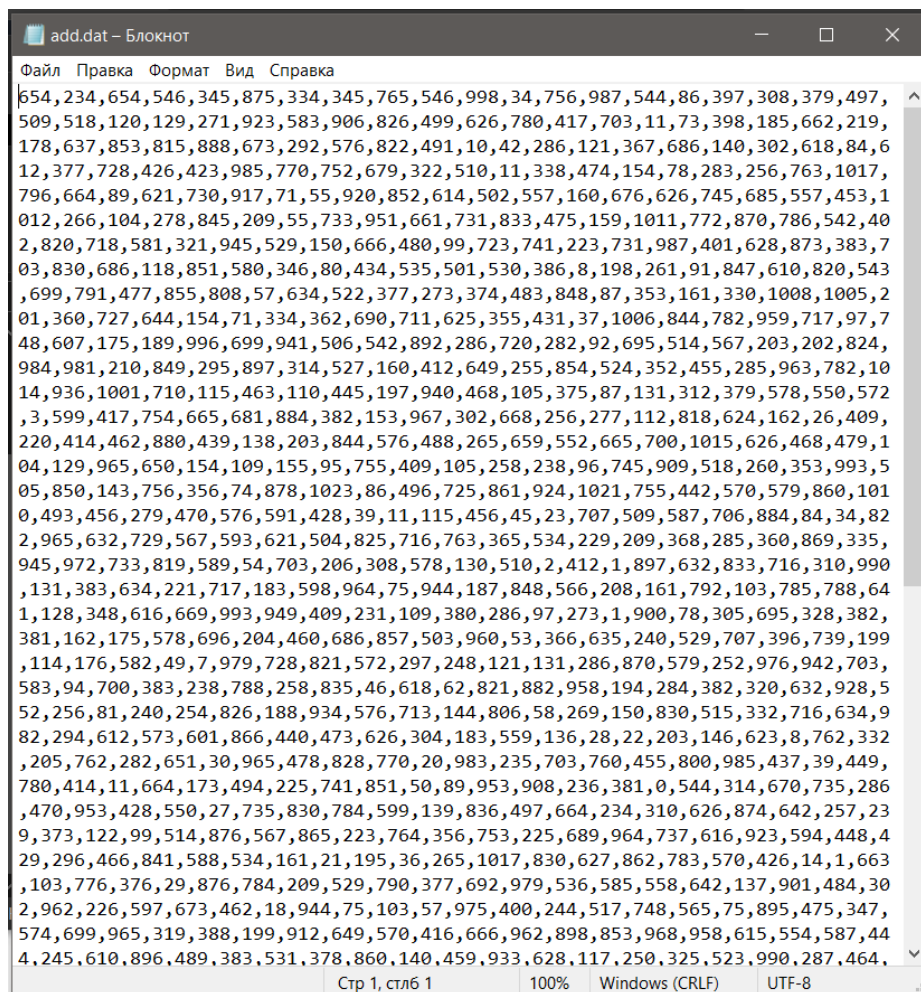


Рисунок 6 – Вывод

```

"""аддитивный метод"""
def add(m, low_i, up_i, start_seq, n_ = 10000):
    x = start_seq
    seq = start_seq.copy()
    n = len(seq)
    l = n_ - n
    for _ in range(1):
        xn = (seq[n - low_i] + seq[n - up_i]) % m
        x.append(xn % 2**10)
        seq.append(xn)
        del seq[0]
    return x

```

Рисунок 7 – Алгоритм

### 1.3 Пятипараметрический метод

Данный метод является частным случаем РСЛОС, использует характеристический многочлен из 5 членов и позволяет генерировать последовательности  $w$ -битовых двоичных целых чисел в соответствии со следующей рекуррентной формулой:

$$X_{n+p} = X_{n+q_1} + X_{n+q_2} + X_{n+q_3} + X_n, \quad n = 1, 2, 3, \dots$$

Параметры  $(p, q_1, q_2, q_3, w)$  и  $X_1, \dots, X_p$ , первоначально задают как начальный вектор.

```

D:\tgpsch>prng.exe /g:5p /i:107,31,57,82,10,111010101011011010 /n
:1200 /f:5p.dat
Числа сгенерированы и сохранены в файл 5p.dat!
D:\tgpsch>

```

Рисунок 8 – Ввод





Рисунок 9 – Вывод

```

"""Sp"""
def _5p(p, q1, q2, q3, w, x0, n_ = 10000):
    x = []
    x0 = int(x0, 2)
    for _ in range(n_):
        cur = 0
        for _ in range(w):
            bit_q1 = (x0 >> p - q1) & 1
            bit_q2 = (x0 >> p - q2) & 1
            bit_q3 = (x0 >> p - q3) & 1
            bit_x0 = x0 & 1
            xor = bit_q1 ^ bit_q2 ^ bit_q3 ^ bit_x0
            cur = (cur << 1) | xor
            x0 = (x0 >> 1) | (xor << p - 1)
        x.append(cur % 2**10)
    return x

```

Рисунок 10 – Алгоритм



#### 1.4 Регистр сдвига с обратной связью (РСЛОС)

*Регистр сдвига с обратной линейной связью (РСЛОС)* – регистр сдвига битовых слов, у которого входной (вдвигаемый) бит является линейной функцией остальных битов. Вдвигаемый вычисленный бит заносится в ячейку с номером 0. Количество ячеек  $p$  называют длиной регистра.

Для натурального  $p$  и  $a_1, a_2, \dots, a_{p-1}$ , принимающих значения 0 или 1, определяют рекуррентную формулу:

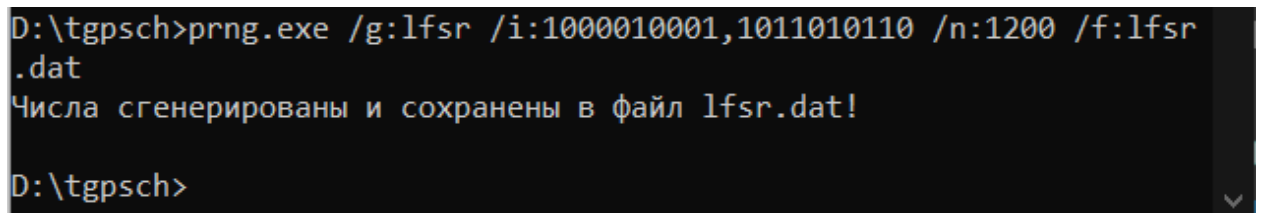
$$X_{n+p} = a_{p-1}X_{n+p-1} + a_{p-2}X_{n+p-2} + \dots + a_1X_{n+1} + X_n, \quad (1)$$

Как видно из формулы, для РСЛОС функция обратной связи является линейной булевой функцией от состояний всех или некоторых битов регистра.

Одна итерация алгоритма, генерирующего последовательность, состоит из следующих шагов:

1. Содержимое ячейки  $p - 1$  формирует очередной бит ПСП битов.
2. Содержимое ячейки 0 определяется значением функции обратной связи, являющейся линейной булевой функцией с коэффициентами  $a_1, a_2, \dots, a_{p-1}$ . Его вычисляют по формуле 1.
3. Содержимое каждого  $i$ -го бита перемещается в  $(i + 1)$ -й,  $0 \leq i < p - 1$ .
4. В ячейку 0 записывается новое содержимое, вычисленное на шаге 2.

Параметры: двоичное представление вектора коэффициентов, начальное значение регистра.



```
D:\tgpsch>prng.exe /g:lfsr /i:1000010001,1011010110 /n:1200 /f:lfsr .dat
Числа сгенерированы и сохранены в файл lfsr.dat!
D:\tgpsch>
```

Рисунок 11 – Ввод

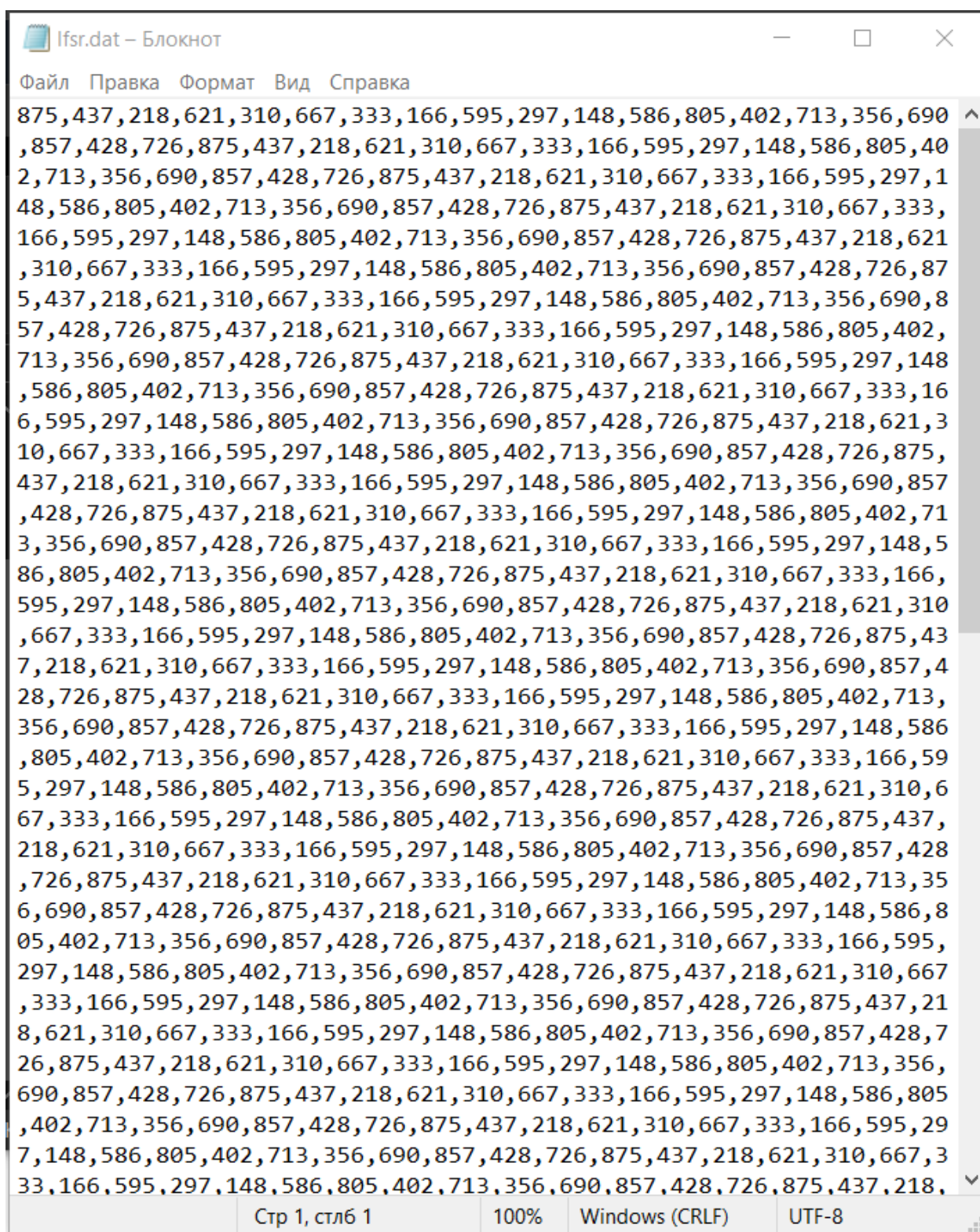


Рисунок 12 – Вывод

```

"""рслос"""
def lfsr(vec_bin, start_reg, n_ = 10000):
    x = []
    reg_len = len(start_reg)
    vec_bin = int(vec_bin, 2)
    start_reg = int(start_reg, 2)
    shift = reg_len - 1
    for _ in range(n_):
        new = (start_reg ^ vec_bin >> shift) & 1
        start_reg = (start_reg >> 1) | (new << (shift))
        x.append(start_reg % 2**10)
    return x

```

Рисунок 13 – Алгоритм

### 1.5 Нелинейная комбинация РСЛОС

Нелинейная функция генератора:

$$f(x_1, x_2, x_3) = x_1x_2 \oplus (1 + x_2)x_3 = x_1x_2 \oplus x_2x_3 \oplus x_3.$$

Параметры: двоичное представление вектора коэффициентов  $R_1, R_2, R_3$ .

```

D:\tgpsch>prng.exe /g:nfsr /i:1001110110,1011010110,1100101001,9,49
1,424,532 /n:1200 /f:nfsr.dat
Числа сгенерированы и сохранены в файл nfsr.dat!

D:\tgpsch>

```

Рисунок 14 – Ввод

nfsrc.dat – Блокнот

Файл Правка Формат Вид Справка

309,275,241,80,463,325,195,65,63,21,12,507,342,306,238,90,54,18,14,5,508,340,204,68,60,20,12,4,3,510,341,307,238,421,355,222,437,364,292,227,417,159,117,44,484,163,414,373,300,283,265,248,424,152,119,466,334,197,444,363,294,285,267,249,87,50,494,165,412,372,211,433,144,399,378,297,280,264,248,87,461,187,105,39,29,11,6,509,340,307,273,240,431,357,220,436,147,398,378,214,77,452,323,318,277,268,260,252,84,51,494,346,201,440,360,216,72,56,23,498,337,304,272,240,80,48,16,15,506,342,205,443,150,397,379,214,434,145,399,133,124,468,179,401,143,122,470,178,110,37,483,161,96,479,330,313,279,242,430,154,118,45,484,348,203,441,151,114,465,335,197,67,62,490,166,98,33,480,351,309,236,420,156,116,44,27,502,338,206,69,451,190,405,371,209,79,58,489,344,311,274,270,250,86,50,17,496,336,207,442,361,295,226,417,352,288,224,95,458,326,194,65,448,320,192,64,63,490,345,311,237,91,54,493,347,201,71,61,20,499,337,207,69,60,491,345,200,440,151,397,132,387,382,298,230,93,459,185,104,472,183,402,369,303,229,92,459,326,317,276,268,251,425,152,392,135,386,382,213,435,145,112,464,176,111,474,329,312,279,269,251,86,461,324,316,235,422,354,222,74,57,488,344,200,71,450,321,319,234,422,157,395,134,386,129,384,383,298,281,264,263,258,257,256,256,255,426,358,221,436,364,219,438,365,292,284,244,83,462,325,316,276,243,430,357,291,225,95,53,19,14,506,169,408,375,301,228,419,353,223,74,454,189,404,371,302,282,246,82,49,495,165,99,33,31,10,505,343,205,68,451,321,192,447,362,294,226,94,53,492,347,310,274,241,431,154,393,376,296,231,418,353,288,287,266,262,253,427,153,119,45,27,9,7,2,510,170,102,34,30,10,6,2,1,511,170,409,375,210,433,367,218,438,146,113,464,335,314,278,242,81,463,186,406,141,388,380,212,76,59,489,167,98,478,181,403,142,389,380,299,281,247,82,462,186,105,472,328,199,445,148,396,132,124,43,486,349,308,275,270,261,259,254,426,153,392,376,215,434,366,218,73,455,189,107,38,482,161,415,138,390,130,126,42,25,503,173,100,476,180,108,36,28,11,505,168,408,136,120,40,24,8,7,509,171,102,477,331,198,445,363,217,72,455,322,318,234,89,456,327,317,235,89,55,18,497,336,304,239,421,156,395,377,215,77,59,22,498,174,101,476,331,313,232,423,354,289,287,245,83,49,16,496,175,410,374,210,78,58,22,13,507,169,103,34,481,351,202,441,360,295,285,244,428,155,393,135,125,43,25,8,504,168,103,477,180,403,369,208,432,144,112,47,485,163,97,32,480,160,96,32,31,501,172,411,374,301,283,246,429,356,291,286,266,249,424,359,290,286,245,428,356,220,75,454,322,193,447,149,115,46,485,348,308,236,91,457,184,407,370,302,229,419,158,394,134,125,468,332,196,67,449,191,106,473,328,312,232,88,55,493,164,412,139,390,381,299,230,418,158,117,467,177,111,37,28,500,172,100,35,481,160,415,373,211,78,453,323,193,64,448,191,405,140,388,131,385,128,384,128,127,469,179,110,474,182,109,475,182,402,142,122,41,487,162,414,138,121,471,178,401,368

Стр 1, стлб 1100%Windows (CRLF)UTF-8

Рисунок 15 – Вывод

```

"""нелинейная комбинация рслос"""
def nfsr(R1, R2, R3, w, x1, x2, x3, n_ = 10000):
    x = []
    lR1 = len(R1)
    lR2 = len(R2)
    lR3 = len(R3)
    for _ in range(n_):
        cur = 0
        for _ in range(w):
            xorR1 = (x1 ^ (x1 >> lR1 - 1))
            xorR2 = (x2 ^ (x2 >> lR2 - 1))
            xorR3 = (x3 ^ (x3 >> lR3 - 1))
            res = ((xorR1 ^ xorR2) + (xorR2 ^ xorR3) + xorR3) & 1
            x1 = (x1 >> 1) | (res << lR1 - 1)
            x2 = (x2 >> 1) | (res << lR2 - 1)
            x3 = (x3 >> 1) | (res << lR3 - 1)
            cur = (cur << 1) | res
        x.append(cur % 2**10)
    return x

```

Рисунок 16 – Алгоритм

## 1.6 Вихрь Мерсенна

Метод Вихрь Мерсенна позволяет генерировать последовательность двоичных псевдослучайных целых  $w$ -битовых чисел в соответствии со следующей рекуррентной формулой

$$X_{n+p} = X_{n+q} \oplus (X_n^r | X_{n+1}^l)A \quad (n = 0, 1, 2, \dots),$$

где  $p, q, r$  – целые константы,  $p$  – степень рекуррентности,  $1 \leq q \leq p$ ;

$X_n$  –  $w$ -битовое двоичное целое число;

$(X_n^r | X_{n+1}^l)$  – двоичное целое число, полученное конкатенацией чисел

$X_n^r$  и  $X_{n+1}^l$ , когда первые  $(w-r)$  битов взяты из  $X_n$ , а последние  $r$  битов из  $X_{n+1}$

в том же порядке;

$A$  – матрица размера  $w \times w$ , состоящая из нулей и единиц, определенная посредством  $a$ ;

$XA$  – произведение, при вычислении которого сначала выполняют операцию  $X \gg 1$  (сдвига битов на одну позицию вправо), если последний бит  $X$  равен 0, а затем, когда последний бит  $X = 1$ , вычисляют  $XA = (X \gg 1) \oplus a$ ,

$$a = (a_{w-1}, a_{w-2}, \dots, a_0),$$

$$X = (x_{w-1}, x_{w-2}, \dots, x_0),$$

$$A = \begin{pmatrix} 0 & 1 & 0 & \dots & \dots & 0 \\ 0 & 0 & 1 & & & 0 \\ 0 & 0 & 0 & \ddots & & 0 \\ \vdots & \dots & \dots & & \ddots & \vdots \\ 0 & 0 & & & & 1 \\ a_{w-1} & a_{w-2} & \dots & \dots & \dots & a_0 \end{pmatrix}.$$

Алгоритм Вихрь Мерсенна состоит из попеременного выполнения процедур *рекурсивной генерации* и «заковки». Рекурсивная генерация представляет из себя РСЛОС с дополнительной рекурсивной функцией для потока выходных битов. Операция «заковки» является процедурой, усиливающей равномерность распределения на больших размерностях битовых векторов.

### Шаги алгоритма.

**Шаг 1а.** Инициализируются значения  $u, h, a$  по формуле:

$u := (1, 0, \dots, 0)$  – всего  $w - r$  бит,  $h := (0, 1, \dots, 1)$  – всего  $r$  бит,

$a := (a_{w-1}, a_{w-2}, \dots, a_0)$  – последняя строка матрицы  $A$ .

**Шаг 1б.**  $X_0, X_1, \dots, X_{p-1}$  заполняются начальными значениями.

**Шаг 2.** Вычисляется  $Y := (y_0, y_1, \dots, y_{w-1}) := (X_n^r | X_{n+1}^l)$ .

**Шаг 3.** Вычисляется новое значение  $X_i$ :

$X_n := X_{(n+q) \bmod p} \oplus (Y \gg 1) \oplus a$ , если младший бит  $y_0 = 1$ ;

$X_n := X_{(n+q) \bmod p} \oplus (Y \gg 1) \oplus 0$ , если младший бит  $y_0 = 0$ ;

**Шаг 4.** Вычисляется  $X_i T$ .

$$Y := X_n,$$



$$Y := Y \oplus (Y \gg u),$$
$$Y := Y \oplus ((Y \ll s) \cdot b),$$
$$Y := Y \oplus ((Y \ll t) \cdot c),$$
$$Z := Y \oplus (Y \gg l).$$

Z подается на выход, как результат.

**Шаг 5.**  $n := (n + 1) \bmod p$ . Переход на шаг 2.

```
D:\tgpsch>prng.exe /g:mt /i:4563 /n:1200 /f:mt.dat
Числа сгенерированы и сохранены в файл mt.dat!

D:\tgpsch>
```

Рисунок 17 – Ввод

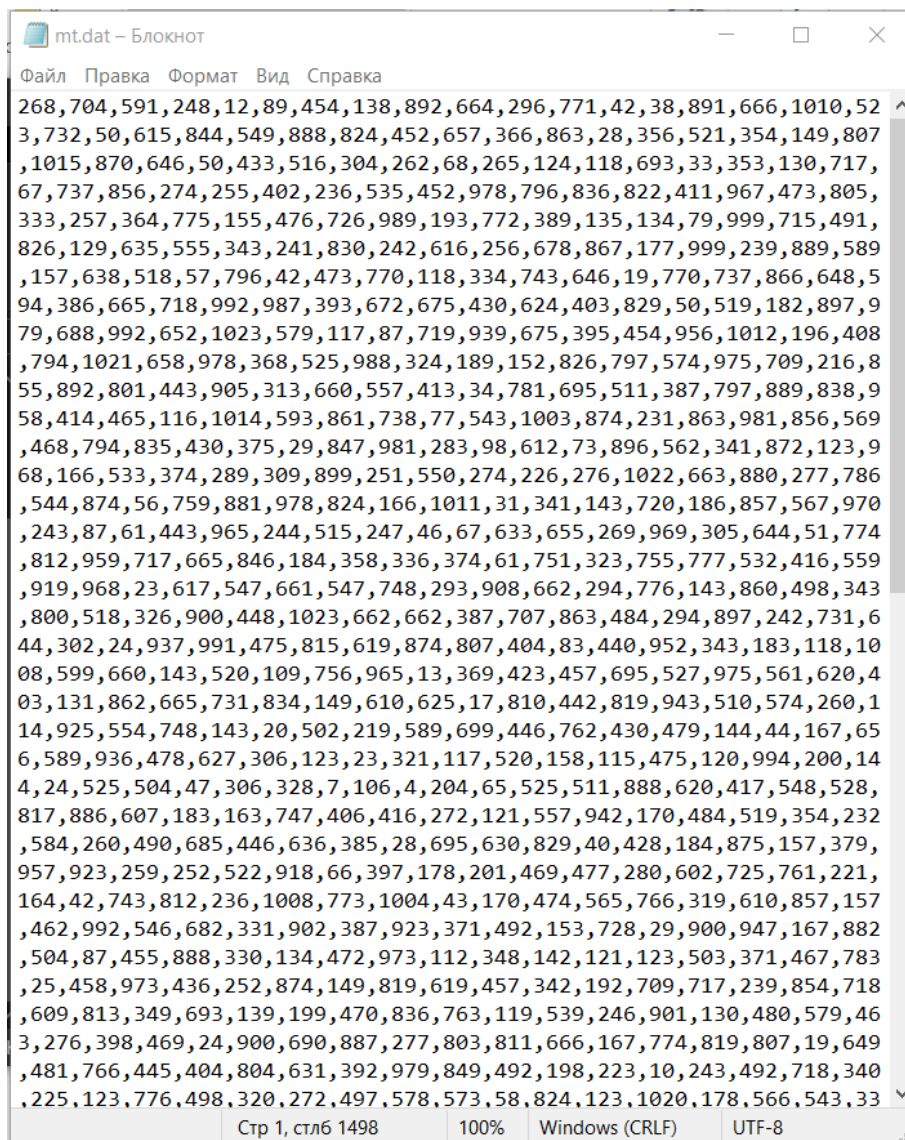


Рисунок 18 – Вывод



```

"""вихрь мерсенна"""
def mt(x0, n = 624, n_ = 10000):
    w = 32
    r = 31
    m = 397
    a = 0x9908B0DF
    u = 11
    d = 0xFFFFFFFF
    s = 7
    t = 15
    l = 18
    b = 0x9D2C5680
    c = 0xEFC60000
    f = 1812433253
    lst = [0] * n
    ind = n + 1

    def f1(core):
        lst[0] = core
        for i in range(1, n):
            tmp = f * (lst[i - 1] ^ (lst[i - 1] >> (w - 2))) + i
            lst[i] = tmp & d

    def f2():
        nonlocal ind
        if ind >= n:
            f3()
            ind = 0

```

Рисунок 19 – Алгоритм 1

```

→ ind = 0
→ y = lst[ind]
→ y = y ^ ((y >> u) & d)
→ y = y ^ ((y << s) & b)
→ y = y ^ ((y << t) & c)
→ y = y ^ (y >> 1)
→ ind = ind + 1
→ return y & d

def f3():
    for i in range(n):
        x = (lst[i] >> r) + (lst[(i + 1) % n] & ((1 << r) - 1))
        x_a = x >> 1
        lst[i] = lst[(i + m) % n] ^ x_a
        if (x % 2) != 0:
            lst[i] ^= a

f1(x0)
res = []
for _ in range(n_):
    o = f2()
    res.append(o % 2**10)
return res

```

Рисунок 20 – Алгоритм 2

**Параметры алгоритма Вихрь Мерсенна:**  $p = 624$ ,  $w = 32$ ,  $r = 31$ ,  $q = 397$ ,  $a = 2567483615$  ( $9908B0DF_{16}$ ),  $u = 11$ ,  $s=7$ ,  $t=15$ ,  $l = 18$ ,  $b = 2636928640$  ( $9D2C5680_{16}$ ),  $c = 4022730752$  ( $EFC60000_{16}$ ).

Вводимые параметры: начальное значение.

## 1.7 RC4

### Описание алгоритма.

1. Инициализация  $S_i$ ,  $i = 0, 1, \dots, 255$ .

a) *for*  $i = 0$  *to* 255:  $S_i = i$ ;

b)  $j = 0$ ;

c) *for*  $i = 0$  *to* 255:  $j = (j + S_i + K_i) \bmod 256$ ;  $Swap(S_i, S_j)$

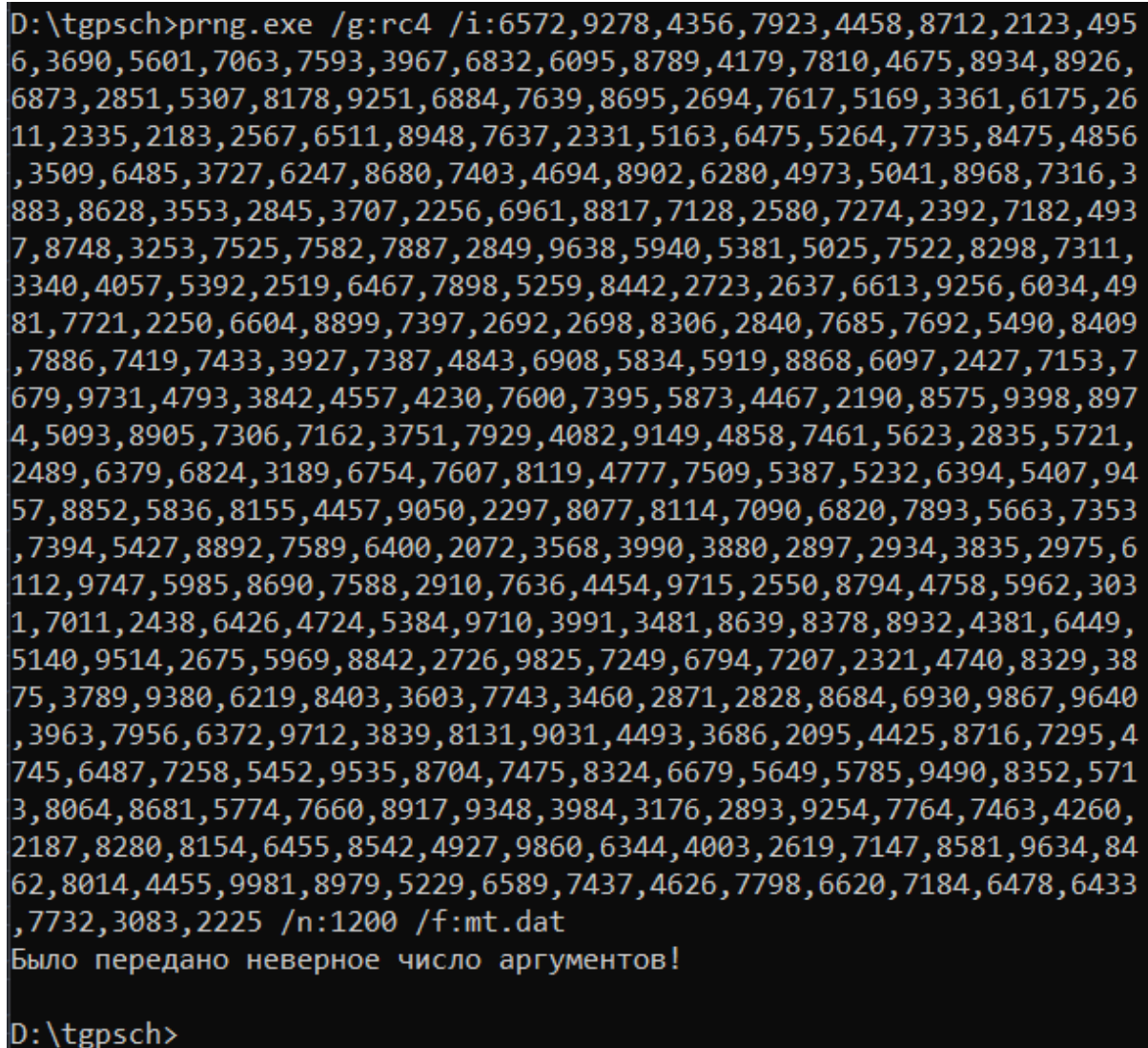
2.  $i = 0, j = 0$ .

3. Итерация алгоритма:

a)  $i = (i + 1) \bmod 256$ ;

- b)  $j = (j + S_i) \bmod 256$ ;
- c)  $Swap(S_i, S_j)$ ;
- d)  $t = (S_i + S_j) \bmod 256$ ;
- e)  $K = S_t$ ;

Параметры: 256 начальных значений.



```
D:\tgpsch>prng.exe /g:rc4 /i:6572,9278,4356,7923,4458,8712,2123,495
6,3690,5601,7063,7593,3967,6832,6095,8789,4179,7810,4675,8934,8926,
6873,2851,5307,8178,9251,6884,7639,8695,2694,7617,5169,3361,6175,26
11,2335,2183,2567,6511,8948,7637,2331,5163,6475,5264,7735,8475,4856
,3509,6485,3727,6247,8680,7403,4694,8902,6280,4973,5041,8968,7316,3
883,8628,3553,2845,3707,2256,6961,8817,7128,2580,7274,2392,7182,493
7,8748,3253,7525,7582,7887,2849,9638,5940,5381,5025,7522,8298,7311,
3340,4057,5392,2519,6467,7898,5259,8442,2723,2637,6613,9256,6034,49
81,7721,2250,6604,8899,7397,2692,2698,8306,2840,7685,7692,5490,8409
,7886,7419,7433,3927,7387,4843,6908,5834,5919,8868,6097,2427,7153,7
679,9731,4793,3842,4557,4230,7600,7395,5873,4467,2190,8575,9398,897
4,5093,8905,7306,7162,3751,7929,4082,9149,4858,7461,5623,2835,5721,
2489,6379,6824,3189,6754,7607,8119,4777,7509,5387,5232,6394,5407,94
57,8852,5836,8155,4457,9050,2297,8077,8114,7090,6820,7893,5663,7353
,7394,5427,8892,7589,6400,2072,3568,3990,3880,2897,2934,3835,2975,6
112,9747,5985,8690,7588,2910,7636,4454,9715,2550,8794,4758,5962,303
1,7011,2438,6426,4724,5384,9710,3991,3481,8639,8378,8932,4381,6449,
5140,9514,2675,5969,8842,2726,9825,7249,6794,7207,2321,4740,8329,38
75,3789,9380,6219,8403,3603,7743,3460,2871,2828,8684,6930,9867,9640
,3963,7956,6372,9712,3839,8131,9031,4493,3686,2095,4425,8716,7295,4
745,6487,7258,5452,9535,8704,7475,8324,6679,5649,5785,9490,8352,571
3,8064,8681,5774,7660,8917,9348,3984,3176,2893,9254,7764,7463,4260,
2187,8280,8154,6455,8542,4927,9860,6344,4003,2619,7147,8581,9634,84
62,8014,4455,9981,8979,5229,6589,7437,4626,7798,6620,7184,6478,6433
,7732,3083,2225 /n:1200 /f:mt.dat
Было передано неверное число аргументов!

D:\tgpsch>
```

Рисунок 21 – Ввод

rc4.dat – Блокнот

Файл Правка Формат Вид Справка

```
6,85,60,3,91,253,243,54,58,179,63,32,44,86,60,60,205,235,7,27,148,1
0,64,190,205,29,198,243,162,234,68,135,108,245,72,243,49,251,107,12
,204,140,234,105,86,145,115,93,128,56,144,146,60,53,35,182,181,3,35
,134,192,103,247,49,126,75,129,155,107,221,233,190,12,233,132,220,3
6,53,84,85,49,202,67,4,57,163,213,123,51,230,231,148,181,97,35,161,
74,36,62,173,206,22,10,142,25,8,219,172,155,192,204,223,66,20,108,1
48,41,39,138,242,237,18,162,162,203,163,237,171,250,211,101,25,139,
7,179,13,145,146,62,56,216,222,20,134,245,111,81,168,104,205,202,25
3,35,95,229,30,48,157,196,26,208,162,5,163,132,101,234,220,127,176,
9,1,152,119,75,104,186,159,208,167,150,205,105,27,24,195,132,55,146
,21,49,177,11,34,167,201,24,198,150,101,179,39,246,11,229,129,30,15
0,217,44,1,124,175,221,25,117,117,38,16,52,65,141,41,141,80,182,190
,123,254,104,99,198,160,52,240,219,129,36,150,253,71,182,153,70,227
,90,78,33,215,202,41,36,253,61,89,108,26,17,35,16,186,60,11,108,90,
51,199,201,241,143,142,163,147,167,6,234,19,154,60,211,32,92,136,24
4,106,160,94,129,218,212,56,159,228,86,245,168,130,142,229,108,71,1
18,52,113,152,1,159,20,98,11,163,127,115,228,167,129,58,106,114,249
,61,234,17,163,226,210,253,165,119,197,117,141,101,72,48,201,51,70,
172,248,92,249,221,26,11,242,170,172,69,111,123,224,150,215,108,81,
164,208,179,238,66,190,103,247,140,69,207,47,255,9,90,77,125,69,192
,220,35,107,71,204,75,179,243,250,101,18,230,125,211,3,8,89,229,200
,98,38,26,135,198,22,249,0,205,200,121,115,17,31,7,229,62,11,145,13
0,83,253,9,241,199,168,172,37,32,4,220,251,253,194,224,109,220,78,5
0,42,62,76,223,89,159,32,71,176,88,212,211,104,238,169,113,171,74,4
9,238,164,250,206,94,24,1,205,67,235,199,9,105,191,239,152,82,71,11
7,112,50,198,194,64,11,45,44,175,53,164,116,252,11,134,228,5,227,26
,225,91,54,90,255,30,38,116,196,68,145,77,25,78,222,123,52,154,120,
53,89,29,34,189,109,79,202,231,210,32,63,148,119,104,244,29,250,203
,201,25,119,105,131,52,183,103,203,48,38,194,250,105,242,162,8,57,7
2,179,124,12,86,73,192,87,98,193,61,20,239,246,99,118,255,156,225,4
8,253,224,132,207,102,239,133,222,118,102,188,110,158,120,249,143,6
2,121,130,129,225,222,160,233,93,148,43,165,172,104,142,28,195,88,6
2,166,166,139,53,122,56,122,10,227,144,189,251,25,163,29,165,87,152
,254,95,55,91,1,51,14,167,230,90,27,161,137,144,183,63,90,204,229,1
85,190,105,244,223,97,247,50,205,132,133,66,165,21,73,34,229,88,247
,73,222,123,197,39,192,225,245,152,41,237,2,29,56,145,169,205,102,7
```

Стр 1, столб 1      100%      Windows (CRLF)      UTF-8

Рисунок 22 – Вывод

```

"""rc4"""
def rc4(k, count_num = 10000):
    len_k = len(k)
    S = [i for i in range(256)]
    j = 0
    for i in range(256):
        j = (j + S[i] + k[i % len_k]) % 256
        S[i], S[j] = S[j], S[i]
    i = 0
    j = 0
    Ks = []
    for _ in range(1, count_num + 1):
        i = (i + 1) % 256
        j = (j + S[i]) % 256
        S[i], S[j] = S[j], S[i]
        cur = S[(S[i] + S[j]) % 256]
        Ks.append(cur % 2**10)
    return Ks

```

Рисунок 23 – Алгоритм

## 1.8 ГПСЧ на основе RSA

### Описание алгоритма:

1. Сгенерировать два секретных простых числа  $p$  и  $q$ , а также  $n = pq$  и  $f = (p - 1)(q - 1)$ . Выбрать случайное целое число  $e$ ,  $1 < e < f$ , такое что  $\text{НОД}(e, f) = 1$ .
2. Выбрать случайное целое  $x_0$  – начальный вектор из интервала  $[1, n - 1]$ .
3. *For*  $i = 1$  *to*  $l$  *do*
  - a.  $x_i \leftarrow x_{i-1}^e \bmod n$ .
  - b.  $z_i \leftarrow$  последний значащий бит  $x_i$
4. Вернуть  $z_1, z_2, \dots, z_l$ .



Параметры: Модуль  $n$ , число  $e$ ,  $w$ , начальное значение  $x$ .  $e$  удовлетворяет условиям:  $1 < e < (p-1)(q-1)$ ,  $\text{НОД}(e, (p-1)(q-1)) = 1$ , где  $p * q = n$ ,  $x \in [1, n]$ ,  $w$  – длина слова.

```
D:\tgpsch>prng.exe /g:rsa /i:10967,571,77,10 /n:1200 /f:rsa.dat
Числа сгенерированы и сохранены в файл rsa.dat!

D:\tgpsch>
```

Рисунок 24 – Ввод

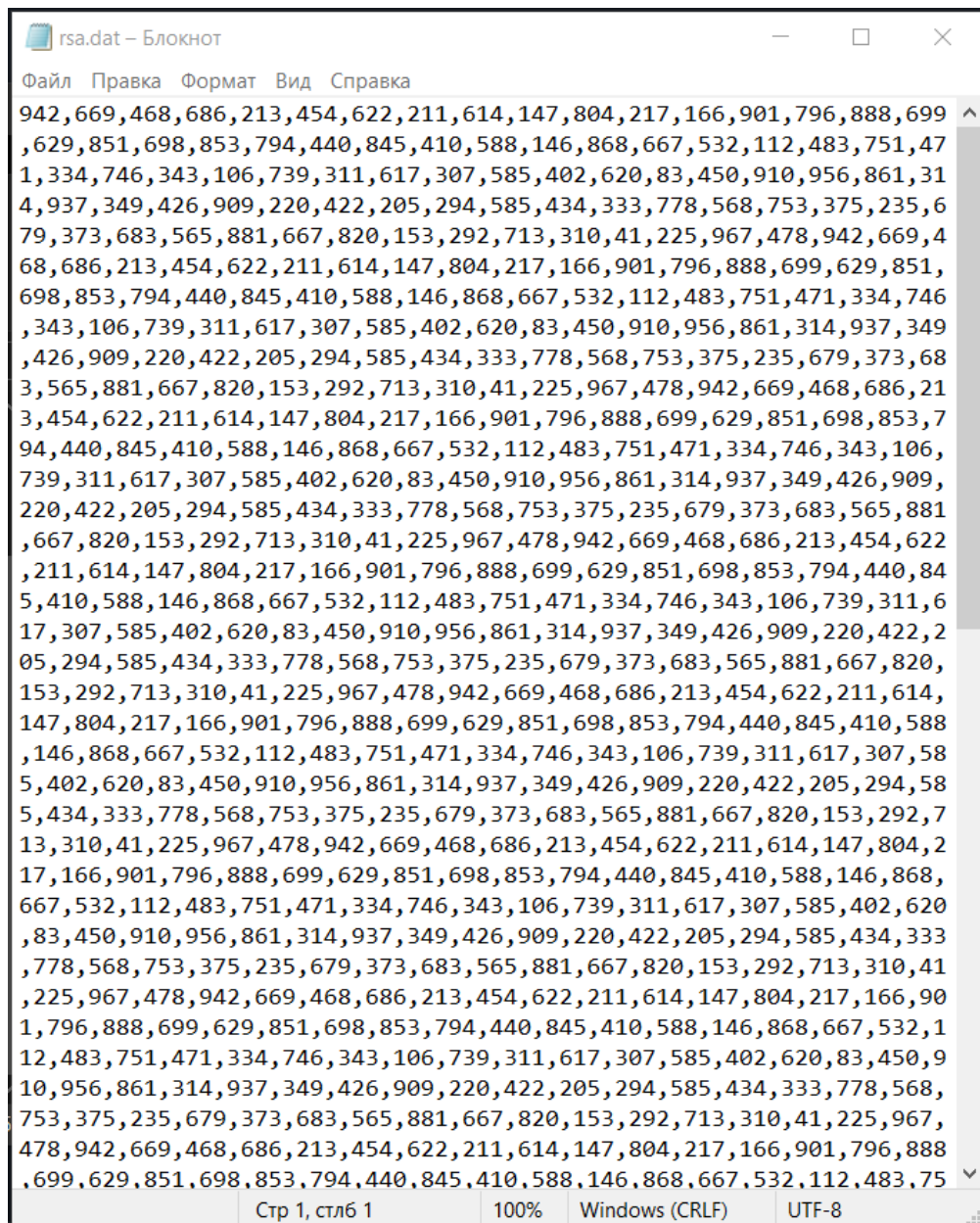


Рисунок 25 – Вывод

```

"""ГПСЧ на основе rsa"""
def rsa(n, e, w, x0, n_ = 10000):
    x = []
    for _ in range(n_):
        cur = 0
        for _ in range(w):
            x0 = pow(x0, e, n)
            cur = (cur << 1) | (x0 & 1)
        x.append(cur % 2**10)
    return x

```

Рисунок 26 – Алгоритм

### 1.9 Алгоритм Блюма-Блюма-Шуба

#### Описание алгоритма:

**На входе:** Длина  $l$ .

**На выходе:** Последовательность псевдослучайных бит  $z_1, z_2, \dots, z_l$ .

1. Сгенерировать два простых числа  $p$  и  $q$ , сравнимых с 3 по модулю 4. Это гарантирует, что каждый квадратичный вычет имеет один квадратный корень, который также является квадратичным вычетом. Произведение этих чисел –  $n=pq$  является целым числом Блюма. Выберем другое случайное целое число  $x$ , взаимно простое с  $n$ .
2. Вычислим  $x_0 = x^2 \bmod n$ , которое будет начальным вектором.
3. *For*  $i = 1$  *to*  $l$  *do*
  1.  $x_i \leftarrow x_{i-1}^2 \bmod n$ .
  2.  $z_i \leftarrow$  последний значащий бит  $x_i$
4. Вернуть  $z_1, z_2, \dots, z_l$ .

Интересным достоинством этого генератора является то, что для получения  $i$ -го бита  $b_i$  при известных  $p$  и  $q$  достаточно воспользоваться формулой



$$b_i = x_0^{2^i \bmod ((p-1)(q-1))} \bmod 2.$$

Параметры: начальное значение и длина слова.

```
D:\tgpsch>prng.exe /g:bbs /i:791,5 /n:1200 /f:bbs.dat
Числа сгенерированы и сохранены в файл bbs.dat!

D:\tgpsch>
```

Рисунок 27 – Ввод

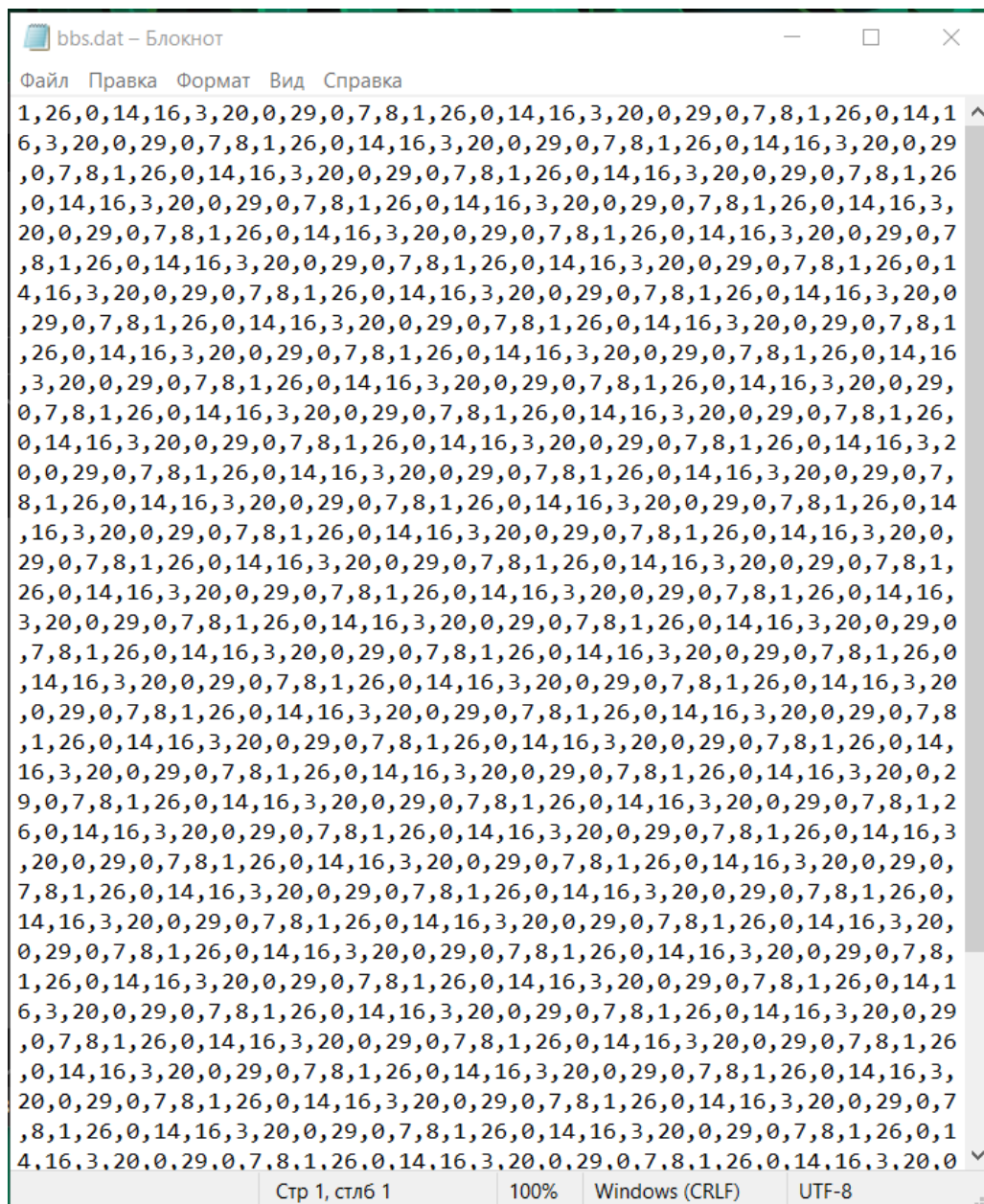


Рисунок 28 – Вывод

```

"""алгоритм блюма-блюма-шуба"""
def bbs(x0, l, n_ = 10000):
    p = 127
    q = 131
    n = p * q
    x = []
    for _ in range(n_):
        cur = 0
        for _ in range(l):
            x0 = pow(x0, 2, n)
            cur = (cur << 1) | (x0 & 1)
        x.append(cur % 2**10)
    return x

```

Рисунок 29 – Алгоритм

## 2 Задание 2 Преобразование ПСЧ к заданному распределению

Создать программу для преобразования последовательности ПСЧ в другую последовательность ПСЧ с заданным распределением:

- a. Стандартное равномерное с заданным интервалом;
- b. Треугольное распределение;
- c. Общее экспоненциальное распределение;
- d. Нормальное распределение;
- e. Гамма распределение (для параметра  $c=k$ );
- f. Логнормальное распределение;
- g. Логистическое распределение;
- h. Биномиальное распределение.

Название программы: rnc.exe

Текстовый файл с десятичными числами (разделитель – любой), интервал преобразуемых значений, параметры распределения.

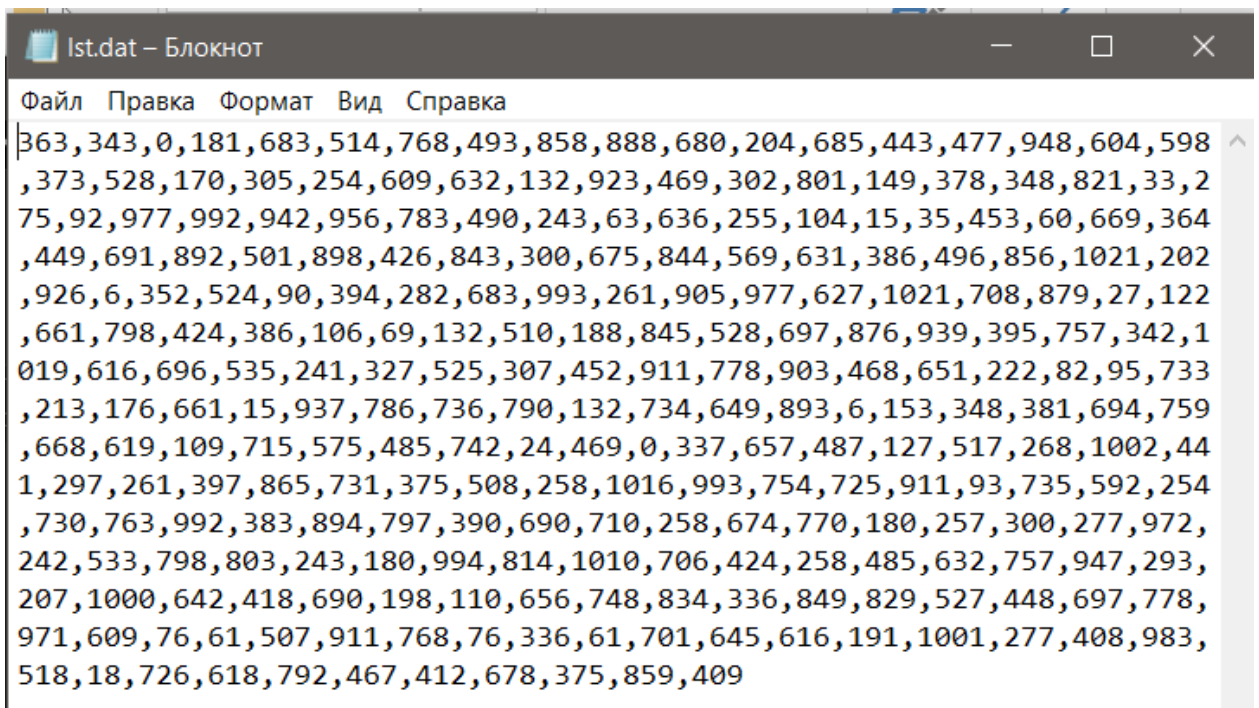


Рисунок 30

Для управления приложением предлагается следующий формат параметров командной строки:

```
D:\tgpsch\task2>rnc.exe /h
Применение: rnc.exe [/f:<filename>] /d:<distribution> /p1:<param1> /p2:<param2> [/p3:<param3>]
rnc.exe /h for help
Преобразование ПСЧ к заданному распределению

Возможные аргументы:
/f: <filename> полное имя файла, из которого будет браться начальная последовательность (по умолчанию lst.dat)
/d: <distribution> код распределения для преобразования последовательности:
    st – стандартное равномерное с заданным интервалом;
    tr – треугольное распределение;
    ex – общее экспоненциальное распределение;
    nr – нормальное распределение;
    gm – гамма распределение;
    ln – логнормальное распределение;
    ls – логистическое распределение;
    rsa – ГПСЧ на основе RSA;
    bi – биномиальное распределение.
/p1:<parameter1> 1-й параметр, необходимый, для генерации ПСЧ заданного распределения
/p1:<parameter2> 2-й параметр, необходимый, для генерации ПСЧ заданного распределения
/p1:<parameter3> 3-й параметр, необходимый, для генерации ПСЧ заданного распределения

D:\tgpsch\task2>
```

Рисунок 31

Если максимальное значение равномерного целого случайного числа  $X$  равно  $(m - 1)$ , для генерации стандартных равномерных случайных чисел необходимо применять следующую формулу:  $U = X/m$ .

## 2.1 Стандартное равномерное с заданным интервалом

Равномерное случайное число должно быть получено в соответствии со следующей формулой:

$$Y = bU + a.$$

```
D:\tgpsch\task2>rng.exe /d:st /p1:117 /p2:33124
Числа сгенерированы и сохранены в файл distr-st.dat!
D:\tgpsch\task2>
```

Рисунок 32 – Ввод



Рисунок 33 – Вывод

```

"""стандартное равномерное распределение"""
def st(a, b, lst):
    res = []
    X_ = max(lst)
    m = X_ + 1
    for x in lst:
        U = x / m
        Y = b * U + a
        res.append(Y)
    return res

```

Рисунок 34 – Алгоритм

## 2.2 Треугольное распределение

Случайное число  $Y$ , подчиняющееся треугольному распределению, определяют по формуле:

$$Y = a + b(U_1 + U_2 - 1).$$

```

D:\tgpsch\task2>rnc.exe /d:tr /p1:273 /p2:1713
Числа сгенерированы и сохранены в файл distr-tr.dat!

```

Рисунок 35 – Ввод



distr-tr.dat – Блокнот

Файл Правка Формат Вид Справка

```
-256.6555772994129,-865.0890410958905,-  
1136.6213307240705,8.17221135029348,566.3219178082193,708.792563600  
7826,673.5939334637965,824.4452054794519,1486.514677103718,1188.164  
3835616437,41.69471624266137,50.075342465753494,450.66927592955017,  
102.03522504892359,948.4784735812132,1161.3463796477495,574.7025440  
313109,187.51761252446198,70.18884540117423,-270.0645792563603,-  
643.8405088062623,-  
503.04598825831704,6.49608610567509,640.0714285714288,-  
159.44031311154606,328.312133072407,893.1663405088061,-  
147.70743639921727,408.76614481408996,152.31898238747556,-  
556.6819960861056,-223.133072407045,519.3904109589039,-  
8.589041095890423,-923.7534246575342,-  
824.8620352250489,351.7778864970646,1860.2906066536202,1801.6262230  
919765,1741.2857142857144,1474.7818003913894,693.707436399217,-  
211.40019569471633,-927.1056751467711,-  
268.3884540117417,53.42759295499019,-838.271037181996,-  
1240.5410958904108,-1356.1937377690804,-622.0508806262231,-  
580.1477495107633,-218.10469667318972,291.4373776908026,-  
77.31017612524454,470.78277886497074,1213.3062622309196,894.8424657  
534244,904.8992172211349,779.1898238747555,687.0029354207436,475.81  
115459882574,194.22211350293537,1106.0342465753424,928.364970645792  
6,571.3502935420743,264.6193737769079,38.34246575342448,826.1213307  
240703,1706.0870841487279,609.9011741682974,450.6692759295498,122.1  
4872798434453,-839.9471624266143,28.285714285714192,-  
410.8590998043053,-628.7553816046967,-  
306.93933463796486,177.46086105675153,1369.1859099804306,661.861056  
7514677,514.3620352250489,1714.46771037182,1248.504892367906,1322.2  
544031311154,1458.0205479452052,1220.0107632093932,78.5694716242661  
6,-1190.2573385518592,-  
127.59393346379647,1005.4667318982389,608.225048923679,-  
82.3385518591,-615.3463796477496,-1146.6780821917807,-  
1103.0988258317027,-363.92759295499036,-  
270.0645792563603,291.4373776908026,861.3199608610568,613.253424657  
5344,1196.5450097847358,1602.1673189823873,795.9510763209393,490.89  
62818003913,402.06164383561656,841.2064579256358,1300.464774951076,  
759.0763209393344,623.3101761252445,-139.32681017612532,-
```

Стр 1, стлб 234 100% Windows (CRLF) UTF-8

Рисунок 36 – Вывод

```

"""треугольное распределение"""
def tr(a, b, lst):
    res = []
    X_ = max(lst)
    m = X_ + 1
    lst = [x / m for x in lst]
    lst.append(lst[-1])
    for i in range(len(lst) - 1):
        U1 = lst[i]
        U2 = lst[i + 1]
        Y = a + b * (U1 + U2 - 1)
        res.append(Y)
    return res

```

Рисунок 37 – Алгоритм

### 2.3 Общее экспоненциальное распределение

Случайное число, соответствующее экспоненциальному распределению, получают по формуле

$$Y = -b \ln(U) + a.$$

```

D:\tgpsch\task2>rnc.exe /d:ex /p1:273 /p2:1723
Числа сгенерированы и сохранены в файл distr-ex.dat!
D:\tgpsch\task2>

```

Рисунок 38 – Ввод



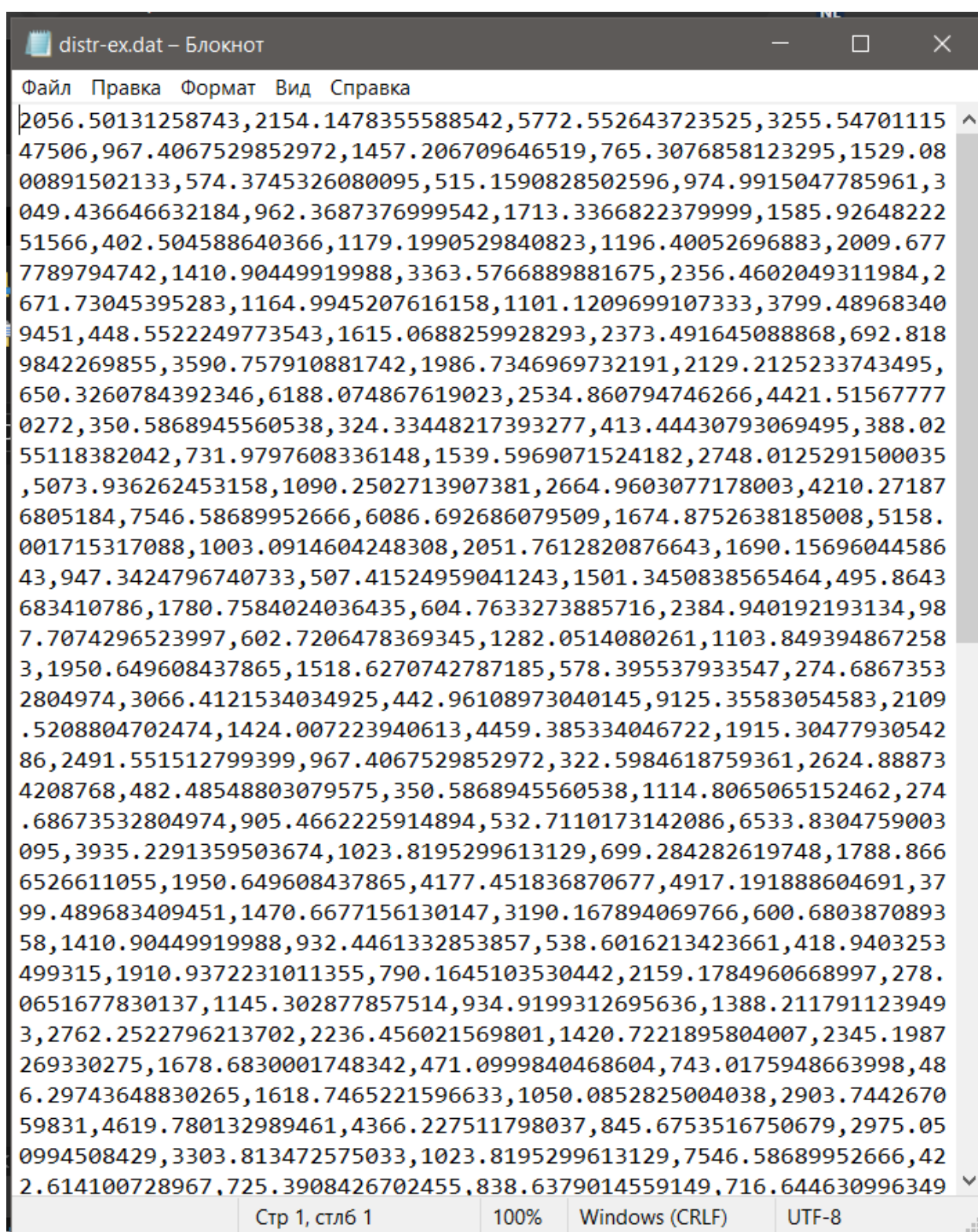


Рисунок 39 – Вывод

```

"""общее экспоненциальное распределение"""
def ex(a, b, lst):
    res = []
    X_ = max(lst)
    m = X_ + 1
    for x in lst:
        U = x / m
        Y = -b * math.log(U) + a
        res.append(Y)
    return res

```

Рисунок 40 – Алгоритм

## 2.4 Нормальное распределение

Два независимых нормальных случайных числа  $Z_1, Z_2$  получают в соответствии со следующей процедурой:

$$Z_1 = \mu + \sigma \sqrt{-2 \ln(1 - U_1)} \cos(2\pi U_2),$$

$$Z_2 = \mu + \sigma \sqrt{-2 \ln(1 - U_1)} \sin(2\pi U_2).$$

```

D:\tgpsch\task2>rnc.exe /d:nr /p1:312 /p2:17120
Числа сгенерированы и сохранены в файл distr-nr.dat!
D:\tgpsch\task2>

```

Рисунок 41 – Ввод

distr-nr.dat – Блокнот

Файл Правка Формат Вид Справка

-

7905.384539891429,14084.80915193371,2505.0637012458774,4760.5295452  
6554,-25117.29582041724,-157.06584432460437,-  
28080.403613388477,3466.870568785384,22562.610857031086,-  
23717.790998596633,8199.205862490624,24384.731459696643,-  
22993.474653098536,10665.383156430871,17557.072729812022,-  
8123.78822319641,-19383.34745458377,-11357.052691760016,-  
15913.966606106493,-1390.0559467377714,-  
2782.4328803492913,10164.408666137824,-10351.04045535783,-  
7022.136455989013,16670.78231948117,17548.790995623833,-  
35453.494040215155,9757.992043537719,3328.506813214186,-  
13695.941826439195,-  
6261.185933850859,7323.816452096755,5448.694260095258,-  
14440.523501847783,-  
212.59840427224333,4666.875741893707,7465.391886402999,-  
1719.1215180730312,40399.52722305423,-  
21164.765814596547,4370.363404128535,-  
39557.86482634559,1813.9125008658848,19817.15993855396,-  
4078.7610632842498,-  
3932.748237803169,10720.961350667025,8051.979129482765,3188.1368061  
7957,940.6121064180643,17593.850037494216,6992.652714472388,-  
15131.84319664822,19924.11251291152,-7932.93073347254,-  
16156.39884190038,-34388.837508659766,2448.0754106990653,-  
30157.723544097567,17862.09974262105,-  
8322.096722628788,31080.003405409403,11852.950776493672,-  
22048.744666514474,-15848.409747338195,-14377.411535715324,-  
16291.679416071329,1847.5294448749257,32952.08063894808,111.3285137  
1063771,9751.546473107042,-6011.054860197517,-  
726.6795745221589,1852.3596986348844,17777.4300664545,11099.9407950  
08899,-2428.4678118436086,16983.834556289068,25342.457917279367,-  
4198.574812582999,10202.3804896798,-8350.431974813304,-  
32046.452969466947,-27679.29730779085,-22119.15480915751,-  
59344.08447058393,33798.80967173592,5922.237075583696,-  
4902.602024233523,-6567.347550699615,-  
25350.779692907876,15516.6227741911,13569.346159481267,10425.495618  
995245,4718.550944255072,4955.059376567489,8427.464321048623,18732.

Стр 1, стлб 1 100% Windows (CRLF) UTF-8

Рисунок 42 – Вывод

```

"""нормальное распределение"""
def nr(mu, sigma, lst):
    res = []
    X_ = max(lst)
    m = X_ + 1
    lst = [x / m for x in lst]
    lst.append(lst[-1])
    for i in range(0, len(lst) - 1, 2):
        U1 = lst[i]
        U2 = lst[i + 1]
        Z1 = mu + sigma * math.sqrt(-2 * math.log(1 - U1)) * math.cos(2 * math.pi * U2)
        Z2 = mu + sigma * math.sqrt(-2 * math.log(1 - U1)) * math.sin(2 * math.pi * U2)
        res.append(Z1)
        res.append(Z2)
    return res

```

Рисунок 43 – Алгоритм

## 2.5 Гамма распределение (алгоритм для $c = k$ ( $k$ – целое число))

Используя независимые равномерные случайные числа  $U_1, U_2, \dots, U_k$ , применяют формулу

$$Y = a - b * \ln\{(1 - U_1)(1 - U_2) \dots (1 - U_k)\}.$$

```

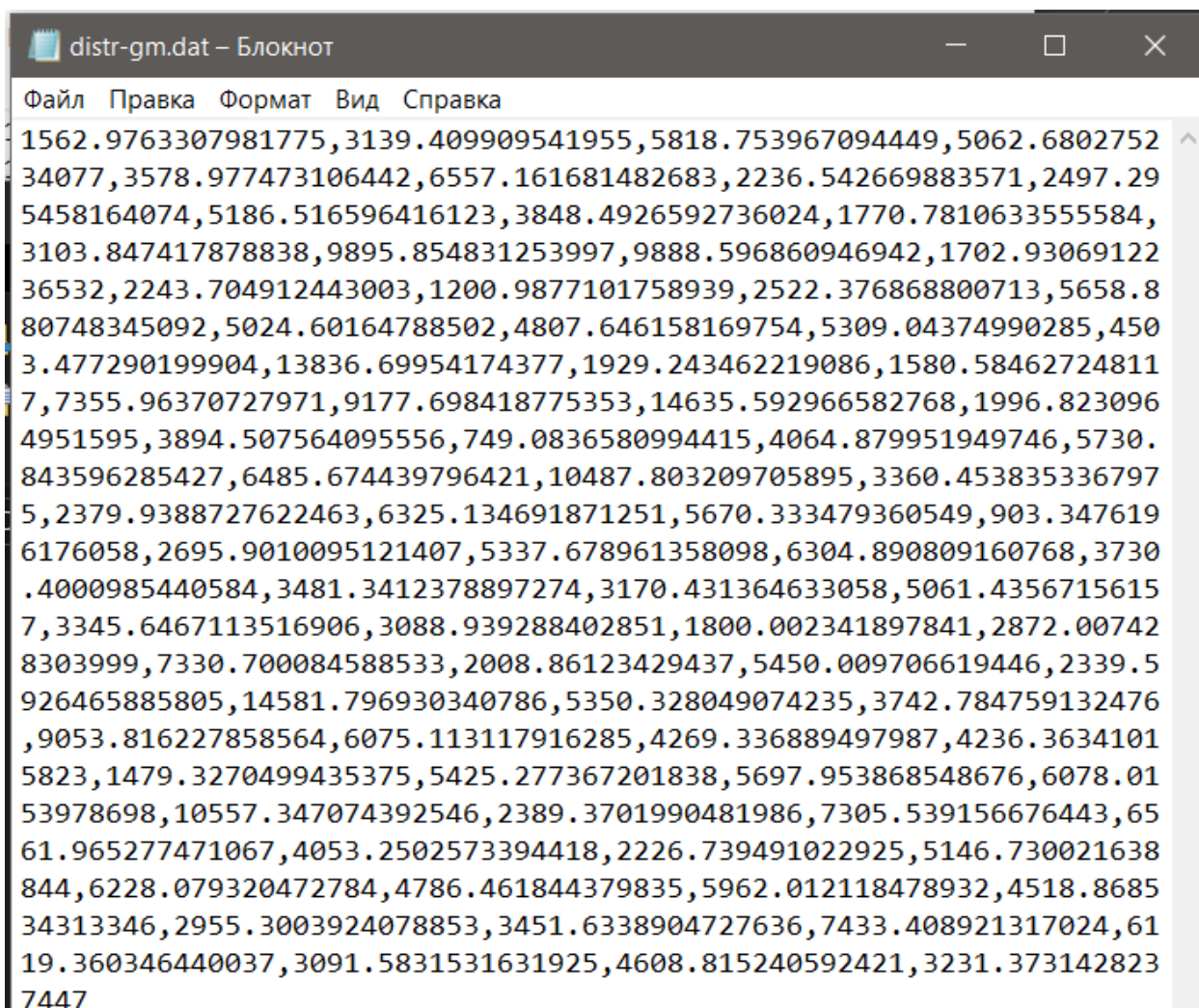
D:\tgpsch\task2>rnc.exe /d:gm /p1:297 /p2:1423 /p3:3
Числа сгенерированы и сохранены в файл distr-gm.dat!

D:\tgpsch\task2>

```

Рисунок 44 – Ввод





```
distr-gm.dat - Блокнот
Файл  Правка  Формат  Вид  Справка
1562.9763307981775,3139.409909541955,5818.753967094449,5062.6802752
34077,3578.977473106442,6557.161681482683,2236.542669883571,2497.29
5458164074,5186.516596416123,3848.4926592736024,1770.7810633555584,
3103.847417878838,9895.854831253997,9888.596860946942,1702.93069122
36532,2243.704912443003,1200.9877101758939,2522.376868800713,5658.8
80748345092,5024.60164788502,4807.646158169754,5309.04374990285,450
3.477290199904,13836.69954174377,1929.243462219086,1580.58462724811
7,7355.96370727971,9177.698418775353,14635.592966582768,1996.823096
4951595,3894.507564095556,749.0836580994415,4064.879951949746,5730.
843596285427,6485.674439796421,10487.803209705895,3360.453835336797
5,2379.9388727622463,6325.134691871251,5670.333479360549,903.347619
6176058,2695.9010095121407,5337.678961358098,6304.890809160768,3730
.4000985440584,3481.3412378897274,3170.431364633058,5061.4356715615
7,3345.6467113516906,3088.939288402851,1800.002341897841,2872.00742
8303999,7330.700084588533,2008.86123429437,5450.009706619446,2339.5
926465885805,14581.796930340786,5350.328049074235,3742.784759132476
,9053.816227858564,6075.113117916285,4269.336889497987,4236.3634101
5823,1479.3270499435375,5425.277367201838,5697.953868548676,6078.01
53978698,10557.347074392546,2389.3701990481986,7305.539156676443,65
61.965277471067,4053.2502573394418,2226.739491022925,5146.730021638
844,6228.079320472784,4786.461844379835,5962.012118478932,4518.8685
34313346,2955.3003924078853,3451.6338904727636,7433.408921317024,61
19.360346440037,3091.5831531631925,4608.815240592421,3231.373142823
7447
```

Рисунок 45 – Вывод

```

"""гамма-распределение"""
def gm(a, b, k, lst):
    res = []
    X_ = max(lst)
    m = X_ + 1
    lst = [x / m for x in lst]
    uk = []
    for i in range(0, len(lst), k):
        uk.append(lst[i : i + k])
        if len(uk[-1]) != k: uk.pop()
        for luk in uk:
            for_log = 1
            for u in luk:
                for_log = for_log * (1 - u)
            Y = a - b * math.log(for_log)
            res.append(Y)
    return res

```

Рисунок 46 – Алгоритм

## 2.6 Логнормальное распределение

Используя стандартные нормальные случайные числа  $Z$ , применяют формулу  $Y = a + \exp(b - Z)$  для получения случайных чисел, соответствующих логнормальному распределению.

```

D:\tgpsch\task2>rnc.exe /d:ln /p1:297 /p2:142
Числа сгенерированы и сохранены в файл distr-ln.dat!
D:\tgpsch\task2>

```

Рисунок 47 – Ввод

distr-In.dat – Блокнот

Файл Правка Формат Вид Справка

```
7.555658419376877e+61,2.0913772316730356e+61,4.11323495301411e
+61,3.605517502920804e+61,2.064896901734665e+62,4.805245017045806e
+61,2.4550792084755644e+62,3.888523429322417e+61,1.2745894334230931e
+61,1.9028133883274988e+62,2.949418553299976e+61,1.1459016972031897e
+61,1.8239880204946774e+62,2.5537315394412378e+61,1.707454520992379e
+61,7.652665102411374e+61,1.4772073921074293e+62,9.243428553865693e
+61,1.2062334551305514e+62,5.164087841142196e+61,5.60163719630189e
+61,2.629564299714845e+61,8.71591334921998e+61,7.175734662504128e
+61,1.7981762861379913e+61,1.708280695366427e+61,3.7766228888968264e
+62,2.692735134479015e+61,3.920077795861789e+61,1.0596568565148367e
+62,6.863771718739396e+61,3.104152121598061e+61,3.463462373803759e
+61,1.10676025199301e+62,4.8208572141984685e+61,3.6252952900338227e
+61,3.078587895115465e+61,5.264307473579524e+61,4.496652588681103e
+60,1.6391998375947425e+62,3.688631013656395e+61,4.799786697466789e
+62,4.2826877584143436e+61,1.4962907877389344e+61,6.04226544705133e
+61,5.990951389236744e+61,2.545454573689456e+61,2.9748920204658524e
+61,3.9523513573369866e+61,4.5068178613181403e+61,1.703790491811051e
+61,3.164782350541161e+61,1.1523667592766627e+62,1.4869722531448924e
+61,7.567825309579508e+61,1.2234361535129784e+62,3.5489163847054e
+62,4.1269497195053895e+61,2.771802431472925e+62,1.6773022159278585e
+61,7.74182468285093e+61,7.750039652167725e+60,2.3825905706627872e
+61,1.7260619730371324e+62,1.2016233110572015e+62,1.1026877505446846e
+62,1.2331419234764936e+62,4.2742864957986683e+61,6.94726220107301e
+60,4.73049946792048e+61,2.693749122891549e+61,6.764217891748945e
+61,4.9678137960286517e+61,4.273080714480654e+61,1.6856181270270277e
+61,2.4897258367279818e+61,5.487009458040515e+61,1.7655940666582857e
+61,1.0835577819778497e+61,6.08470035681228e+61,2.623738440985135e
+61,7.754648759449178e+61,3.0951017658532292e+62,2.398227488978897e
+62,1.7331754384653688e+62,1.5245869404616931e+63,6.61201985297935e
+60,3.3689750667231687e+61,6.340138204930526e+61,6.987622543686247e
+61,2.0932510324071543e+62,1.9235815485232852e+61,2.1553036772212754e
+61,2.5897666129811366e+61,3.614369155652351e+61,3.564780881198147e
+61,2.9103553771554115e+61,1.594174149203774e+61,3.0105820896838396e
+62,5.684087191249919e+61,1.8194115944729029e+61,1.5268506400476514e
+62,2.545853084681358e+62,1.0792789407255918e+61,1.0756008806418232e
+62,1.134484762901336e+61,7.152823231280898e+62,3.648199518279025e
```

Стр 1, стлб 1 100% Windows (CRLF) UTF-8

Рисунок 48 – Вывод



```

"""логнормальное распределение"""
def ln(a, b, lst):
    res = []
    lst = nr(0, 1, lst)
    for Z in lst:
        Y = a + math.exp(b * -Z)
        res.append(Y)
    return res

```

Рисунок 49 – Алгоритм

## 2.7 Логистическое распределение

Случайные числа, соответствующие логистическому распределению, получают по формуле

$$Y = a + b \ln\left(\frac{U}{1 - U}\right).$$

```

D:\tgpsch\task2>rnc.exe /d:ls /p1:297 /p2:142
Числа сгенерированы и сохранены в файл distr-ls.dat!
D:\tgpsch\task2>

```

Рисунок 50 – Ввод

distr-ls.dat – Блокнот

Файл Правка Формат Вид Справка

```
212.3224605662528,200.0295233842286,-  
150.28337938013743,78.87456272358861,396.47025481102827,298.6673381  
384891,454.1166761903372,286.9919453901835,531.9727493762545,565.54  
07359118587,394.5940526772717,99.79858732828862,397.72569892320064,  
258.9819555310439,278.0757588639638,659.1410960852209,349.269172688  
2283,345.8277364180517,218.35268981744105,306.4516288456041,68.1260  
912513743,175.62350291310636,139.88332380966276,352.1486358143388,3  
65.54945701328944,26.004425300146295,614.0163325802932,273.60475690  
320015,173.62696436179715,479.85315086818616,45.94552651503167,221.  
34175667116324,203.13407637842772,496.8249845978561,-  
185.8265213621849,155.10023957979308,-  
31.502233291953075,734.0510310426898,793.7906530384256,647.16896686  
61523,676.5806502855464,465.5070184564195,285.32219005034995,131.57  
71564407116,-  
89.63140148585057,367.90928953029135,140.62629761834611,-  
12.248521790034715,-300.348658209522,-  
177.18372086805766,264.62566017265186,-  
97.00312415899208,387.78287245368034,212.9287489562221,262.37147828  
59071,401.51504567146617,570.4822988616686,291.44156042507274,578.1  
441673829971,249.31621244697092,517.0405236068763,172.2895416783992  
2,391.4850833514452,518.0043900911498,329.3743398273482,364.9609589  
629303,226.0907104697086,288.6610093861849,529.9201274654981,1280.8  
523701793865,98.05279671429264,618.8466981313247,-  
431.72542056527277,205.6021917762581,304.22660823608726,-  
34.92828649845637,230.80112548182234,160.00647763306551,396.4702548  
1102827,798.7477464363353,145.04396102346308,587.4980632641082,734.  
0510310426898,362.61263015391546,1280.8523701793865,412.45325730553  
62,554.8635179743883,-  
215.18063370390541,13.23093196007028,382.8923751756704,477.40568147  
44572,248.1722635819483,226.0907104697086,-9.233972419813028,-  
75.82219266608587,26.004425300146295,296.44422629639865,85.44961558  
691074,518.972539059235,306.4516288456041,405.3403524371717,551.429  
8446303837,641.4884317404565,231.3873698460846,446.04794669807416,1  
99.40594732272828,1124.5709937267538,356.19892015831374,404.7002236  
1341506,310.34837258150515,130.0394932784376,189.93886219720952,304  
.7827695822893,176.9482590948329,264.0625062985162,595.911802824562
```

Стр 1, слб 1      100%      Windows (CRLF)      UTF-8

Рисунок 51 – Вывод

```

"""логистическое распределение"""
def ls(a, b, lst):
    res = []
    X_ = max(lst)
    m = X_ + 1
    for x in lst:
        U = x / m
        Y = a + b * math.log(U / (1 - U))
        res.append(Y)
    return res

```

Рисунок 52 – Алгоритм

## 2.8 Биномиальное распределение

Если вероятность появления события при каждом испытании равна  $p$ , то вероятность того, что это событие произойдет  $y$  раз за  $n$  испытаний, определяют по формуле

$$p(y) = \binom{n}{y} p^y (1 - p)^{n-y}, \quad y = 0, 1, \dots, n,$$

где  $0 < p < 1$ .

```

D:\tgpsch\task2>rnc.exe /d:bi /p1:297 /p2:7
Числа сгенерированы и сохранены в файл distr-bi.dat!
D:\tgpsch\task2>

```

Рисунок 53 – Ввод (был сокращен список чисел файла lst.dat)

```

distr-bi.dat - Блокнот
Файл  Правка  Формат  Вид  Справка
8159616,7932800,9632768,8882176,4348160,3478784,2729216,7690880,858
4064,535680,7996544,2237824,1915904,2289664,9376768,4438912,5779200
,5967104,3107200,7116288,4819072,8601344,7639040,6807424,6608640,22
64832,9961088,8774144,5472512,4586880,2585600,4135424,2183808,80364
80

```

Рисунок 54 – Вывод

```

4  """биномиальное распределение"""
5  def bi(p, p2, lst):
6      res = []
7      X_ = max(lst)
8      m = X_ + 1
9      lst = [x / m for x in lst]
10     n = len(lst)
11     for i in range(0, n):
12         sum = 0
13         y = lst[i]
14         while sum < lst[i]:
15             k = 0
16             while y > k:
17                 sum += math.comb(n, k) * pow(p, k) * pow(1 - p, n - k)
18                 k = k + 1
19             y = y + 1
20             t = 1
21             for _ in range(p2):
22                 t *= 10
23             res.append(int(y * 10**23) % t)
24     return res

```

Рисунок 55 – Алгоритм

## ПРИЛОЖЕНИЕ А

### Листинг Задание 1 Генератор псевдослучайных чисел

```
import sys
"""линейный конгруэнтный метод"""
def lc (m, a, c, x0, n_ = 10000):
    x = []
    x.append(x0);
    for i in range(1, n_ - 1):
        x.append(((a * x[i - 1] + c) % m) % 2**10)
    return x

"""аддитивный метод"""
def add(m, low_i, up_i, start_seq, n_ = 10000):
    x = start_seq
    seq = start_seq.copy()
    n = len(seq)
    l = n_ - n
    for _ in range(l):
        xn = (seq[n - low_i] + seq[n - up_i]) % m
        x.append(xn % 2**10)
        seq.append(xn)
        del seq[0]
    return x

"""5p"""
def _5p(p, q1, q2, q3, w, x0, n_ = 10000):
    x = []
    x0 = int(x0, 2)
    for _ in range(n_):
        cur = 0
        for _ in range(w):
            bit_q1 = (x0 >> p - q1) & 1
            bit_q2 = (x0 >> p - q2) & 1
            bit_q3 = (x0 >> p - q3) & 1
            bit_x0 = x0 & 1
            xor = bit_q1 ^ bit_q2 ^ bit_q3 ^ bit_x0
            cur = (cur << 1) | xor
            x0 = (x0 >> 1) | (xor << p - 1)
        x.append(cur % 2**10)
    return x

"""рслос"""
def lfsr(vec_bin, start_reg, n_ = 10000):
    x = []
    reg_len = len(start_reg)
    vec_bin = int(vec_bin, 2)
    start_reg = int(start_reg, 2)
    shift = reg_len - 1
    for _ in range(n_):
        new = (start_reg ^ vec_bin >> shift) & 1
        start_reg = (start_reg >> 1) | (new << (shift))
        x.append(start_reg % 2**10)
    return x

"""нелинейная комбинация рслос"""
def nlsr(R1, R2, R3, w, x1, x2, x3, n_ = 10000):
    x = []
    lR1 = len(R1)
    lR2 = len(R2)
    lR3 = len(R3)
    for _ in range(n_):
```

```

cur = 0
for _ in range(w):
    xorR1 = (x1 ^ (x1 >> lR1 - 1))
    xorR2 = (x2 ^ (x2 >> lR1 - 1))
    xorR3 = (x3 ^ (x3 >> lR1 - 1))
    res = ((xorR1 ^ xorR2) + (xorR2 ^ xorR3) + xorR3) & 1
    x1 = (x1 >> 1) | (res << lR1 - 1)
    x2 = (x2 >> 1) | (res << lR2 - 1)
    x3 = (x3 >> 1) | (res << lR3 - 1)
    cur = (cur << 1) | res
x.append(cur % 2**10)
return x

"""вихрь мерсенна"""
def mt(x0, n = 624, n_ = 10000):
    w = 32
    r = 31
    m = 397
    a = 0x9908B0DF
    u = 11
    d = 0xFFFFFFFF
    s = 7
    t = 15
    l = 18
    b = 0x9D2C5680
    c = 0xEFC60000
    f = 1812433253
    lst = [0] * n
    ind = n + 1

    def f1(core):
        lst[0] = core
        for i in range(1, n):
            tmp = f * (lst[i - 1] ^ (lst[i - 1] >> (w - 2))) + i
            lst[i] = tmp & d

    def f2():
        nonlocal ind
        if ind >= n:
            f3()
            ind = 0
        y = lst[ind]
        y = y ^ ((y >> u) & d)
        y = y ^ ((y << s) & b)
        y = y ^ ((y << t) & c)
        y = y ^ (y >> l)
        ind = ind + 1
        return y & d

    def f3():
        for i in range(n):
            x = (lst[i] >> r) + (lst[(i + 1) % n] & ((1 << r) - 1))
            x_a = x >> 1
            lst[i] = lst[(i + m) % n] ^ x_a
            if (x % 2) != 0:
                lst[i] ^= a

    f1(x0)
    res = []
    for _ in range(n_):
        o = f2()
        res.append(o % 2**10)
    return res

```



```

"""rc4"""
def rc4(k, count_num = 10000):
    len_k = len(k)
    S = [i for i in range(256)]
    j = 0
    for i in range(256):
        j = (j + S[i] + k[i % len_k]) % 256
        S[i], S[j] = S[j], S[i]
    i = 0
    j = 0
    Ks = []
    for _ in range(1, count_num + 1):
        i = (i + 1) % 256
        j = (j + S[i]) % 256
        S[i], S[j] = S[j], S[i]
        cur = S[(S[i] + S[j]) % 256]
        Ks.append(cur % 2**10)
    return Ks

"""ПСЧ на основе rsa"""
def rsa(n, e, w, x0, n_ = 10000):
    x = []
    for _ in range(n_):
        cur = 0
        for _ in range(w):
            x0 = pow(x0, e, n)
            cur = (cur << 1) | (x0 & 1)
        x.append(cur % 2**10)
    return x

"""алгоритм блюма-блума-шуба"""
def bbs(x0, l, n_ = 10000):
    p = 127
    q = 131
    n = p * q
    x = []
    for _ in range(n_):
        cur = 0
        for _ in range(l):
            x0 = pow(x0, 2, n)
            cur = (cur << 1) | (x0 & 1)
        x.append(cur % 2**10)
    return x

def generate_pseudo_random(method, args, n):
    if method == 'lc':
        int_values = [int(x) for x in args.split(',')]
        if len(int_values) != 4:
            print('Было передано неверное число аргументов!')
            return []
        m, a, c, x0 = int_values
        return lc(m, a, c, x0, n)

    elif method == 'add':
        args_split = args.split(',')
        if len(args_split) < 4:
            print('Было передано неверное число аргументов!')
            return []
        m = int(args_split[0])
        low_i = int(args_split[1])
        up_i = int(args_split[2])
        start_seq = list(map(int, args_split[3:]))

```

```

        return add(m, low_i, up_i, start_seq, n)

elif method == '5p':
    args_split = args.split(',')
    if len(args_split) != 6:
        print('Было передано неверное число аргументов!')
        return []
    p = int(args_split[0])
    q1 = int(args_split[1])
    q2 = int(args_split[2])
    q3 = int(args_split[3])
    w = int(args_split[4])
    x0 = args_split[5]
    return _5p(p, q1, q2, q3, w, x0, n)

elif method == 'lfsr':
    args_split = args.split(',')
    if len(args_split) != 2:
        print('Было передано неверное число аргументов!')
        return []
    vec_bin = args_split[0]
    start_reg = args_split[1]
    return lfsr(vec_bin, start_reg, n)

elif method == 'nfsr':
    args_split = args.split(',')
    if len(args_split) != 7:
        print('Было передано неверное число аргументов!')
        return []
    R1 = args_split[0]
    R2 = args_split[1]
    R3 = args_split[2]
    w = int(args_split[3])
    x1 = int(args_split[4])
    x2 = int(args_split[5])
    x3 = int(args_split[6])
    return nfsr(R1, R2, R3, w, x1, x2, x3, n)

elif method == 'mt':
    args_split = args.split(',')
    if len(args_split) != 1:
        print('Было передано неверное число аргументов!')
        return []
    x0 = int(args_split[0])
    return mt(x0, 624, n)

elif method == 'rc4':
    args_split = args.split(',')
    if len(args_split) != 256:
        print('Было передано неверное число аргументов!')
        return []
    k = list(map(int, args_split))
    return rc4(k, n)

elif method == 'rsa':
    args_split = args.split(',')
    if len(args_split) != 4:
        print('Было передано неверное число аргументов!')
        return []
    m, e, w, x0 = map(int, args_split)
    return rsa(m, e, w, x0, n)

elif method == 'bbs':

```

```

args_split = args.split(',')
if len(args_split) != 2:
    print('Было передано неверное число аргументов!')
    return []
x0 = int(args_split[0])
l = int(args_split[1])
return bbs(x0, l, n)

else:
    print("Введенный метод не предусмотрен или не существует!")
    return []

def print_usage():
    print('Применение: prng.exe /g:<generator> [/n:<count>] [/f:<filename>] [/i:<params>]')
    print('          prng.exe /h for help')

def print_help():
    print_usage()
    print('Генерация последовательности псевдослучайных чисел по выбранному методу')
    print()
    print('Возможные аргументы:')
    print('  /g: <generator>  параметр указывает на метод генерации ПСЧ:')
    print('                    lc  - линейный конгруэнтный метод;')
    print('                    add - аддитивный метод;')
    print('                    5p  - пятипараметрический метод;')
    print('                    lfsr - регистр сдвига с обратной связью (РСЛОС);')
    print('                    nfsr - нелинейная комбинация РСЛОС;')
    print('                    mt  - вихрь Мерсенна;')
    print('                    rc4 - RC4;')
    print('                    rsa - ГПСЧ на основе RSA;')
    print('                    bbs - алгоритм Блюма-Блюма-Шуба;')
    print('  /n: <count>      количество генерируемых чисел (по умолчанию 10000)')
    print('  /f: <filename>    полное имя файла, в который будут выводиться данные (по умолчанию rnd.dat)')
    print('  /i: <params>      перечисление параметров для выбранного генератора')

def main():
    args = sys.argv[1:]
    method = None
    n = 10000
    filename = 'rnd.dat'

    for arg in args:
        if arg.startswith('/g:'):
            method = arg[3:]
        elif arg.startswith('/n:'):
            n = int(arg[3:])
        elif arg.startswith('/f:'):
            filename = arg[3:]
        elif arg.startswith("/i:"):
            input_args = arg[3:]
        elif arg == '/h':
            if arg != args[0]:
                print('Ошибка: /h не может быть использован одновременно с другими аргументами!')
            print_usage()
            return

```

```

        else:
            print_help()
            return

if method is None:
    print('Метод не указан!')
    print_usage()
    return

elif input_args is None:
    print('Неуказаны аргументы для функций!')
    print_usage()
    return

random_numbers = generate_pseudo_random(method, input_args, n)

if random_numbers == []:
    return
else:
    with open(filename, 'w', encoding = 'UTF-8') as f:
        f.write(','.join(map(str, random_numbers)))
    print(f'Числа сгенерированы и сохранены в файл {filename}!')

if __name__ == '__main__':
    main()

```

## ПРИЛОЖЕНИЕ Б

### Листинг Задание 2 Преобразование ПСЧ к заданному распределению

```
import sys
import math

"""стандартное равномерное распределение"""
def st(a, b, lst):
    res = []
    X_ = max(lst)
    m = X_ + 1
    for x in lst:
        U = x / m
        Y = b * U + a
        res.append(Y)
    return res

"""треугольное распределение"""
def tr(a, b, lst):
    res = []
    X_ = max(lst)
    m = X_ + 1
    lst = [x / m for x in lst]
    lst.append(lst[-1])
    for i in range(len(lst) - 1):
        U1 = lst[i]
        U2 = lst[i + 1]
        Y = a + b * (U1 + U2 - 1)
        res.append(Y)
    return res

"""общее экспоненциальное распределение"""
def ex(a, b, lst):
    res = []
    X_ = max(lst)
    m = X_ + 1
    for x in lst:
        U = x / m
        Y = -b * math.log(U) + a
        res.append(Y)
    return res

"""нормальное распределение"""
def nr(mu, sigma, lst):
    res = []
    X_ = max(lst)
    m = X_ + 1
    lst = [x / m for x in lst]
    lst.append(lst[-1])
    for i in range(0, len(lst) - 1, 2):
        U1 = lst[i]
        U2 = lst[i + 1]
        Z1 = mu + sigma * math.sqrt(-2 * math.log(1 - U1)) * math.cos(2 *
math.pi * U2)
        Z2 = mu + sigma * math.sqrt(-2 * math.log(1 - U1)) * math.sin(2 *
math.pi * U2)
        res.append(Z1)
        res.append(Z2)
    return res

"""гамма распределение"""
def gm(a, b, k, lst):
```

```

res = []
X_ = max(lst)
m = X_ + 1
lst = [x / m for x in lst]
uk = []
for i in range(0, len(lst), k):
    uk.append(lst[i : i + k])
if len(uk[-1]) != k: uk.pop()
for luk in uk:
    for_log = 1
    for u in luk:
        for_log = for_log * (1 - u)
    Y = a - b * math.log(for_log)
    res.append(Y)
return res

"""логнормальное распределение"""
def ln(a, b, lst):
    res = []
    lst = nr(0, 1, lst)
    for Z in lst:
        Y = a + math.exp(b - Z)
        res.append(Y)
    return res

"""логистическое распределение"""
def ls(a, b, lst):
    res = []
    X_ = max(lst)
    m = X_ + 1
    for x in lst:
        U = x / m
        Y = a + b * math.log(U / (1 - U))
        res.append(Y)
    return res

"""биномиальное распределение"""
def bi(p, p2, lst):
    res = []
    X_ = max(lst)
    m = X_ + 1
    lst = [x / m for x in lst]
    n = len(lst)
    for i in range(0, n):
        sum = 0
        y = lst[i]
        while sum < lst[i]:
            k = 0
            while y > k:
                sum += math.comb(n, k) + pow(p, k) + pow(1 - p, n - k)
                k = k + 1
            y = y + 1
        t = 1
        for _ in range(p2):
            t *= 10
        res.append(int(y * 10**23) % t)
    return res

def print_usage():
    print("Применение: rnc.exe [/f:<filename>] /d:<distribution> /p1:<param1> /p2:<param2> [/p3:<param3>]")
    print('          rnc.exe /h for help')

```



```

def print_help():
    print_usage()
    print('Преобразование ПСЧ к заданному распределению')
    print()
    print('Возможные аргументы:')
    print(' /f: <filename>    полное имя файла, из которого будет браться
начальная последовательность (по умолчанию lst.dat)')
    print(' /d: <distribution>  код распределения для преобразования
последовательности:')
    print('
                                st - стандартное равномерное с заданным
интервалом;')
    print('
                                tr - треугольное распределение;')
    print('
                                ex - общее экспоненциальное распределение;')
    print('
                                nr - нормальное распределение;')
    print('
                                gm - гамма распределение;')
    print('
                                ln - логнормальное распределение;')
    print('
                                ls - логистическое распределение;')
    print('
                                rsa - ГПСЧ на основе RSA;')
    print('
                                bi - биномиальное распределение.')
    print(' /p1:<parameter1>    1-й параметр, необходимый, для генерации
ПСЧ заданного распределения')
    print(' /p2:<parameter2>    2-й параметр, необходимый, для генерации
ПСЧ заданного распределения')
    print(' /p3:<parameter3>    3-й параметр, необходимый, для генерации
ПСЧ заданного распределения')

def main():
    args = sys.argv[1:]
    filename = 'lst.dat'
    distribution = None
    params = {}

    for arg in args:
        if arg.startswith("/f:"):
            filename = arg[3:]
        elif arg.startswith("/d:"):
            distribution = arg[3:]
        elif arg.startswith("/p"):
            param_name, param_value = arg[1:].split(":")
            params[param_name] = int(param_value)
        elif arg == '/h':
            if arg != args[0]:
                print('Ошибка: /h не может быть использован одновременно с
другими аргументами!')
                print_usage()
                return
            else:
                print_help()
                return

    if distribution is None:
        print('Метод не указан!')
        print_usage()
        return

    if params is None:
        print('Неуказаны аргументы для функций!')
        print_usage()
        return

    if filename is None or distribution is None:

```

```

        print("Применение: rnc.exe /f:<filename> /d:<distribution>
/p1:<param1> /p2:<param2> /p3:<param3>")
        return

    try:
        with open(filename, "r") as file:
            numbers = file.readline().strip().split(',')
            lst = [int(num) for num in numbers]
    except FileNotFoundError:
        print(f"Файл '{filename}' не найден!")
        return

    transformed_numbers = []

    if distribution == "st":
        transformed_numbers = st(params["p1"], params["p2"], lst)
    elif distribution == "tr":
        transformed_numbers = tr(params["p1"], params["p2"], lst)
    elif distribution == "ex":
        transformed_numbers = ex(params["p1"], params["p2"], lst)
    elif distribution == "nr":
        transformed_numbers = nr(params["p1"], params["p2"], lst)
    elif distribution == "gm":
        transformed_numbers = gm(params["p1"], params["p2"],
params["p3"], lst)
    elif distribution == "ln":
        transformed_numbers = ln(params["p1"], params["p2"], lst)
    elif distribution == "ls":
        transformed_numbers = ls(params["p1"], params["p2"], lst)
    elif distribution == "bi":
        transformed_numbers = bi(params["p1"], params["p2"], lst)
    else:
        print('Введенный метод не предусмотрен или не существует!')
        return

    output_filename = f"distr-{distribution}.dat"

    with open(output_filename, 'w', encoding = 'UTF-8') as f:
        f.write(','.join(map(str, transformed_numbers)))
        print(f'Числа сгенерированы и сохранены в файл {output_filename}!')

if __name__ == '__main__':
    main()

```