Министерство образования и науки Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

«САРАТОВСКИЙ 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

факультета компьютерных наук и информационных технологий

*Змеевой Вероники Александровны*

фамилия, имя, отчество

Научный руководитель

Ст. преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_ И.И. Слеповичев

подпись, дата

Саратов 2024

**СОДЕРЖАНИЕ**

[**Задание 1. Генерация псевдослучайных чисел.** 3](#_Toc165136953)

[**1.1** **Линейный конгруэнтный метод** 4](#_Toc165136954)

[**1.2 Аддитивный метод** 6](#_Toc165136955)

[**1.3 Пятипараметрический метод** 7](#_Toc165136956)

[**1.4 Регистр сдвига с обратной связью (РСЛОС)** 9](#_Toc165136957)

[**1.5 Нелинейная комбинация РСЛОС** 11](#_Toc165136958)

[**1.6 Вихрь Мерсенна** 12](#_Toc165136959)

[**1.7 RC4** 14](#_Toc165136960)

[**1.8 ГПСЧ на основе RSA** 16](#_Toc165136961)

[**1.9 Алгоритм Блюма-Блюма-Шуба** 18](#_Toc165136962)

[**Задание 2. Преобразование ПСЧ к заданному распределению.** 20](#_Toc165136963)

[**2.1 Стандартное равномерное с заданным интервалом** 21](#_Toc165136964)

[**2.2 Треугольное распределение** 22](#_Toc165136965)

[**2.3 Общее экспоненциальное распределение** 23](#_Toc165136966)

[**2.4 Нормальное распределение** 24](#_Toc165136967)

[**2.5 Гамма распределение (для параметра c=k)** 25](#_Toc165136968)

[**2.6 Логнормальное распределение** 26](#_Toc165136969)

[**2.7 Логистическое распределение** 27](#_Toc165136970)

[**2.8 Биномиальное распределение** 28](#_Toc165136971)

[**ПРИЛОЖЕНИЕ А** 30](#_Toc165136972)

[**ПРИЛОЖЕНИЕ Б** 40](#_Toc165136973)

# **Задание 1. Генерация псевдослучайных чисел.**

**Описание задания**: создать 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;

Для управления приложением предлагается следующий формат параметров командной строки:

-g <код\_метода> - параметр указывает на метод генерации ПСЧ, при этом код\_метода может быть одним из следующих:

· 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;

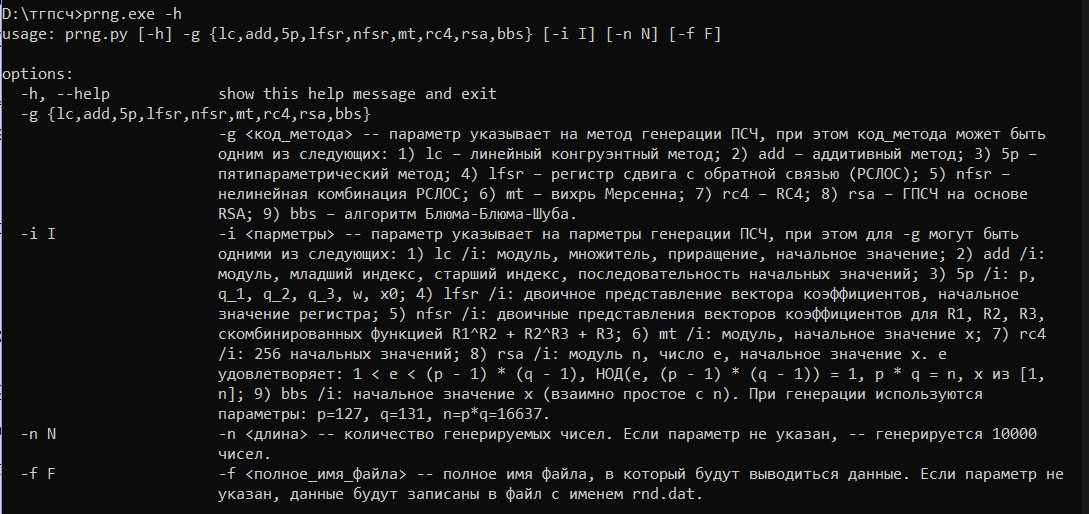
· bbs – алгоритм Блюма-Блюма-Шуба;

-i <число> - инициализационный вектор генератора.

-n <длина> - количество генерируемых чисел. Если параметр не указан, - генерируется 10000 чисел.

-f <полное\_имя\_файла> - полное имя файла, в который будут выводиться данные. Если параметр не указан, данные должны записываться в файл с именем rnd.dat.

-h – информация о допустимых параметрах командной строки программы.



**1.1** **Линейный конгруэнтный метод**

**Описание алгоритма:**

Одним из простых и популярных методов сейчас является *линейный конгруэнтный метод* (ЛКМ), предложенный Д.Г. Лехмером в 1949 году. В его основе лежит выбор четырех ключевых чисел:

* , модуль;
* , множитель;
* , приращение (инкремент);
* , начальное значение.

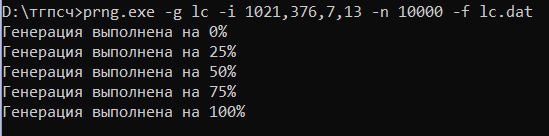
Последовательность ПСЧ, получаемая по формуле:

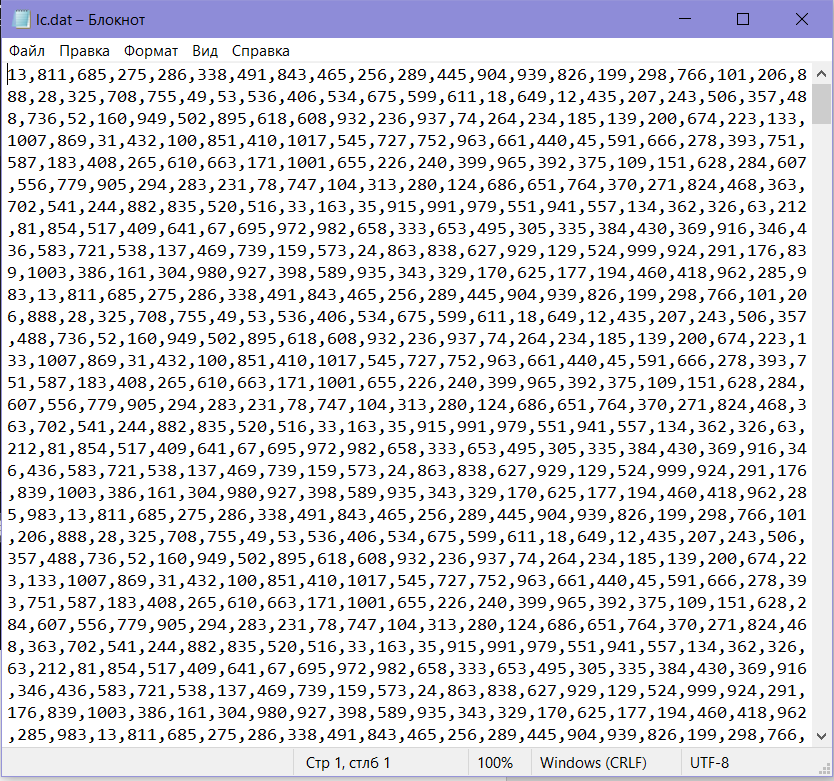
называется линейной конгруэнтной последовательностью (ЛКП). Ключом для неё служит .

**Пример ввода:**

prng.exe -g lc -i 1021,376,7,13 -n 10000 -f lc.dat

**Пример работы программы:**





**1.2 Аддитивный метод**

**Описание алгоритма**

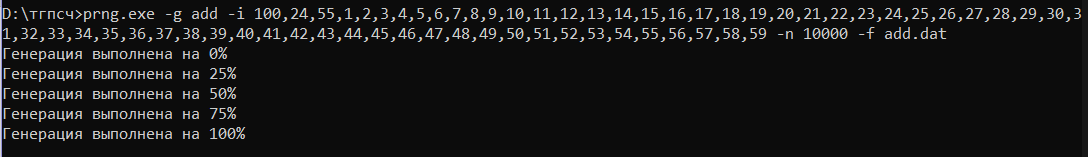
Формула:

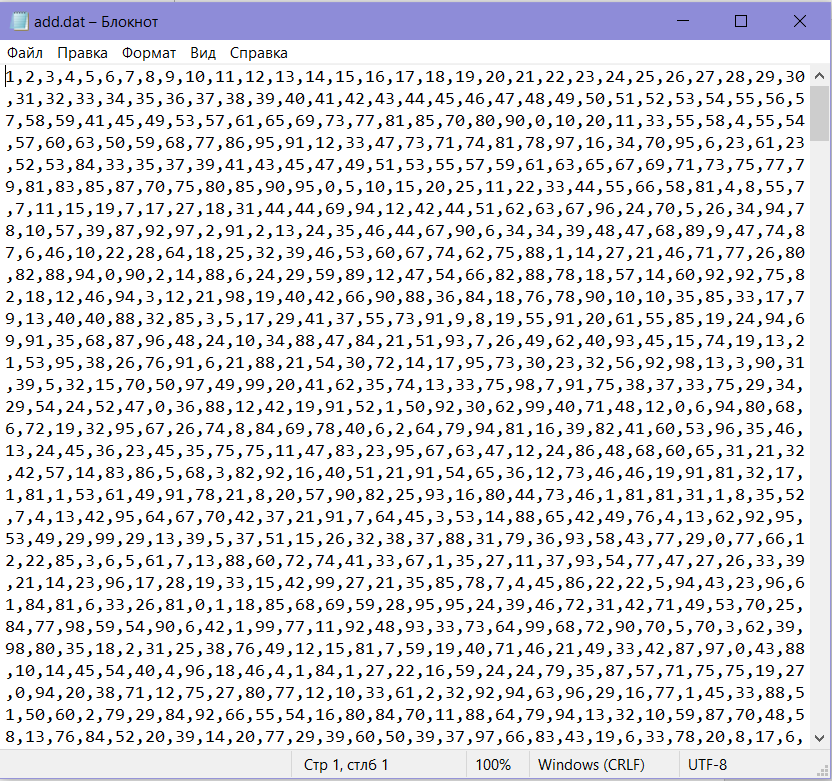
Параметры, поступающие на вход: модуль m, коэффициент k, коэффициент j, числа .

**Пример ввода:**

prng.exe -g add -i 100,24,55,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51,52,53,54,55,56,57,58,59 -n 10000 -f add.dat

**Пример работы программы:**





**1.3 Пятипараметрический метод**

**Описание алгоритма:**

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:

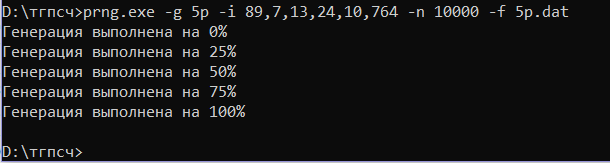
При этом

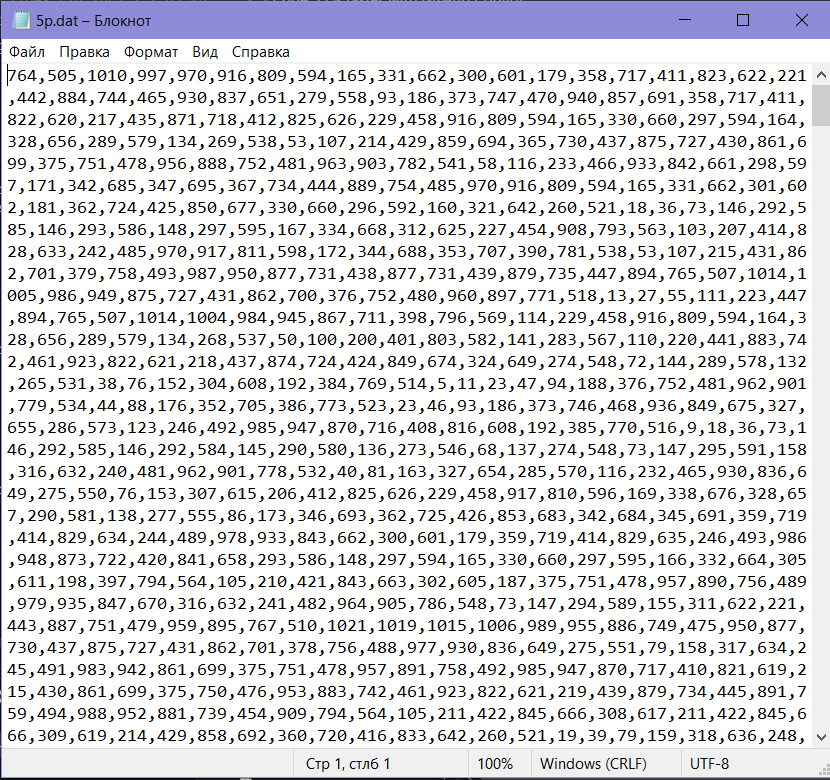
Параметры и , первоначально задают как начальный вектор.

**Пример ввода:**

prng.exe -g 5p -i 89,7,13,24,10,764 -n 10000 -f 5p.dat

**Пример работы программы:**





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

**Описание алгоритма:**

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 числа p и , принимающих значения 0 или 1, определяют рекуррентную формулу

(3.7)

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

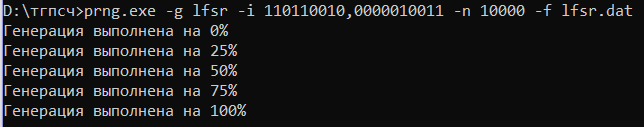
Одна итерация алгоритма, генерирующего последовательность, состоит из следующих шагов:

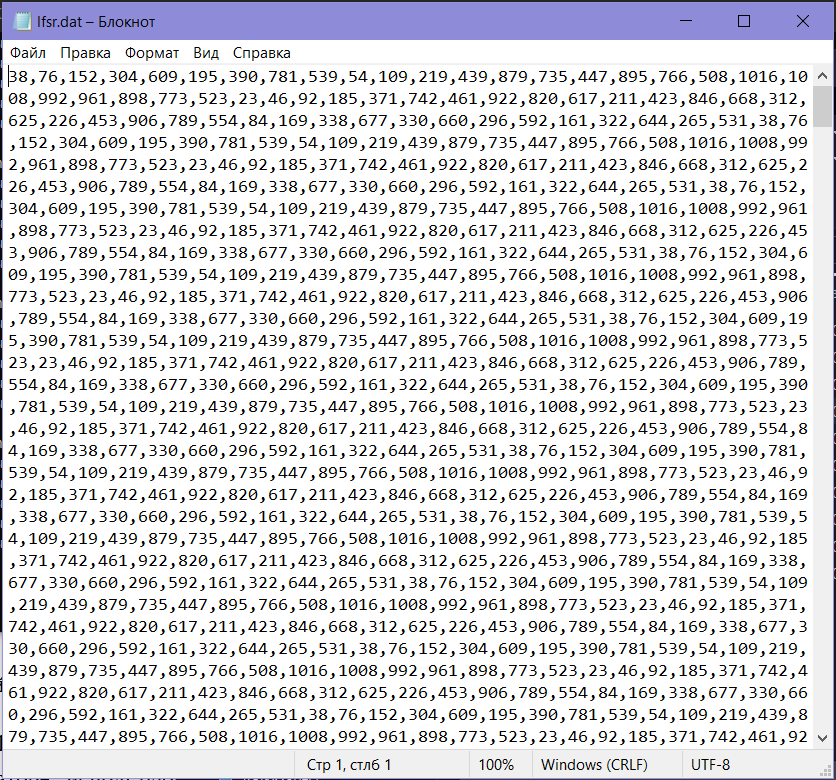
1. Содержимое ячейки формирует очередной бит ПСП битов.
2. Содержимое ячейки 0 определяется значением функции обратной связи, являющейся линейной булевой функцией с коэффициентами . Его вычисляют по формуле 3.7.
3. Содержимое каждого i-го бита перемещается в -й, .
4. В ячейку 0 записывается новое содержимое, вычисленное на шаге 2.

**Пример ввода:**

prng.exe -g lfsr -i 110110010,0000010011 -n 10000 -f lfsr.dat

**Пример работы программы:**





**1.5 Нелинейная комбинация РСЛОС**

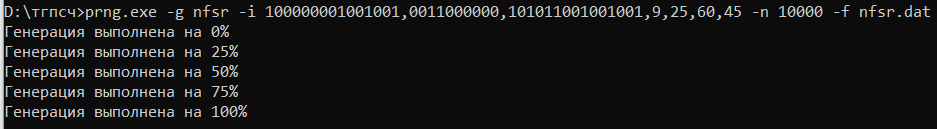
**Описание алгоритма:**

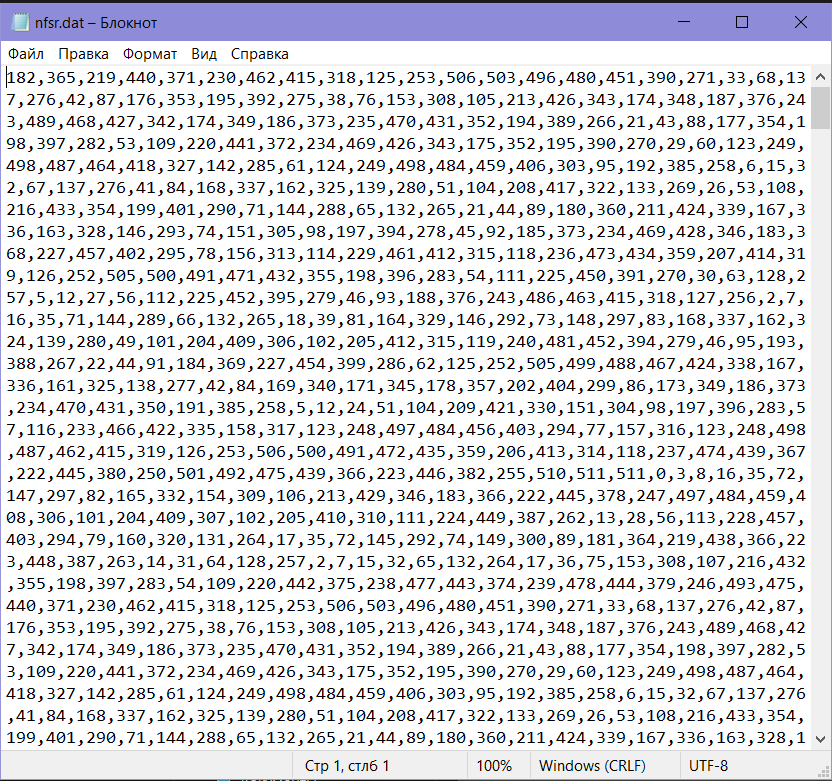
Генератор Геффа является примером нелинейной комбинации РСЛОС. В этом генераторе используются три РСЛОС, объединённые нелинейным образом. Длины этих регистров – попарно простые числа. Нелинейная функция генератора:

**Пример ввода:**

prng.exe -g nfsr -i 100000001001001,0011000000,101011001001001,9,25,60,45 -n 10000 -f nfsr.dat

**Пример работы программы:**





**1.6 Вихрь Мерсенна**

**Описание алгоритма:**

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

**Шаги алгоритма.**

**Шаг 1а**. Инициализируются значения по формуле:

– всего бит, – всего *r* бит,

*–* последняя строка матрицы *А.*

**Шаг 1б**. заполняются начальными значениями.

**Шаг 2**. Вычисляется

**Шаг 3**. Вычисляется новое значение :

**Шаг 4**. Вычисляется

*Z* подается на выход, как результат.

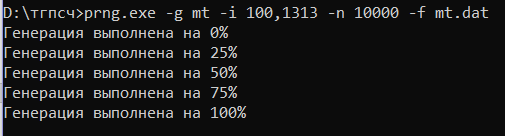
**Шаг 5**. . Переход на шаг 2.

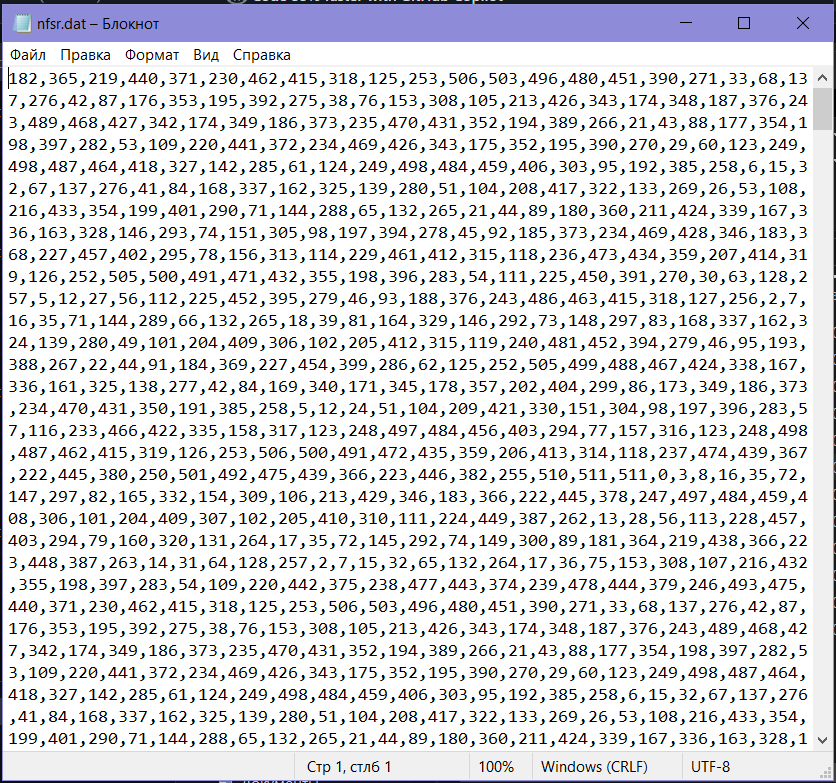
**Параметры алгоритма Вихрь Мерсенна**: , s=7, t=15,

**Пример ввода:**

prng.exe -g mt -i 100,1313 -n 10000 -f mt.dat

**Пример работы программы:**





**1.7 RC4**

Являясь потоковым шифром, в основе которого генератор псевдослучайных чисел, RC4 широко используется в различных криптографических протоколах. Достоинством алгоритма является высокая скорость работы и переменный размер ключа.

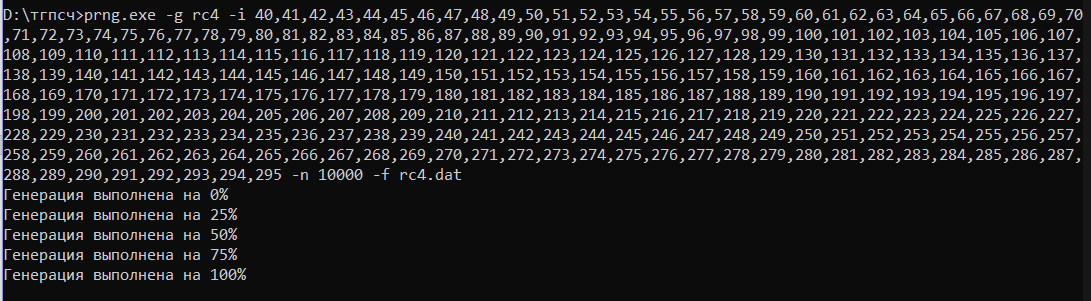
**Описание алгоритма.**

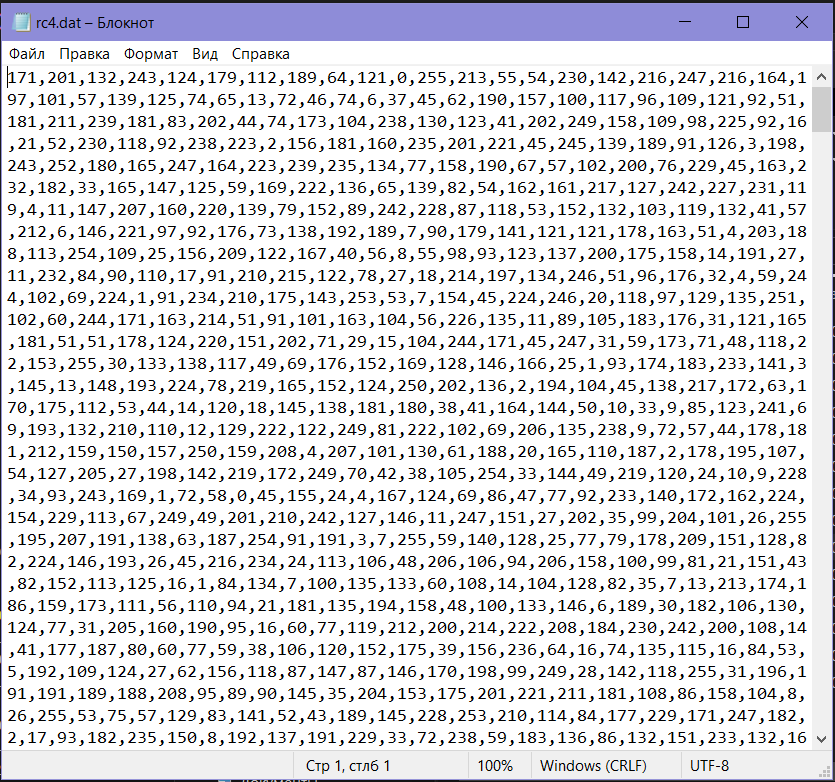
1. Инициализация ,
2. Итерация алгоритма:

**Пример ввода:**

prng.exe -g rc4 -i 40,41,42,43,44,45,46,47,48,49,50,51,52,53,54,55,56,57,58,59,60,61,62,63,64,65,66,67,68,69,70,71,72,73,74,75,76,77,78,79,80,81,82,83,84,85,86,87,88,89,90,91,92,93,94,95,96,97,98,99,100,101,102,103,104,105,106,107,108,109,110,111,112,113,114,115,116,117,118,119,120,121,122,123,124,125,126,127,128,129,130,131,132,133,134,135,136,137,138,139,140,141,142,143,144,145,146,147,148,149,150,151,152,153,154,155,156,157,158,159,160,161,162,163,164,165,166,167,168,169,170,171,172,173,174,175,176,177,178,179,180,181,182,183,184,185,186,187,188,189,190,191,192,193,194,195,196,197,198,199,200,201,202,203,204,205,206,207,208,209,210,211,212,213,214,215,216,217,218,219,220,221,222,223,224,225,226,227,228,229,230,231,232,233,234,235,236,237,238,239,240,241,242,243,244,245,246,247,248,249,250,251,252,253,254,255,256,257,258,259,260,261,262,263,264,265,266,267,268,269,270,271,272,273,274,275,276,277,278,279,280,281,282,283,284,285,286,287,288,289,290,291,292,293,294,295 -n 10000 -f rc4.dat

**Пример работы программы:**





**1.8 ГПСЧ на основе RSA**

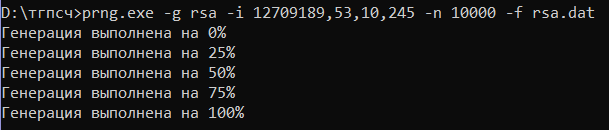
**Описание алгоритма.**

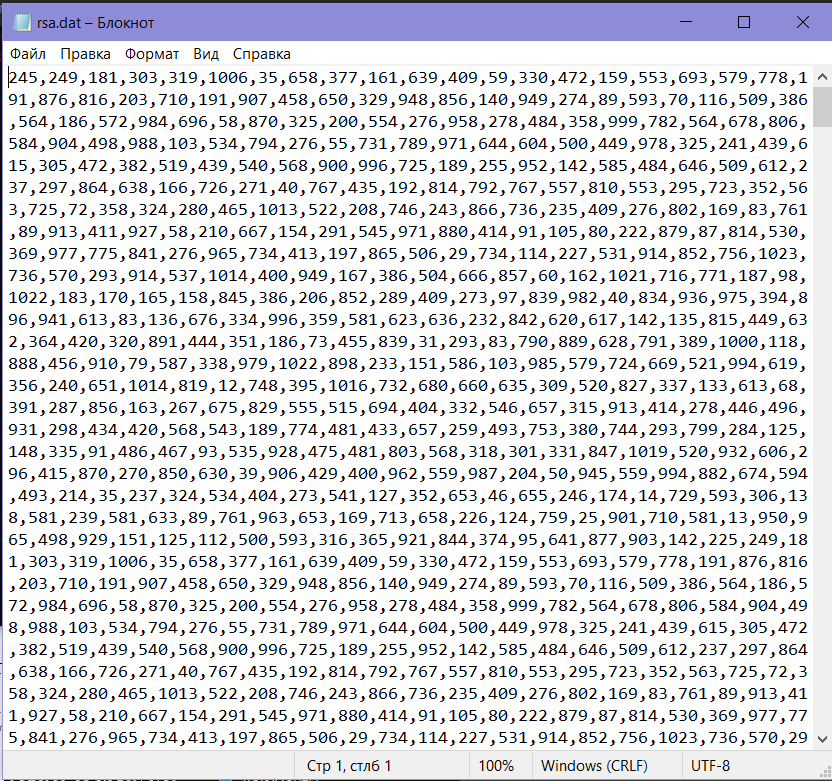
1. Сгенерировать два секретных простых числа *p* и *q*, а также и . Выбрать случайное целое число , такое что
2. Выбрать случайное целое – начальный вектор из интервала
3. .

**Пример ввода:**

prng.exe -g rsa -i 12709189,53,10,245 -n 10000 -f rsa.dat

**Пример работы программы:**





**1.9 Алгоритм Блюма-Блюма-Шуба**

**Описание алгоритма:**

**На входе:** Длина *l*.

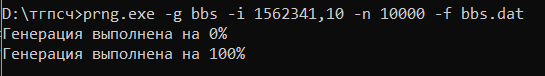
**На выходе:** Последовательность псевдослучайных бит .

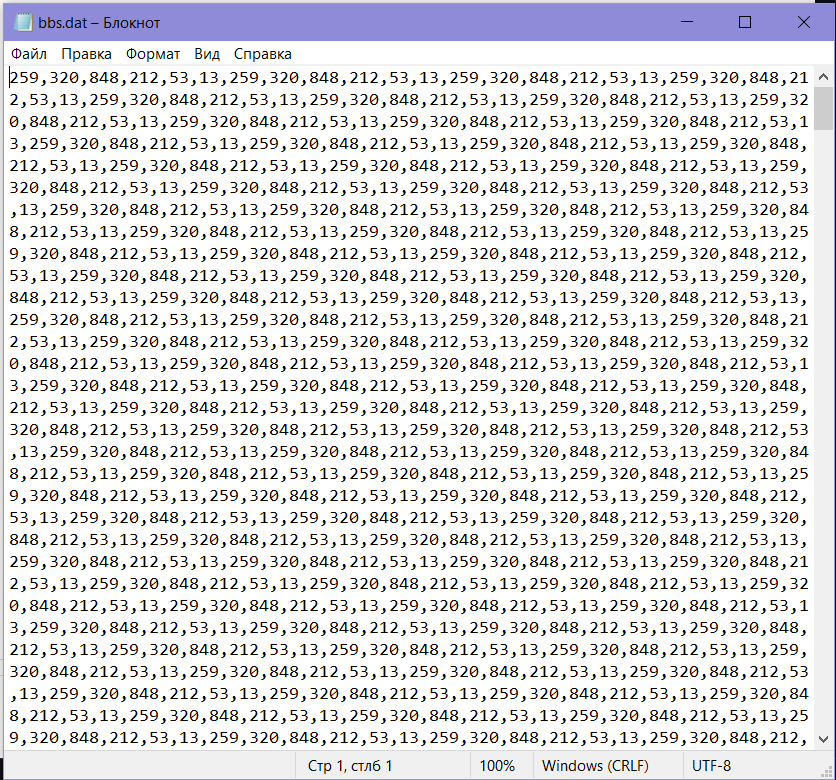
1. Сгенерировать два простых числа *p* и *q*, сравнимых с 3 по модулю 4. Это гарантирует, что каждый квадратичный вычет имеет один квадратный корень, который также является квадратичным вычетом. Произведение этих чисел – *n=pq* является целым числом Блюма. Выберем другое случайное целое число *x*, взаимно простое с *n*.
2. Вычислим , которое будет начальным вектором.
3. .

**Пример ввода:**

prng.exe -g bbs -i 1562341,10 -n 10000 -f bbs.dat

**Пример работы программы:**





# **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

На входе

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

Для управления приложением предлагается следующий формат параметров командной строки:

-f <имя\_файла> - имя файла с входной последовательностью.

-d <распределение> - код распределения для преобразования последовательности. Рекомендуется использовать следующие коды распределений:

• 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;

• bi – биномиальное распределение.

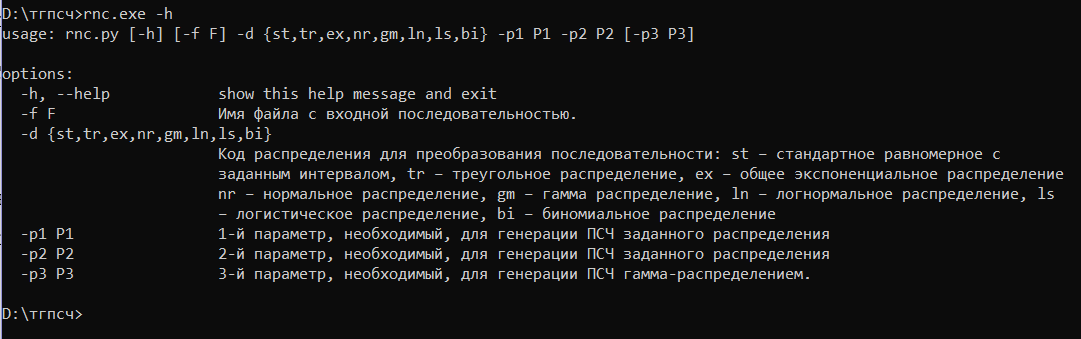
-p1 <параметр1> - 1-й параметр, необходимый, для генерации ПСЧ заданного распределения.

-p2 <параметр2> - 2-й параметр, необходимый, для генерации ПСЧ заданного распределения.

-p3 <параметр3> - 3-й параметр, необходимый, для генерации ПСЧ гамма-распределением.

На выходе

Текстовый файл distr-xx.dat с преобразованными числами, где <xx> – код распределения.



## **2.1 Стандартное равномерное с заданным интервалом**

**Описание алгоритма:**

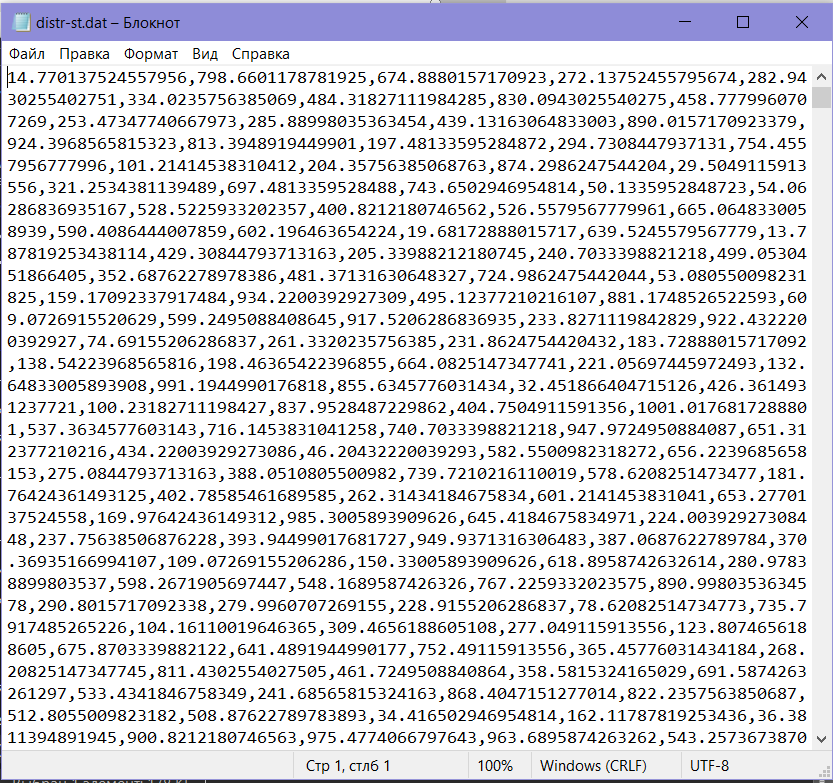
Если стандартное равномерное случайное число получено методом, установленным в предыдущем параграфе, то равномерное случайное число должно быть получено в соответствии со следующей формулой

**Пример ввода:**

rnc.exe -f lc.dat -d st -p1 2 -p2 1000

**Пример работы программы:**





## **2.2 Треугольное распределение**

**Описание алгоритма:**

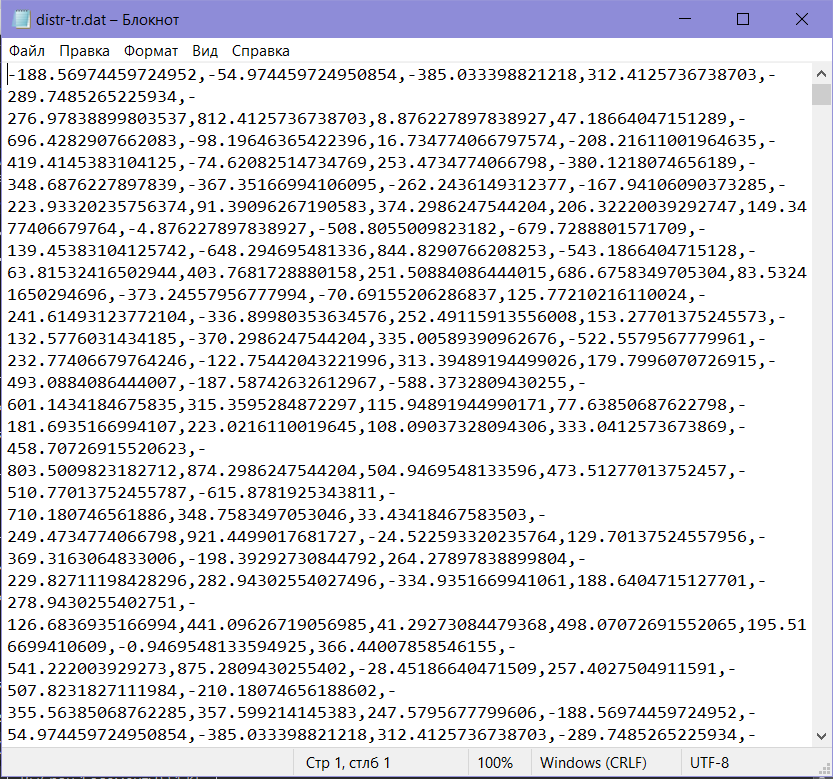
Если стандартные случайные числа и независимо получены методом генерации стандартного равномерного числа, то случайное число *Y*, подчиняющееся треугольному распределению, определяют по формуле .

**Пример ввода:**

rnc.exe -f lc.dat -d tr -p1 2 -p2 1000

**Пример работы программы:**





## **2.3 Общее экспоненциальное распределение**

**Описание алгоритма:**

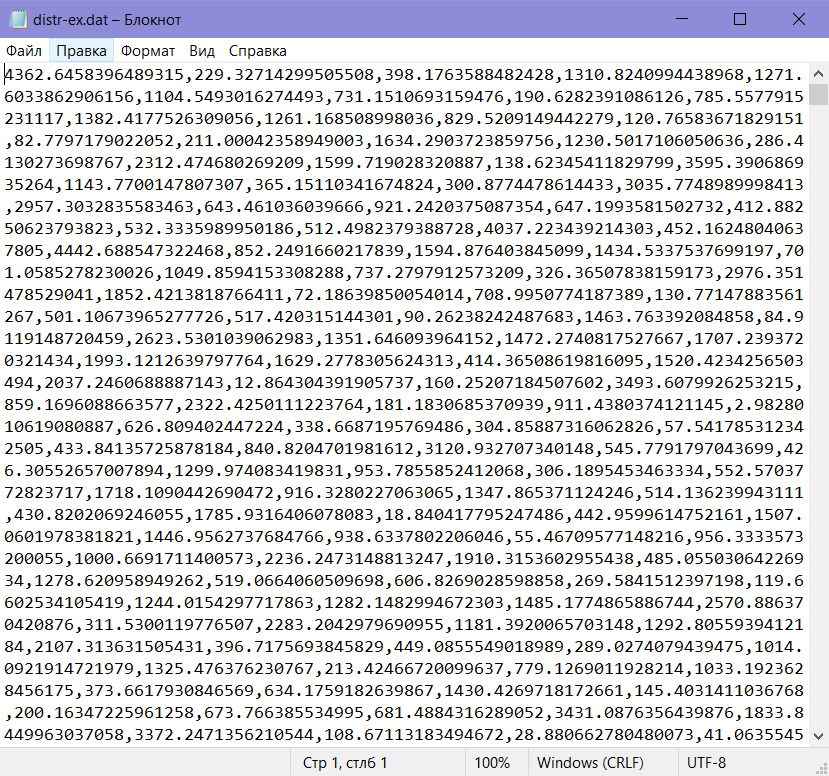
Если стандартное равномерное случайное число *U* генерировано одним из методов, установленным в разделе 2, то случайное число, соответствующее экспоненциальному распределению, получают по формуле

**Пример ввода:**

rnc.exe -f lc.dat -d ex -p1 2 -p2 1000

**Пример работы программы:**





## **2.4 Нормальное распределение**

**Описание алгоритма:**

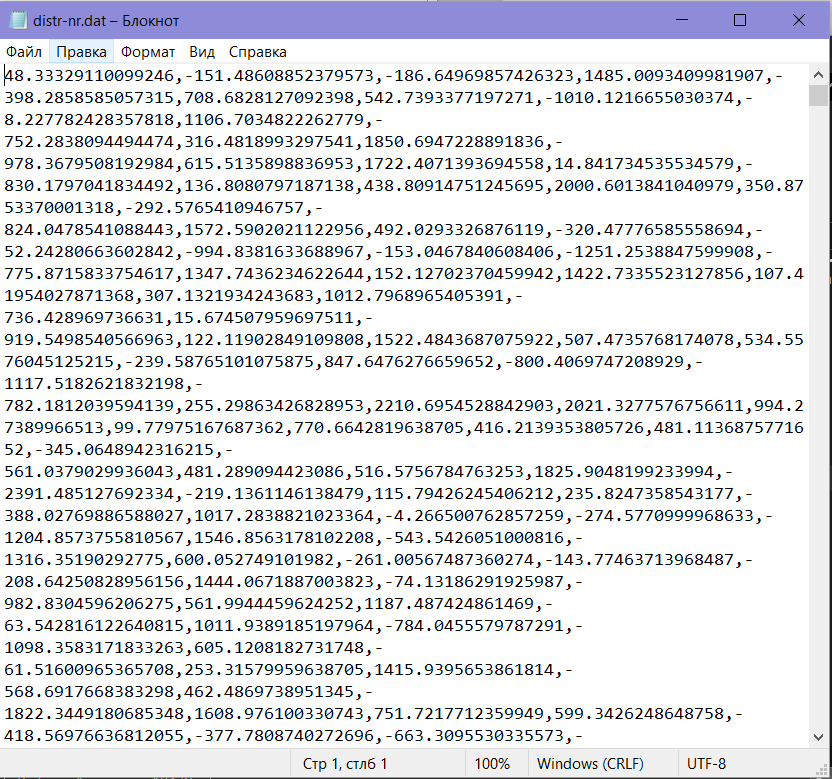
Если стандартные равномерные случайные числа и независимо сгенерированы методом, установленным в разделе 2, то два независимых нормальных случайных числа получают в соответствии со следующей процедурой

**Пример ввода:**

rnc.exe -f lc.dat -d nr -p1 2 -p2 1000

**Пример работы программы:**





## **2.5 Гамма распределение (для параметра c=k)**

**Описание алгоритма:**

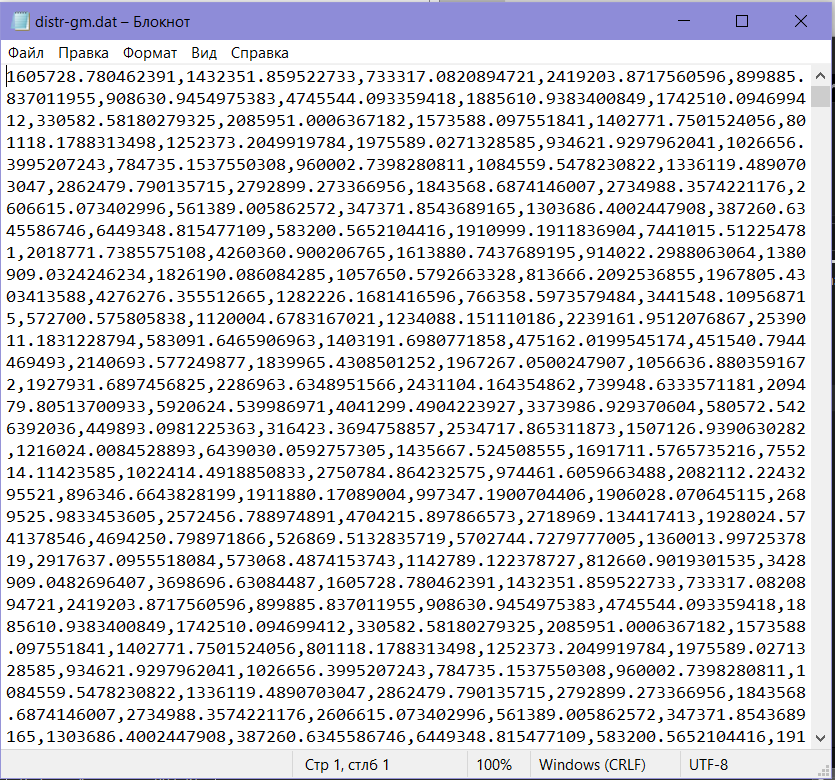
Используя независимые равномерные случайные числа , применяют формулу

**Пример ввода:**

rnc.exe -f lc.dat -d gm -p1 0 -p2 1000000 -p3 2

**Пример работы программы:**





## **2.6 Логнормальное распределение**

**Описание алгоритма:**

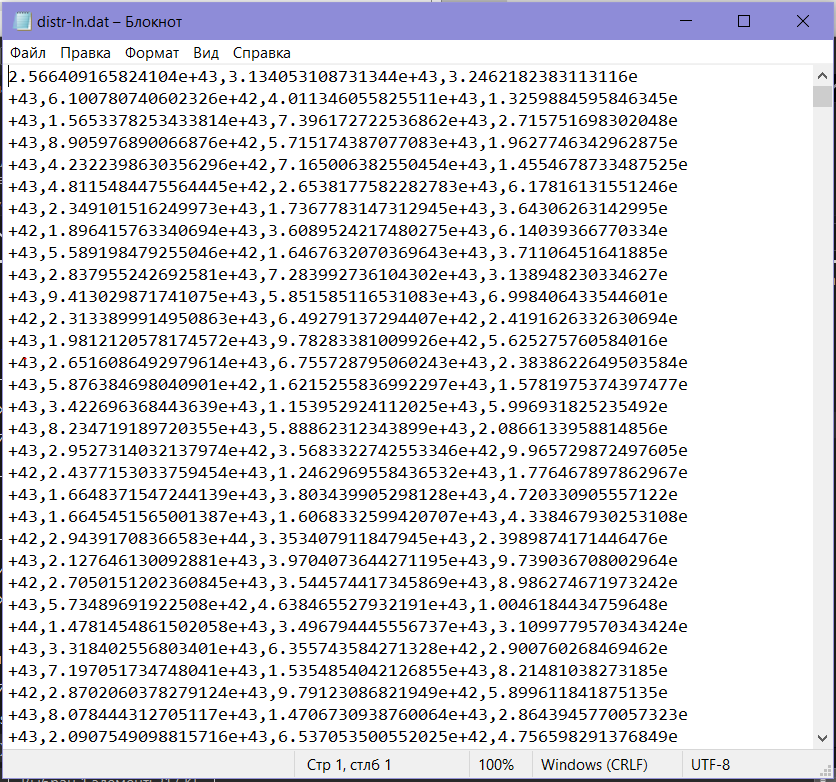
Используя стандартные нормальные случайные числа *Z*, применяют формулу для получения случайных чисел, соответствующих логнормальному распределению.

**Пример ввода:**

rnc.exe -f lc.dat -d ln -p1 2 -p2 100

**Пример работы программы:**





## **2.7 Логистическое распределение**

**Описание алгоритма:**

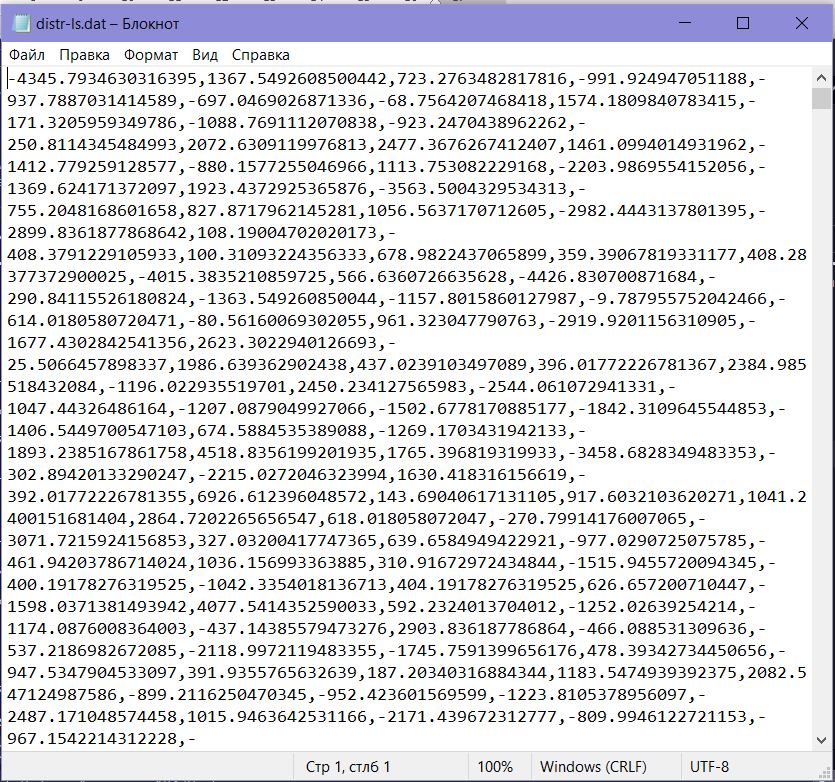
Если стандартные равномерные случайные числа U генерированы методом, изложенным выше, то случайные числа, соответствующие логистическому распределению, получают по формуле

**Пример ввода:**

rnc.exe -f lc.dat -d ls -p1 2 -p2 1000

**Пример работы программы:**





## **2.8 Биномиальное распределение**

**Описание алгоритма:**

Вычисляют функцию распределения

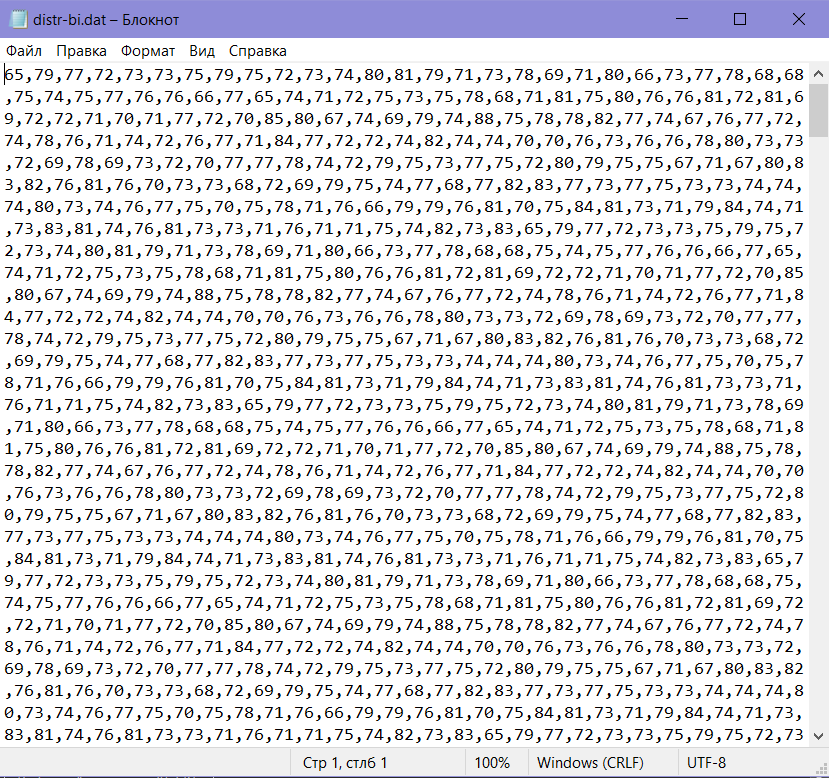
Для получения случайного числа *Y* генерируют стандартное равномерное случайное число *U*. Случайное число *Y* является наименьшим значением *y*, для которого

**Пример ввода:**

rnc.exe -f lc.dat -d bi -p1 0.75 -p2 100

**Пример работы программы:**





# **ПРИЛОЖЕНИЕ А**

**Программа для задания 1**

import random

import sys

import math

import argparse

def lc(m, a, c, x0, n):

    if m <= 0 or a > m or a < 0 or c > m or c < 0 or x0 > m or x0 < 0:

        print("Ошибка!")

        return

    print('Генерация выполнена на 0%')

    x = [0] \* n

    x[0] = x0

    for i in range(1, n):

            x[i] = (a \* x[i - 1] + c) % m

            if (i / n \* 100 == 25):

                print('Генерация выполнена на 25%')

            if (i / n \* 100 == 50):

                print('Генерация выполнена на 50%')

            if (i / n \* 100 == 75):

                print('Генерация выполнена на 75%')

    print('Генерация выполнена на 100%')

    return x

def add(m, k, j, starts, n):

    if m <= 0 or k >= j or k < 1 or j + 3 > len(starts):

        print("Ошибка!")

        return

    print('Генерация выполнена на 0%')

    x = starts

    for i in range(n):

        x.append((x[i - k] + x[i - j]) % m)

        if (i / n \* 100 == 25):

            print('Генерация выполнена на 25%')

        if (i / n \* 100 == 50):

            print('Генерация выполнена на 50%')

        if (i / n \* 100 == 75):

            print('Генерация выполнена на 75%')

    print('Генерация выполнена на 100%')

    return x

def get\_bit(num, num\_bit):

    return (num & ( 1 << num\_bit )) >> num\_bit

def set\_bit(num, num\_bit, bit):

    mask = 1 << num\_bit

    num &= ~mask

    if bit:

        return num | mask

    else:

        return num

def shift(num, s):

    new\_num = 0

    bit = 0

    for i in range(s):

        new\_num = set\_bit(new\_num, i, bit)

        bit = get\_bit(num, i)

    new\_num = set\_bit(new\_num, 0, bit)

    return new\_num

def lfsr(x0, reg, n):

    x = []

    print('Генерация выполнена на 0%')

    lenreg = len(reg)

    reg = int(reg, 2)

    x0 = int(x0, 2)

    for i in range(n):

        cur\_bit = 0

        for j in range(lenreg):

            cur\_bit ^= get\_bit(x0, j) \* get\_bit(reg, j)

        reg = shift(reg, lenreg)

        reg = set\_bit(reg, 0, cur\_bit)

        x.append(reg)

        if (i / n \* 100 == 25):

            print('Генерация выполнена на 25%')

        if (i / n \* 100 == 50):

            print('Генерация выполнена на 50%')

        if (i / n \* 100 == 75):

            print('Генерация выполнена на 75%')

    print('Генерация выполнена на 100%')

    return(x)

def p5(p,q1,q2,q3,w,x0,n):

    if q1 >= p or q2 >= p or q3 >= p:

        print("Ошибка!")

        return

    x = []

    x.append(x0)

    mask\_w = 0

    for i in range(w):

        mask\_w = set\_bit(mask\_w, i, 1)

    mask\_p = 0

    for i in range(p):

        mask\_p = set\_bit(mask\_p, i, 1)

    print('Генерация выполнена на 0%')

    for i in range(n):

        cur\_bit = 0

        cur\_bit ^= get\_bit(x0, q1)

        cur\_bit ^= get\_bit(x0, q2)

        cur\_bit ^= get\_bit(x0, q3)

        cur\_bit ^= get\_bit(x0, 0)

        x0 = shift(x0, p)

        x0 = set\_bit(x0, 0, cur\_bit)

        x0 = x0 & mask\_p

        x.append(x0 & mask\_w)

        if (i / n \* 100 == 25):

            print('Генерация выполнена на 25%')

        if (i / n \* 100 == 50):

            print('Генерация выполнена на 50%')

        if (i / n \* 100 == 75):

            print('Генерация выполнена на 75%')

    print('Генерация выполнена на 100%')

    return x

def lfsr\_help(x0, reg, n):

    x = []

    lenreg = len(reg)

    reg = int(reg, 2)

    for i in range(n):

        cur\_bit = 0

        for j in range(lenreg):

            cur\_bit ^= get\_bit(x0, j) \* get\_bit(reg, j)

        reg = shift(reg, lenreg)

        reg = set\_bit(reg, 0, cur\_bit)

        x.append(reg)

    return(x)

# print(p5(87, 20, 40, 69, 9, 712, 100))

def nfsr(r1, r2, r3, w, x1, x2, x3, n):

    len\_R1 = len(r1)

    len\_R2 = len(r2)

    len\_R3 = len(r3)

    R1 = lfsr\_help(x1, r1, n)

    R2 = lfsr\_help(x2, r2, n)

    R3 = lfsr\_help(x3, r3, n)

    w1 = 0

    x = []

    print('Генерация выполнена на 0%')

    for i in range(int(w)):

        w1 = set\_bit(w1, i, 1)

    for i in range(n):

        x.append(((R1[i] ^ R2[i]) + (R2[i] ^ R3[i]) + R3[i]) & w1)

        if (i / n \* 100 == 25):

            print('Генерация выполнена на 25%')

        if (i / n \* 100 == 50):

            print('Генерация выполнена на 50%')

        if (i / n \* 100 == 75):

            print('Генерация выполнена на 75%')

    print('Генерация выполнена на 100%')

    return x

#print(nfsr("100000001001001", "0011000000", "101011001001001", 9, 25, 60, 45, 100))

def mt(mod, x0, n):

    p, w, r, q, a, u, s, t, l, b, c = 624, 32, 31, 397, 2567483615, 11, 7, 15, 18, 2636928640, 4022730752

    lower\_mask = (1 << r) - 1

    w1 = 0

    for i in range(w):

        w1 = set\_bit(w1, i, 1)

    upper\_mask = (~lower\_mask \* -1) & w1

    res = []

    print('Генерация выполнена на 0%')

    MT = []

    MT.append(x0)

    for i in range(1, p):

        MT.append((MT[i - 1] ^ (MT[i - 1] >> 30)) + i)

    ind = p

    for j in range(n):

        if (ind >= p):

            for i in range(p):

                x = (MT[i] & upper\_mask) + (MT[(i + 1) % p] & lower\_mask)

                xA = x >> 1

                if (x & 1):

                    xA ^= a

                MT[i] = MT[(i + q) % p] ^ xA

            ind = 0

        y = MT[ind]

        ind += 1

        y ^= (y >> u)

        y ^= (y << s) & b

        y ^= (y << t) & c

        y ^= (y >> l)

        res.append(y % mod)

        if (j / n \* 100 == 25):

            print('Генерация выполнена на 25%')

        if (j / n \* 100 == 50):

            print('Генерация выполнена на 50%')

        if (j / n \* 100 == 75):

            print('Генерация выполнена на 75%')

    print('Генерация выполнена на 100%')

    return res

#print(mt(1000, 1234, 100))

def rc4(k, n):

    s = [i for i in range(256)]

    j = 0

    for i in range(256):

        j = (j + s[i] + k[i]) % 256

        s[i], s[j] = s[j], s[i]

    i, j = 0, 0

    x = []

    print('Генерация выполнена на 0%')

    for k in range(n):

        num = 0

        i = (i + 1) % 256

        j = (j + s[i]) % 256

        s[i], s[j] = s[j], s[i]

        num = s[(s[i] + s[j]) % 256]

        x.append(num)

        if (k / n \* 100 == 25):

            print('Генерация выполнена на 25%')

        if (k / n \* 100 == 50):

            print('Генерация выполнена на 50%')

        if (k / n \* 100 == 75):

            print('Генерация выполнена на 75%')

    print('Генерация выполнена на 100%')

    return x

#print(rc4([213,968,838,64,355,214,212,36,695,139,897,518,656,956,810,510,985,105,670,8,907,951,685,989,222,931,169,286,289,556,731,902,688,701,771,533,990,630,708,884,255,683,25,214,792,348,34,758,9,781,946,580,615,955,585,5,886,563,81,38,809,444,619,222,544,53,635,621,630,251,497,257,2,467,897,790,728,676,722,838,465,781,10,828,903,235,857,841,146,719,681,678,961,652,491,38,256,909,251,21,110,811,273,25,642,286,489,478,184,812,770,846,241,141,266,500,375,827,633,761,154,663,461,206,529,212,667,342,360,165,523,749,582,803,553,345,786,990,361,702,256,380,234,238,73,965,266,300,847,755,969,681,146,843,125,306,845,752,879,458,788,833,727,817,122,239,765,877,827,327,733,658,644,880,150,474,493,689,670,368,611,263,113,417,834,103,725,754,117,824,623,338,540,337,879,521,183,370,808,120,571,871,301,210,796,744,398,106,845,745,842,876,399,27,105,601,802,831,53,266,157,352,175,303,505,484,994,425,292,729,654,584,860,420,412,49,281,417,703,400,48,404,772,389,733,152,271,585,404,333,381,696,928,609,659,180,9], 100))

def rsa(N, e, w, x0, n):

    x = []

    x.append(x0)

    print('Генерация выполнена на 0%')

    for i in  range(n):

        z = 0

        for j in range(w):

            x0 = pow(x0, e, N)

            z = set\_bit(z, w - j - 1, x0 & 1)

        x.append(z)

        if (i / n \* 100 == 25):

            print('Генерация выполнена на 25%')

        if (i / n \* 100 == 50):

            print('Генерация выполнена на 50%')

        if (i / n \* 100 == 75):

            print('Генерация выполнена на 75%')

    print('Генерация выполнена на 100%')

    return x

#print(rsa(12709189, 53, 10, 245, 100))

def bbs(x0, w, N):

    p = 127

    q = 131

    n = p \* q

    x = []

    print('Генерация выполнена на 0%')

    for i in range(N):

        z = 0

        for j in range(w):

            x0 = x0 \* x0 % n

            z = set\_bit(z, w - j - 1, x0 & 1)

        x.append(z)

        if (i / n \* 100 == 25):

            print('Генерация выполнена на 25%')

        if (i / n \* 100 == 50):

            print('Генерация выполнена на 50%')

        if (i / n \* 100 == 75):

            print('Генерация выполнена на 75%')

    print('Генерация выполнена на 100%')

    return x

#print(bbs(15621, 10, 100))

def write\_to\_file(data, filepath="rnd.dat"):

    with open(filepath, "w", encoding="UTF-8") as f:

        f.write(data)

if \_\_name\_\_ == "\_\_main\_\_":

    parser = argparse.ArgumentParser(prog="prng.py")

    helpg = """

            -g <код\_метода> -- параметр указывает на метод генерации ПСЧ, при этом код\_метода может быть

            одним из следующих:\n

            1) lc – линейный конгруэнтный метод;\n

            2) add – аддитивный метод;\n

            3) 5p – пятипараметрический метод;\n

            4) lfsr – регистр сдвига с обратной связью (РСЛОС);\n

            5) nfsr – нелинейная комбинация РСЛОС;\n

            6) mt – вихрь Мерсенна;\n

            7) rc4 – RC4;\n

            8) rsa – ГПСЧ на основе RSA;\n

            9) bbs – алгоритм Блюма-Блюма-Шуба.\n

"""

    helpi = """

            -i <парметры> -- параметр указывает на парметры генерации ПСЧ, при этом для -g могут быть

            одними из следующих:\n

            1) lc /i: модуль, множитель, приращение, начальное значение;\n

            2) add /i: модуль, младший индекс, старший индекс, последовательность начальных значений;\n

            3) 5p /i: p, q\_1, q\_2, q\_3, w, x0;\n

            4) lfsr /i: двоичное представление вектора коэффициентов, начальное значение регистра;\n

            5) nfsr /i: двоичные представления векторов коэффициентов для R1, R2, R3, скомбинированных функцией R1^R2 + R2^R3 + R3;\n

            6) mt /i: модуль, начальное значение x;\n

            7) rc4 /i: 256 начальных значений;\n

            8) rsa /i: модуль n, число e, начальное значение x. e удовлетворяет: 1 < e < (p - 1) \* (q - 1), НОД(e, (p - 1) \* (q - 1)) = 1, p \* q = n, x из [1, n];\n

            9) bbs /i: начальное значение x (взаимно простое с n). При генерации используются параметры: p=127, q=131, n=p\*q=16637.\n

"""

    helpn = '-n <длина> -- количество генерируемых чисел. Если параметр не указан, -- генерируется 10000 чисел.\n'

    helpf = '-f <полное\_имя\_файла> -- полное имя файла, в который будут выводиться данные. Если параметр не указан, данные будут записаны в файл с именем rnd.dat.\n'

    helph = '-h информация о допустимых параметрах командной строки программы.'

    parser.add\_argument("-g", help=helpg, required=True, choices=["lc", "add", "5p", "lfsr", "nfsr", "mt", "rc4", "rsa", "bbs"], nargs=1)

    parser.add\_argument("-i", type=str, help=helpi)

    parser.add\_argument("-n", nargs=1, type=int, default=[10000], help=helpn)

    parser.add\_argument("-f", nargs=1, default=["rnd.dat"], help=helpf)

    args = parser.parse\_args()

    def check(n, args):

        if len(args) != n:

                raise Exception("Передано неверное количество аргументов")

        for arg in args:

            if not arg.isdigit():

                raise Exception("Переданы нечисловые параметры")

        return True

    g\_name = args.g[0]

    i\_args = args.i.split(sep=",")

    f\_path = args.f[0]

    n\_ = args.n[0]

    try:

        match g\_name:

            case 'lc':

                if check(4, i\_args):

                    m, a, c = int(i\_args[0]), int(i\_args[1]), int(i\_args[2])

                    x0 = int(i\_args[3])

                    x = lc(m, a, c, x0, n\_)

                res\_str = ",".join(map(str, x))

                write\_to\_file(res\_str, filepath=f\_path)

            case 'add':

                if check(4, i\_args):

                    m = int(i\_args[0])

                    k, j = int(i\_args[1]), int(i\_args[2])

                    starts = list(map(int, i\_args[3:]))

                    x = add(m, k, j, starts, n\_)

                res\_str = ",".join(map(str, x))

                write\_to\_file(res\_str, filepath=f\_path)

            case '5p':

                if check(6, i\_args):

                    p, q1, q2, q3, w = int(i\_args[0]), int(i\_args[1]), int(i\_args[2]), int(i\_args[3]), int(i\_args[4])

                    x0 = int(i\_args[5])

                    x = p5(p, q1, q2, q3, w, x0, n\_)

                    res\_str = ",".join(map(str, x))

                    write\_to\_file(res\_str, filepath=f\_path)

            case 'lfsr':

                if check(2, i\_args):

                    x0, reg = i\_args[0], i\_args[1]

                    x = lfsr(x0, reg, n\_)

                    res\_str = ",".join(map(str, x))

                    write\_to\_file(res\_str, filepath=f\_path)

            case 'nfsr':

                if check(7, i\_args):

                    r1, r2, r3 = i\_args[0], i\_args[1], i\_args[2]

                    w, x1, x2, x3 = int(i\_args[3]), int(i\_args[4]), int(i\_args[5]), int(i\_args[6])

                    x = nfsr(r1, r2, r3, w, x1, x2, x3, n\_)

                    res\_str = ",".join(map(str, x))

                    write\_to\_file(res\_str, filepath=f\_path)

            case 'mt':

                if check(2, i\_args):

                    mod = int(i\_args[1])

                    x0 = int(i\_args[0])

                    x = mt(mod, x0, n\_)

                    res\_str = ",".join(map(str, x))

                    write\_to\_file(res\_str, filepath=f\_path)

            case 'rc4':

                if check(256, i\_args):

                    k = list(map(int, i\_args))

                    x = rc4(k, n\_)

                    res\_str = ",".join(map(str, x))

                    write\_to\_file(res\_str, filepath=f\_path)

            case 'rsa':

                if check(4, i\_args):

                    N, e, w, x0 = int(i\_args[0]), int(i\_args[1]), int(i\_args[2]), int(float(i\_args[3]))

                    x = rsa(N, e, w, x0, n\_)

                    res\_str = ",".join(map(str, x))

                    write\_to\_file(res\_str, filepath=f\_path)

            case 'bbs':

                if check(2, i\_args):

                    x0, w = int(i\_args[0]), int(i\_args[1])

                    x = bbs(x0, w, n\_)

                    res\_str = ",".join(map(str, x))

                    write\_to\_file(res\_str, filepath=f\_path)

    except Exception as err:

        print("В процессе генерации произошла ошибка! ", str(err))

# **ПРИЛОЖЕНИЕ Б**

**Программа для задания 2**

import argparse

import numpy as np

def U(x, m):

    return x / m

def st(a, b, l):

    y = []

    m = max(l) + 1

    for x in l:

        y.append(a + U(x ,m) \* b)

    return y

def tr(a, b, l):

    y = []

    m = max(l) + 1

    for i in range(0, len(l)-1, 2):

        y.append(a + b \* (U(l[i], m) + U(l[i + 1], m) - 1))

    return y

def ex(a, b, l):

    y = []

    m = max(l) + 1

    for x in l:

        y.append(a - b \* np.log(U(x, m)))

    return y

def nr(a, b, l):

    y = []

    m = max(l) + 1

    for i in range(0, len(l)-1, 2):

        y.append(a + b \* np.sqrt(-2 \* np.log(1 - U(l[i], m))) \* np.cos(2 \* np.pi \* U(l[i+1], m)))

        y.append(a + b \* np.sqrt(-2 \* np.log(1 - U(l[i], m))) \* np.sin(2 \* np.pi \* U(l[i+1], m)))

    return y

def gm(a, b, c, l):

    y = []

    m = max(l) + 1

    u = []

    uk = []

    for x in l:

        u.append(U(x, m))

    for i in range(0, len(u), c):

        uk.append((u[i : i + c]))

    if len(uk[-1]) != c:

        uk.pop()

    for mass in uk:

        x = 1

        for el in mass:

            x \*= 1 - el

        y.append(a - b \* np.log(x))

    return y

def ln(a, b, l):

    y = []

    m = max(l) + 1

    l = nr(0, 1, l)

    for x in l:

        y.append(a + np.exp(b - x))

    return y

def ls(a, b, l):

    y = []

    m = max(l) + 1

    u = []

    for x in l:

        u.append(U(x, m))

    for x in u:

        y.append(a + b \* np.log(x / (1 - x)))

    return y

def factor(x):

    y = 1

    for i in range(x):

        y \*= (i + 1)

    return y

def bi(a, b, l):

    y = []

    m = max(l) + 1

    u = []

    for x in l:

        u.append(U(x, m))

    for i in u:

        s = 0

        k = 0

        while(True):

            s += (factor(b) / (factor(k) \* factor(b - k)) \* (a \*\* k) \* ((1 - a) \*\* (b - k)))

            if s > i:

                y.append(k)

                break

            if k < b - 1:

                k += 1

                continue

            y.append(b)

    return y

def write\_to\_file(data, f):

    with open(f, "w", encoding="UTF-8") as f:

        f.write(data)

if \_\_name\_\_ == "\_\_main\_\_":

    parser = argparse.ArgumentParser(prog="rnc.py")

    helpd = """Код распределения для преобразования последовательности:\n

                st – стандартное равномерное с заданным интервалом,\n

                tr – треугольное распределение,\n

                ex – общее экспоненциальное распределение\n

                nr – нормальное распределение,\n

                gm – гамма распределение,\n

                ln – логнормальное распределение,\n

                ls – логистическое распределение,\n

                bi – биномиальное распределение"""

    helpf = "Имя файла с входной последовательностью."

    helpp1 = "1-й параметр, необходимый, для генерации ПСЧ заданного распределения"

    helpp2 = "2-й параметр, необходимый, для генерации ПСЧ заданного распределения"

    helpp3 = "3-й параметр, необходимый, для генерации ПСЧ гамма-распределением."

    parser.add\_argument("-f", nargs=1, default=["rnd.dat"], help=helpf)

    parser.add\_argument("-d", help=helpd, required=True, choices=["st", "tr", "ex", "nr", "gm", "ln", "ls", "bi"], nargs=1)

    parser.add\_argument("-p1", nargs=1, type=float, required=True, help=helpp1)

    parser.add\_argument("-p2", nargs=1, type=int, required=True, help=helpp2)

    parser.add\_argument("-p3", nargs=1, type=int, help=helpp3, default=[None])

    args = parser.parse\_args()

    def check(n, args):

        if len(args) != n:

                raise Exception("Передано неверное количество аргументов")

        for arg in args:

            if not arg.isdigit():

                raise Exception("Переданы нечисловые параметры")

        return True

    d\_name = args.d[0]

    f\_name = args.f[0]

    p1 = args.p1[0]

    p2 = args.p2[0]

    p3 = args.p3[0]

    try:

        with open(f\_name, "r") as f:

            line = f.readline()

            l  = list(map(int, line.split(",")))

        match d\_name:

            case 'st':

                    y = st(p1, p2, l)

                    res\_str = ",".join(map(str, y))

                    write\_to\_file(res\_str, "distr-st.dat")

            case 'tr':

                    y = tr(p1, p2, l)

                    res\_str = ",".join(map(str, y))

                    write\_to\_file(res\_str, "distr-tr.dat")

            case 'ex':

                    y = ex(p1, p2, l)

                    res\_str = ",".join(map(str, y))

                    write\_to\_file(res\_str, "distr-ex.dat")

            case 'nr':

                    y = nr(p1, p2, l)

                    res\_str = ",".join(map(str, y))

                    write\_to\_file(res\_str, "distr-nr.dat")

            case 'gm':

                    y = gm(p1, p2, p3, l)

                    res\_str = ",".join(map(str, y))

                    write\_to\_file(res\_str, "distr-gm.dat")

            case 'ln':

                    y = ln(p1, p2, l)

                    res\_str = ",".join(map(str, y))

                    write\_to\_file(res\_str, "distr-ln.dat")

            case 'ls':

                    y = ls(p1, p2, l)

                    res\_str = ",".join(map(str, y))

                    write\_to\_file(res\_str, "distr-ls.dat")

            case 'bi':

                    y = bi(p1, p2, l)

                    res\_str = ",".join(map(str, y))

                    write\_to\_file(res\_str, "distr-bi.dat")

    except Exception as err:

        print("В процессе генерации произошла ошибка! " + str(err))