

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

подпись, дата

И.И. Слеповичев

Саратов 2024

СОДЕРЖАНИЕ

Задание 1. Генерация псевдослучайных чисел.	3
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	12
1.7 RC4	14
1.8 ГПСЧ на основе RSA	16
1.9 Алгоритм Блума-Блума-Шуба	18
Задание 2. Преобразование ПСЧ к заданному распределению.	20
2.1 Стандартное равномерное с заданным интервалом	21
2.2 Треугольное распределение	22
2.3 Общее экспоненциальное распределение	23
2.4 Нормальное распределение	24
2.5 Гамма распределение (для параметра $c=k$)	25
2.6 Логнормальное распределение	26
2.7 Логистическое распределение	27
2.8 Биномиальное распределение	28
ПРИЛОЖЕНИЕ А	30
ПРИЛОЖЕНИЕ Б	40

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.

```
D:\trnc4>prng.exe -h
usage: prng.py [-h] -g {lc,add,5p,lfsr,nfsr,mt,rc4,rsa,bbs} [-i I] [-n N] [-f F]

options:
  -h, --help            show this help message and exit
  -g {lc,add,5p,lfsr,nfsr,mt,rc4,rsa,bbs}
                        -g <код_метода> -- параметр указывает на метод генерации ПСЧ, при этом код_метода может быть
                        одним из следующих: 1) lc – линейный конгруэнтный метод; 2) add – аддитивный метод; 3) 5p –
                        пятипараметрический метод; 4) lfsr – регистр сдвига с обратной связью (РСЛОС); 5) nfsr –
                        нелинейная комбинация РСЛОС; 6) mt – вихрь Мерсенна; 7) rc4 – RC4; 8) rsa – ГПСЧ на основе
                        RSA; 9) bbs – алгоритм Блюма-Блюма-Шуба.
  -i I                  -i <параметры> -- параметр указывает на параметры генерации ПСЧ, при этом для -g могут быть
                        одними из следующих: 1) lc /i: модуль, множитель, приращение, начальное значение; 2) add /i:
                        модуль, младший индекс, старший индекс, последовательность начальных значений; 3) 5p /i: p,
                        q_1, q_2, q_3, w, x0; 4) lfsr /i: двоичное представление вектора коэффициентов, начальное
                        значение регистра; 5) nfsr /i: двоичные представления векторов коэффициентов для R1, R2, R3,
                        скомбинированных функцией R1^R2 + R2^R3 + R3; 6) mt /i: модуль, начальное значение x; 7) rc4
                        /i: 256 начальных значений; 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]; 9) bbs /i: начальное значение x (взаимно простое с n). При генерации используются
                        параметры: p=127, q=131, n=p*q=16637.
  -n N                  -n <длина> -- количество генерируемых чисел. Если параметр не указан, -- генерируется 10000
                        чисел.
  -f F                  -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.
```

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:

- $m > 0$, модуль;
- $0 \leq a \leq m$, множитель;

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

Последовательность ПСЧ, получаемая по формуле:

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

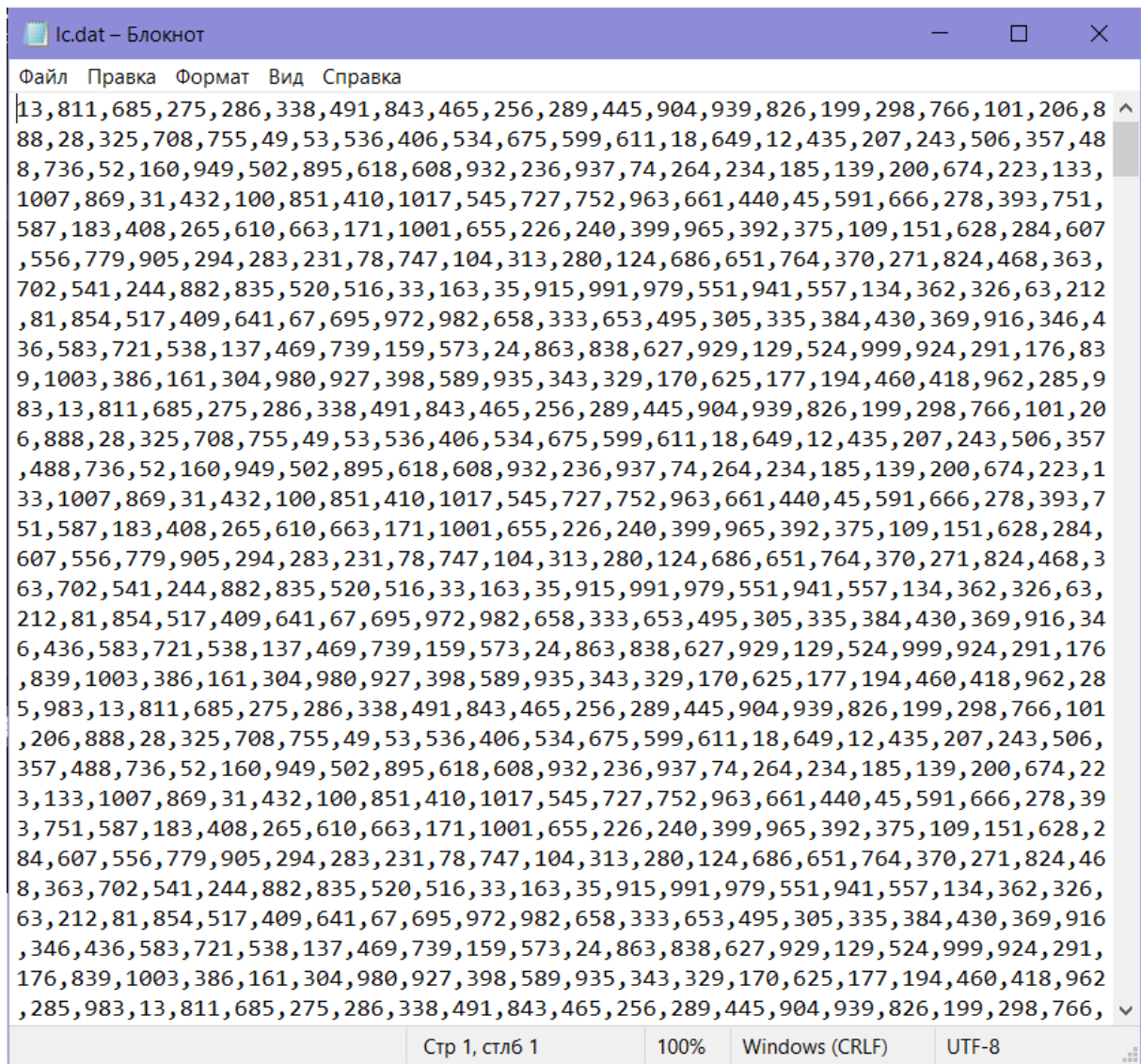
называется линейной конгруэнтной последовательностью (ЛКП). Ключом для неё служит X_0 .

Пример ввода:

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

Пример работы программы:

```
D:\тгпсч>prng.exe -g lc -i 1021,376,7,13 -n 10000 -f lc.dat
Генерация выполнена на 0%
Генерация выполнена на 25%
Генерация выполнена на 50%
Генерация выполнена на 75%
Генерация выполнена на 100%
```



1.2 Аддитивный метод

Описание алгоритма

Формула:

$$X_{n+1} = (X_{n-k} + X_{n-j}) \bmod m, n > j, j > k \geq 1, m > 0.$$

Параметры, поступающие на вход: модуль m , коэффициент k , коэффициент j , числа X_0, \dots, X_n .

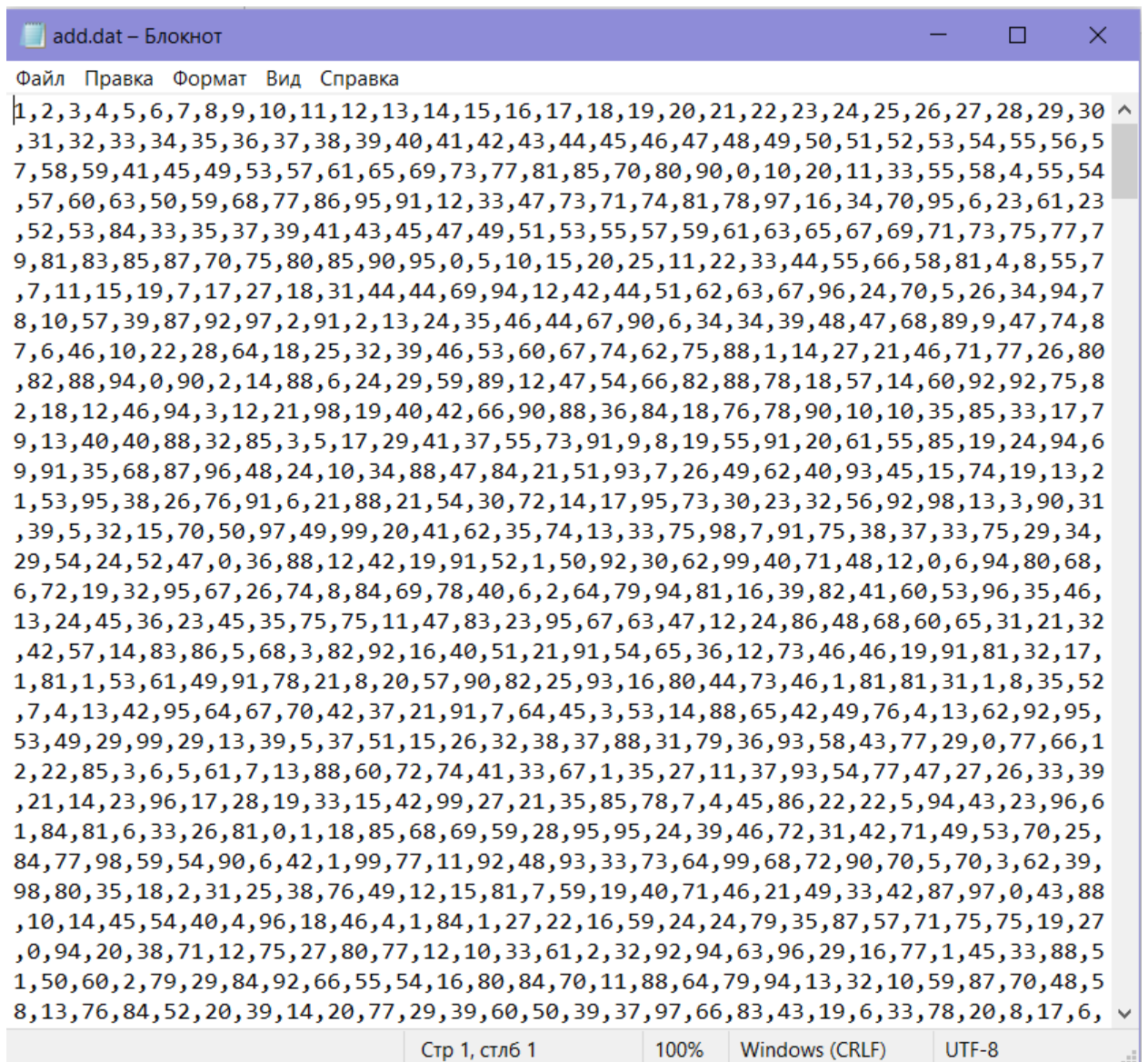
Пример ввода:

```
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:

```
D:\тгпсч>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
Генерация выполнена на 0%
Генерация выполнена на 25%
Генерация выполнена на 50%
Генерация выполнена на 75%
Генерация выполнена на 100%
```



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:

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

При этом $p > w > q_1 > q_2 > q_3 > 0, w > 0$.

Параметры (p, q_1, q_2, q_3, w) и X_1, \dots, X_p , 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:

```
D:\тгпсч>prng.exe -g 5p -i 89,7,13,24,10,764 -n 10000 -f 5p.dat
Генерация выполнена на 0%
Генерация выполнена на 25%
Генерация выполнена на 50%
Генерация выполнена на 75%
Генерация выполнена на 100%

D:\тгпсч>
```


Sp.dat – Блокнот

Файл Правка Формат Вид Справка

764,505,1010,997,970,916,809,594,165,331,662,300,601,179,358,717,411,823,622,221
,442,884,744,465,930,837,651,279,558,93,186,373,747,470,940,857,691,358,717,411,
822,620,217,435,871,718,412,825,626,229,458,916,809,594,165,330,660,297,594,164,
328,656,289,579,134,269,538,53,107,214,429,859,694,365,730,437,875,727,430,861,6
99,375,751,478,956,888,752,481,963,903,782,541,58,116,233,466,933,842,661,298,59
7,171,342,685,347,695,367,734,444,889,754,485,970,916,809,594,165,331,662,301,60
2,181,362,724,425,850,677,330,660,296,592,160,321,642,260,521,18,36,73,146,292,5
85,146,293,586,148,297,595,167,334,668,312,625,227,454,908,793,563,103,207,414,8
28,633,242,485,970,917,811,598,172,344,688,353,707,390,781,538,53,107,215,431,86
2,701,379,758,493,987,950,877,731,438,877,731,439,879,735,447,894,765,507,1014,1
005,986,949,875,727,431,862,700,376,752,480,960,897,771,518,13,27,55,111,223,447
,894,765,507,1014,1004,984,945,867,711,398,796,569,114,229,458,916,809,594,164,3
28,656,289,579,134,268,537,50,100,200,401,803,582,141,283,567,110,220,441,883,74
2,461,923,822,621,218,437,874,724,424,849,674,324,649,274,548,72,144,289,578,132
,265,531,38,76,152,304,608,192,384,769,514,5,11,23,47,94,188,376,752,481,962,901
,779,534,44,88,176,352,705,386,773,523,23,46,93,186,373,746,468,936,849,675,327,
655,286,573,123,246,492,985,947,870,716,408,816,608,192,385,770,516,9,18,36,73,1
46,292,585,146,292,584,145,290,580,136,273,546,68,137,274,548,73,147,295,591,158
,316,632,240,481,962,901,778,532,40,81,163,327,654,285,570,116,232,465,930,836,6
49,275,550,76,153,307,615,206,412,825,626,229,458,917,810,596,169,338,676,328,65
7,290,581,138,277,555,86,173,346,693,362,725,426,853,683,342,684,345,691,359,719
,414,829,634,244,489,978,933,843,662,300,601,179,359,719,414,829,635,246,493,986
,948,873,722,420,841,658,293,586,148,297,594,165,330,660,297,595,166,332,664,305
,611,198,397,794,564,105,210,421,843,663,302,605,187,375,751,478,957,890,756,489
,979,935,847,670,316,632,241,482,964,905,786,548,73,147,294,589,155,311,622,221,
443,887,751,479,959,895,767,510,1021,1019,1015,1006,989,955,886,749,475,950,877,
730,437,875,727,431,862,701,378,756,488,977,930,836,649,275,551,79,158,317,634,2
45,491,983,942,861,699,375,751,478,957,891,758,492,985,947,870,717,410,821,619,2
15,430,861,699,375,750,476,953,883,742,461,923,822,621,219,439,879,734,445,891,7
59,494,988,952,881,739,454,909,794,564,105,211,422,845,666,308,617,211,422,845,6
66,309,619,214,429,858,692,360,720,416,833,642,260,521,19,39,79,159,318,636,248,

Стр 1, стлб 1100%Windows (CRLF)UTF-8

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 a_1, a_2, \dots, a_{p-1} , принимающих значения 0 или 1, определяют рекуррентную формулу

$$X_{n+p} = a_{p-1}X_{n+p-1} + a_{p-2}X_{n+p-2} + \dots + a_1X_{n+1} + X_n, \quad (3.7)$$

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 $p - 1$ 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 3.7.
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.

Пример ввода:

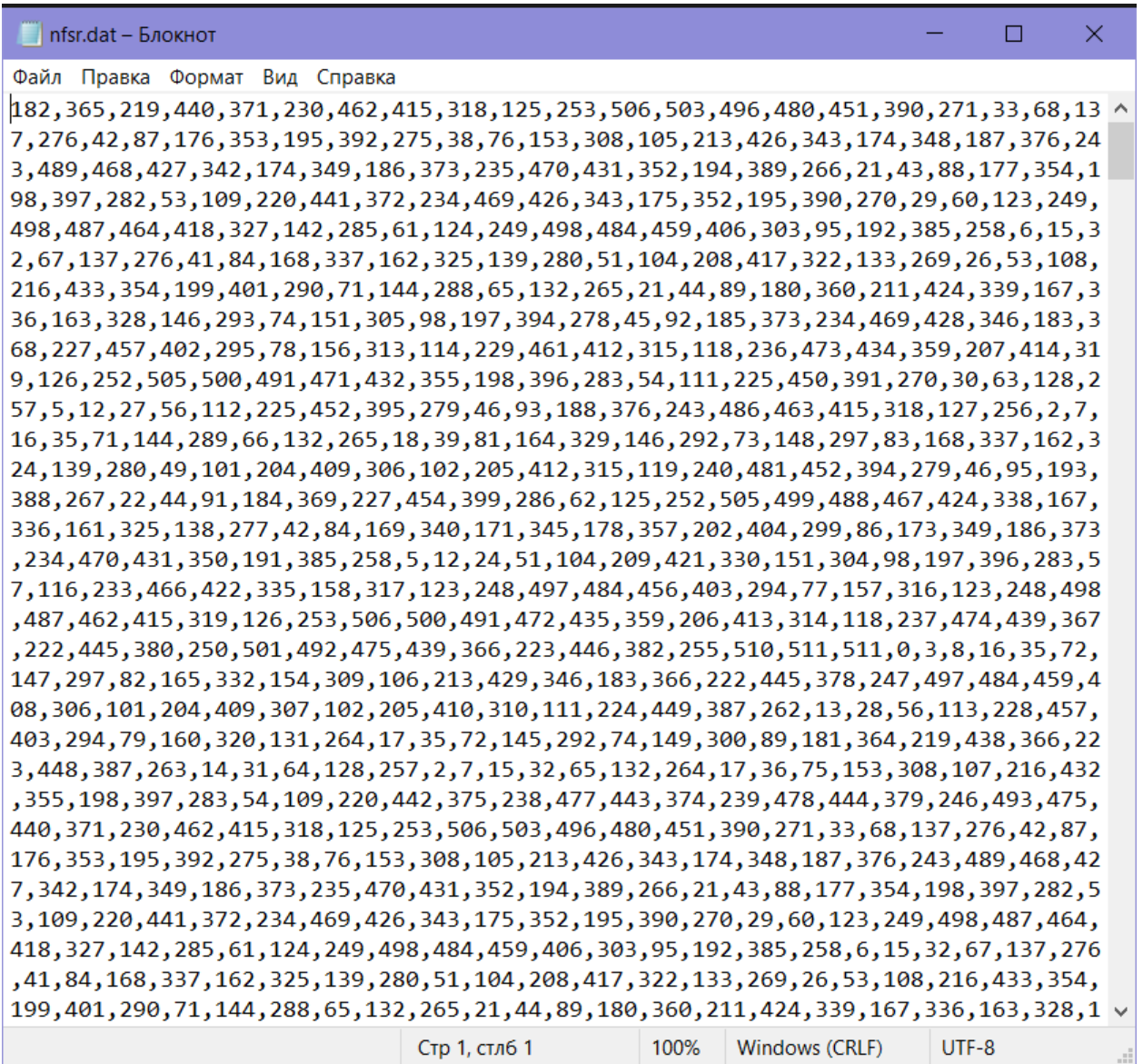
```
prng.exe -g lfsr -i 110110010,0000010011 -n 10000 -f lfsr.dat
```

Пример работы программы:

```
D:\тгпсч>prng.exe -g lfsr -i 110110010,0000010011 -n 10000 -f lfsr.dat
Генерация выполнена на 0%
Генерация выполнена на 25%
Генерация выполнена на 50%
Генерация выполнена на 75%
Генерация выполнена на 100%
```


Пример работы программы:

```
D:\trnpsc>prng.exe -g nfsr -i 100000001001001,0011000000,101011001001001,9,25,60,45 -n 10000 -f nfsr.dat
Генерация выполнена на 0%
Генерация выполнена на 25%
Генерация выполнена на 50%
Генерация выполнена на 75%
Генерация выполнена на 100%
```



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 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.

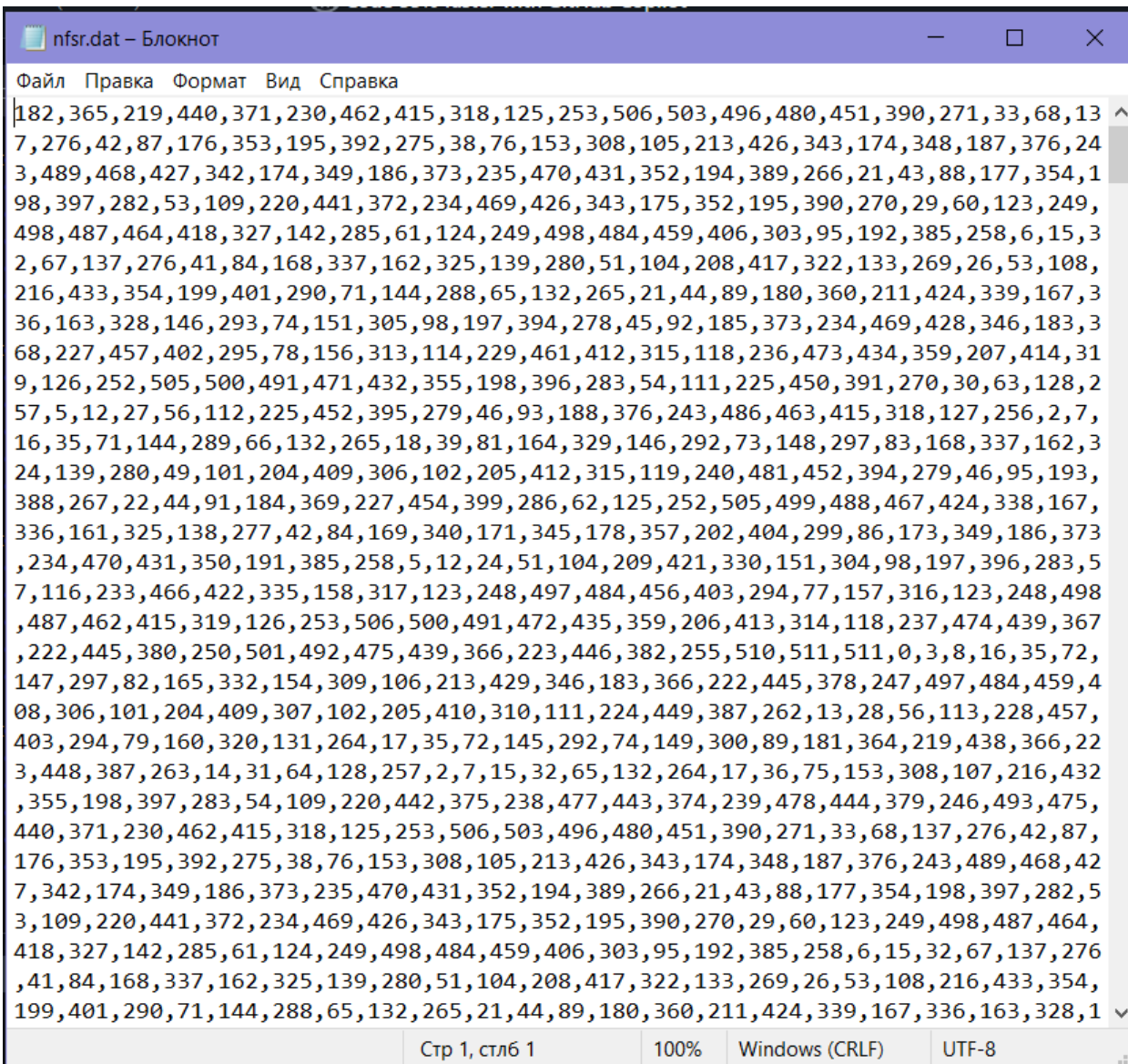
Параметры алгоритма Вихрь Мерсенна: $p = 624$, $w = 32$, $r = 31$,
 $q = 397$, $a = 2567483615$ (9908B0DF₁₆), $u = 11$, $s=7$, $t=15$, $l = 18$, $b =$
 2636928640 (9D2C5680₁₆), $c = 4022730752$ (EFC60000₁₆).

Пример ввода:

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

Пример работы программы:

```
D:\тгпсч>prng.exe -g mt -i 100,1313 -n 10000 -f mt.dat
Генерация выполнена на 0%
Генерация выполнена на 25%
Генерация выполнена на 50%
Генерация выполнена на 75%
Генерация выполнена на 100%
```



The screenshot shows a Notepad window titled "nfsr.dat – Блокнот". The menu bar includes "Файл", "Правка", "Формат", "Вид", and "Справка". The text area contains a single line of 10,000 random numbers generated by the prng.exe program. The numbers are displayed in a single line, wrapping around the width of the window. The status bar at the bottom indicates "Стр 1, стлб 1", "100%", "Windows (CRLF)", and "UTF-8".

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

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) $t = (S_i + S_j) \bmod 256$;

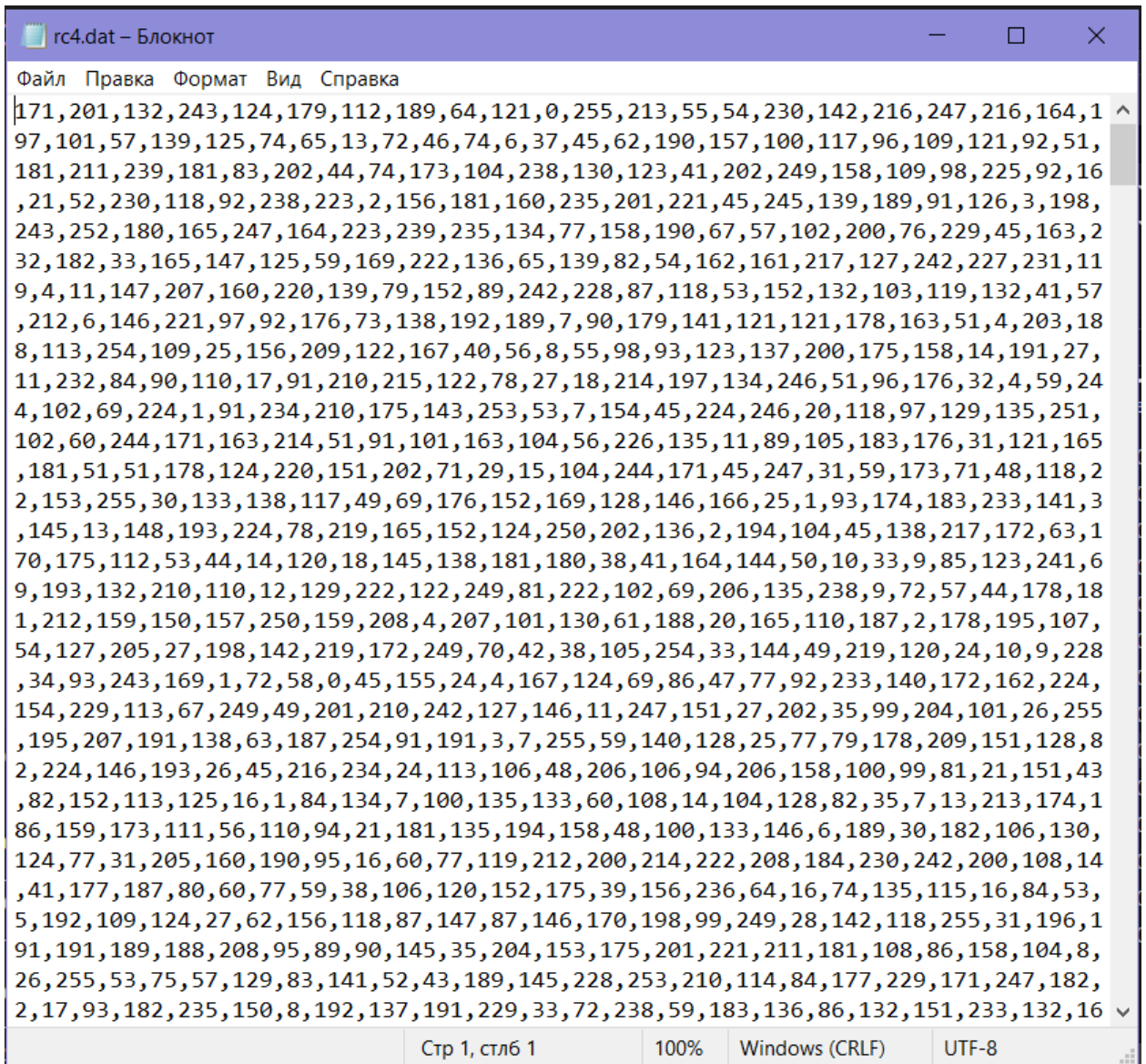
e) $K = S_t$;

Пример ввода:

```
prng.exe                -g                                rc4                -i
40,41,42,43,44,45,46,47,48,49,50,51,52,53,54,55,56,57,58,59,60,61,62,63,64,65,6
6,67,68,69,70,71,72,73,74,75,76,77,78,79,80,81,82,83,84,85,86,87,88,89,90,91,92,
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:

```
D:\тгпсч>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
Генерация выполнена на 0%
Генерация выполнена на 25%
Генерация выполнена на 50%
Генерация выполнена на 75%
Генерация выполнена на 100%
```



```
rc4.dat - Блокнот
Файл  Правка  Формат  Вид  Справка
171,201,132,243,124,179,112,189,64,121,0,255,213,55,54,230,142,216,247,216,164,1
97,101,57,139,125,74,65,13,72,46,74,6,37,45,62,190,157,100,117,96,109,121,92,51,
181,211,239,181,83,202,44,74,173,104,238,130,123,41,202,249,158,109,98,225,92,16
,21,52,230,118,92,238,223,2,156,181,160,235,201,221,45,245,139,189,91,126,3,198,
243,252,180,165,247,164,223,239,235,134,77,158,190,67,57,102,200,76,229,45,163,2
32,182,33,165,147,125,59,169,222,136,65,139,82,54,162,161,217,127,242,227,231,11
9,4,11,147,207,160,220,139,79,152,89,242,228,87,118,53,152,132,103,119,132,41,57
,212,6,146,221,97,92,176,73,138,192,189,7,90,179,141,121,121,178,163,51,4,203,18
8,113,254,109,25,156,209,122,167,40,56,8,55,98,93,123,137,200,175,158,14,191,27,
11,232,84,90,110,17,91,210,215,122,78,27,18,214,197,134,246,51,96,176,32,4,59,24
4,102,69,224,1,91,234,210,175,143,253,53,7,154,45,224,246,20,118,97,129,135,251,
102,60,244,171,163,214,51,91,101,163,104,56,226,135,11,89,105,183,176,31,121,165
,181,51,51,178,124,220,151,202,71,29,15,104,244,171,45,247,31,59,173,71,48,118,2
2,153,255,30,133,138,117,49,69,176,152,169,128,146,166,25,1,93,174,183,233,141,3
,145,13,148,193,224,78,219,165,152,124,250,202,136,2,194,104,45,138,217,172,63,1
70,175,112,53,44,14,120,18,145,138,181,180,38,41,164,144,50,10,33,9,85,123,241,6
9,193,132,210,110,12,129,222,122,249,81,222,102,69,206,135,238,9,72,57,44,178,18
1,212,159,150,157,250,159,208,4,207,101,130,61,188,20,165,110,187,2,178,195,107,
54,127,205,27,198,142,219,172,249,70,42,38,105,254,33,144,49,219,120,24,10,9,228
,34,93,243,169,1,72,58,0,45,155,24,4,167,124,69,86,47,77,92,233,140,172,162,224,
154,229,113,67,249,49,201,210,242,127,146,11,247,151,27,202,35,99,204,101,26,255
,195,207,191,138,63,187,254,91,191,3,7,255,59,140,128,25,77,79,178,209,151,128,8
2,224,146,193,26,45,216,234,24,113,106,48,206,106,94,206,158,100,99,81,21,151,43
,82,152,113,125,16,1,84,134,7,100,135,133,60,108,14,104,128,82,35,7,13,213,174,1
86,159,173,111,56,110,94,21,181,135,194,158,48,100,133,146,6,189,30,182,106,130,
124,77,31,205,160,190,95,16,60,77,119,212,200,214,222,208,184,230,242,200,108,14
,41,177,187,80,60,77,59,38,106,120,152,175,39,156,236,64,16,74,135,115,16,84,53,
5,192,109,124,27,62,156,118,87,147,87,146,170,198,99,249,28,142,118,255,31,196,1
91,191,189,188,208,95,89,90,145,35,204,153,175,201,221,211,181,108,86,158,104,8,
26,255,53,75,57,129,83,141,52,43,189,145,228,253,210,114,84,177,229,171,247,182,
2,17,93,182,235,150,8,192,137,191,229,33,72,238,59,183,136,86,132,151,233,132,16
```

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 .

Пример ввода:

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

Пример работы программы:

```
D:\тгпсч>prng.exe -g rsa -i 12709189,53,10,245 -n 10000 -f rsa.dat
Генерация выполнена на 0%
Генерация выполнена на 25%
Генерация выполнена на 50%
Генерация выполнена на 75%
Генерация выполнена на 100%
```

```
rsa.dat – Блокнот
Файл  Правка  Формат  Вид  Справка
245,249,181,303,319,1006,35,658,377,161,639,409,59,330,472,159,553,693,579,778,1
91,876,816,203,710,191,907,458,650,329,948,856,140,949,274,89,593,70,116,509,386
,564,186,572,984,696,58,870,325,200,554,276,958,278,484,358,999,782,564,678,806,
584,904,498,988,103,534,794,276,55,731,789,971,644,604,500,449,978,325,241,439,6
15,305,472,382,519,439,540,568,900,996,725,189,255,952,142,585,484,646,509,612,2
37,297,864,638,166,726,271,40,767,435,192,814,792,767,557,810,553,295,723,352,56
3,725,72,358,324,280,465,1013,522,208,746,243,866,736,235,409,276,802,169,83,761
,89,913,411,927,58,210,667,154,291,545,971,880,414,91,105,80,222,879,87,814,530,
369,977,775,841,276,965,734,413,197,865,506,29,734,114,227,531,914,852,756,1023,
736,570,293,914,537,1014,400,949,167,386,504,666,857,60,162,1021,716,771,187,98,
1022,183,170,165,158,845,386,206,852,289,409,273,97,839,982,40,834,936,975,394,8
96,941,613,83,136,676,334,996,359,581,623,636,232,842,620,617,142,135,815,449,63
2,364,420,320,891,444,351,186,73,455,839,31,293,83,790,889,628,791,389,1000,118,
888,456,910,79,587,338,979,1022,898,233,151,586,103,985,579,724,669,521,994,619,
356,240,651,1014,819,12,748,395,1016,732,680,660,635,309,520,827,337,133,613,68,
391,287,856,163,267,675,829,555,515,694,404,332,546,657,315,913,414,278,446,496,
931,298,434,420,568,543,189,774,481,433,657,259,493,753,380,744,293,799,284,125,
148,335,91,486,467,93,535,928,475,481,803,568,318,301,331,847,1019,520,932,606,2
96,415,870,270,850,630,39,906,429,400,962,559,987,204,50,945,559,994,882,674,594
,493,214,35,237,324,534,404,273,541,127,352,653,46,655,246,174,14,729,593,306,13
8,581,239,581,633,89,761,963,653,169,713,658,226,124,759,25,901,710,581,13,950,9
65,498,929,151,125,112,500,593,316,365,921,844,374,95,641,877,903,142,225,249,18
1,303,319,1006,35,658,377,161,639,409,59,330,472,159,553,693,579,778,191,876,816
,203,710,191,907,458,650,329,948,856,140,949,274,89,593,70,116,509,386,564,186,5
72,984,696,58,870,325,200,554,276,958,278,484,358,999,782,564,678,806,584,904,49
8,988,103,534,794,276,55,731,789,971,644,604,500,449,978,325,241,439,615,305,472
,382,519,439,540,568,900,996,725,189,255,952,142,585,484,646,509,612,237,297,864
,638,166,726,271,40,767,435,192,814,792,767,557,810,553,295,723,352,563,725,72,3
58,324,280,465,1013,522,208,746,243,866,736,235,409,276,802,169,83,761,89,913,41
1,927,58,210,667,154,291,545,971,880,414,91,105,80,222,879,87,814,530,369,977,77
5,841,276,965,734,413,197,865,506,29,734,114,227,531,914,852,756,1023,736,570,29
```

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

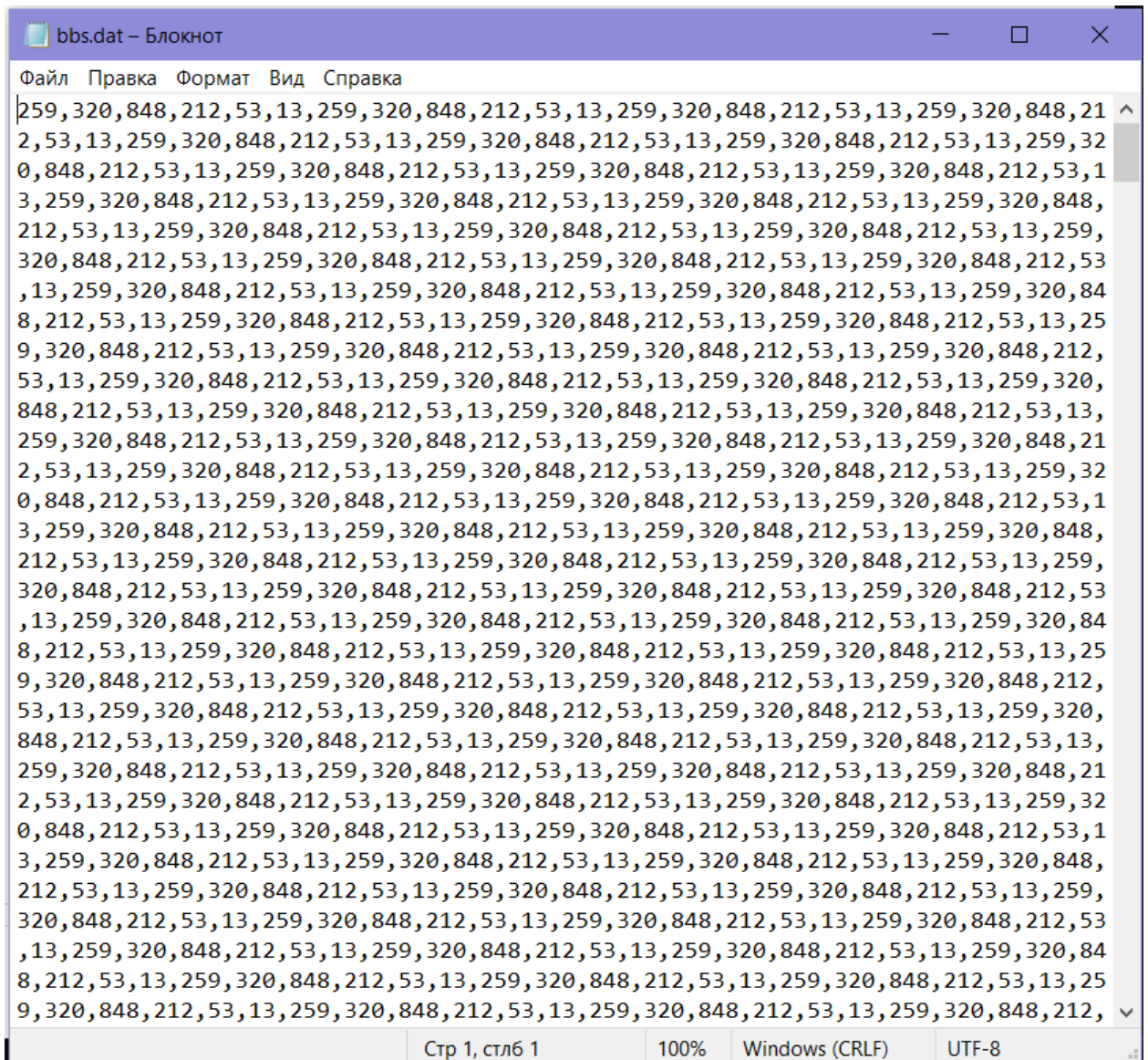
4. Вернуть z_1, z_2, \dots, z_l .

Пример ввода:

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

Пример работы программы:

```
D:\тгпсч>prng.exe -g bbs -i 1562341,10 -n 10000 -f bbs.dat
Генерация выполнена на 0%
Генерация выполнена на 100%
```



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: gnc.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.

```
D:\тгпсч>rnc.exe -h
usage: rnc.py [-h] [-f F] -d {st,tr,ex,nr,gm,ln,ls,bi} -p1 P1 -p2 P2 [-p3 P3]

options:
  -h, --help            show this help message and exit
  -f F                  Имя файла с входной последовательностью.
  -d {st,tr,ex,nr,gm,ln,ls,bi}
                        Код распределения для преобразования последовательности: st – стандартное равномерное с
                        заданным интервалом, tr – треугольное распределение, ex – общее экспоненциальное распределение
  -p1 P1                nr – нормальное распределение, gm – гамма распределение, ln – логнормальное распределение, ls
                        – логистическое распределение, bi – биномиальное распределение
  -p2 P2                1-й параметр, необходимый, для генерации ПСЧ заданного распределения
  -p3 P3                2-й параметр, необходимый, для генерации ПСЧ заданного распределения
                        3-й параметр, необходимый, для генерации ПСЧ гамма-распределением.

D:\тгпсч>
```

2.1 Стандартное равномерное с заданным интервалом

Описание алгоритма:

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

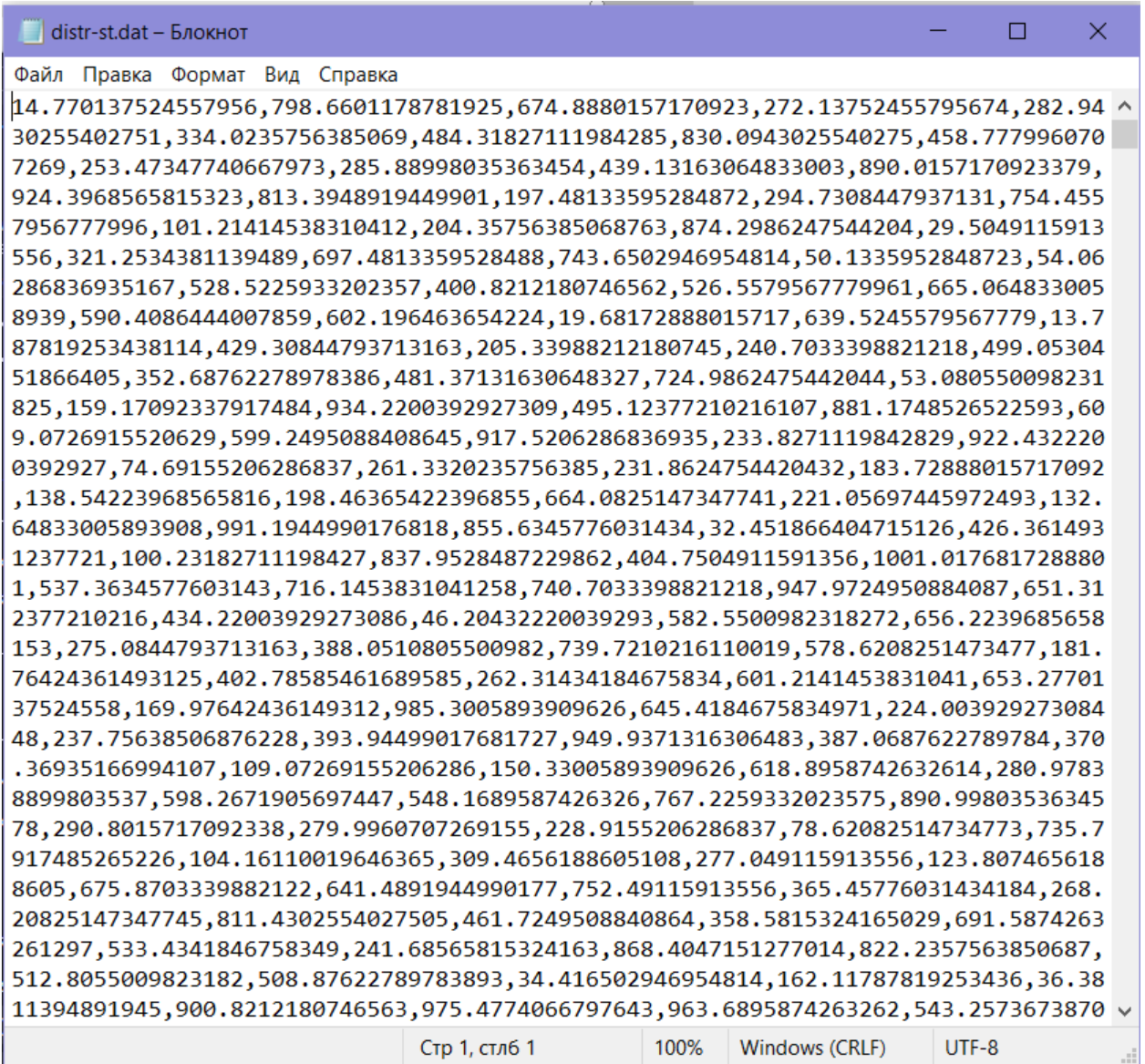
$$Y = bU + a.$$

Пример ввода:

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

Пример работы программы:

```
D:\тгпсч>rnc.exe -f lc.dat -d st -p1 2 -p2 1000
```



```
14.770137524557956,798.6601178781925,674.8880157170923,272.13752455795674,282.94
30255402751,334.0235756385069,484.31827111984285,830.0943025540275,458.777996070
7269,253.47347740667973,285.88998035363454,439.13163064833003,890.0157170923379,
924.3968565815323,813.3948919449901,197.48133595284872,294.7308447937131,754.455
7956777996,101.21414538310412,204.35756385068763,874.2986247544204,29.5049115913
556,321.2534381139489,697.4813359528488,743.6502946954814,50.1335952848723,54.06
286836935167,528.5225933202357,400.8212180746562,526.5579567779961,665.064833005
8939,590.4086444007859,602.196463654224,19.68172888015717,639.5245579567779,13.7
87819253438114,429.30844793713163,205.33988212180745,240.7033398821218,499.05304
51866405,352.68762278978386,481.37131630648327,724.9862475442044,53.080550098231
825,159.17092337917484,934.2200392927309,495.12377210216107,881.1748526522593,60
9.0726915520629,599.2495088408645,917.5206286836935,233.8271119842829,922.432220
0392927,74.69155206286837,261.3320235756385,231.8624754420432,183.72888015717092
,138.54223968565816,198.46365422396855,664.0825147347741,221.05697445972493,132.
64833005893908,991.1944990176818,855.6345776031434,32.451866404715126,426.361493
1237721,100.23182711198427,837.9528487229862,404.7504911591356,1001.017681728880
1,537.3634577603143,716.1453831041258,740.7033398821218,947.9724950884087,651.31
2377210216,434.22003929273086,46.20432220039293,582.5500982318272,656.2239685658
153,275.0844793713163,388.0510805500982,739.7210216110019,578.6208251473477,181.
76424361493125,402.78585461689585,262.31434184675834,601.2141453831041,653.27701
37524558,169.97642436149312,985.3005893909626,645.4184675834971,224.003929273084
48,237.75638506876228,393.94499017681727,949.9371316306483,387.0687622789784,370
.36935166994107,109.07269155206286,150.33005893909626,618.8958742632614,280.9783
8899803537,598.2671905697447,548.1689587426326,767.2259332023575,890.99803536345
78,290.8015717092338,279.9960707269155,228.9155206286837,78.62082514734773,735.7
917485265226,104.16110019646365,309.4656188605108,277.049115913556,123.807465618
8605,675.8703339882122,641.4891944990177,752.49115913556,365.45776031434184,268.
20825147347745,811.4302554027505,461.7249508840864,358.5815324165029,691.5874263
261297,533.4341846758349,241.68565815324163,868.4047151277014,822.2357563850687,
512.8055009823182,508.87622789783893,34.416502946954814,162.11787819253436,36.38
11394891945,900.8212180746563,975.4774066797643,963.6895874263262,543.2573673870
```

2.2 Треугольное распределение

Описание алгоритма:

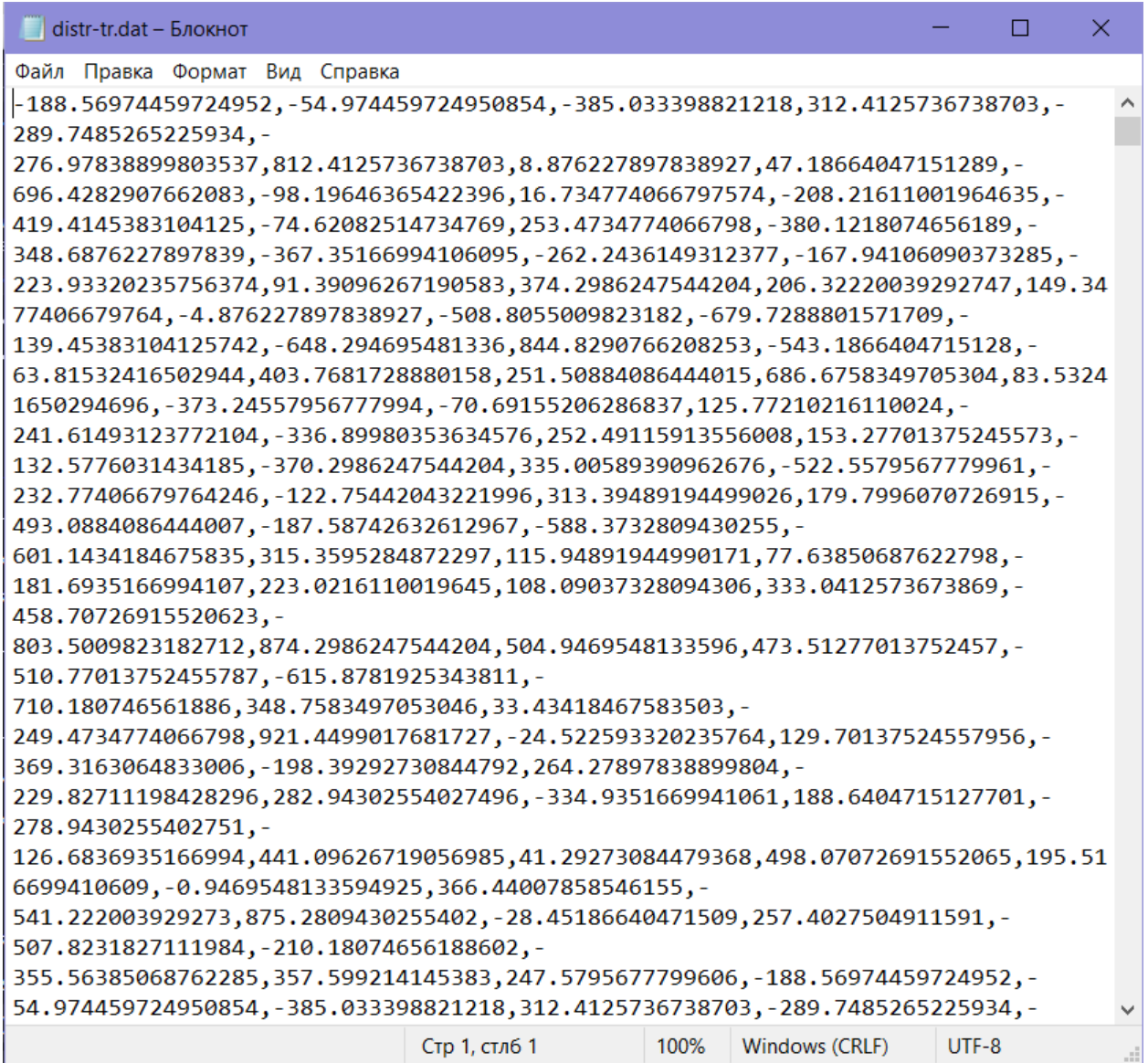
Если стандартные случайные числа U_1 и U_2 независимо получены методом генерации стандартного равномерного числа, то случайное число Y , подчиняющееся треугольному распределению, определяют по формуле $Y = a + b(U_1 + U_2 - 1)$.

Пример ввода:

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

Пример работы программы:

```
D:\тгпсч>rnc.exe -f lc.dat -d tr -p1 2 -p2 1000
```



```
Файл  Правка  Формат  Вид  Справка
|-188.56974459724952, -54.974459724950854, -385.033398821218, 312.4125736738703, -
289.7485265225934, -
276.97838899803537, 812.4125736738703, 8.876227897838927, 47.18664047151289, -
696.4282907662083, -98.19646365422396, 16.734774066797574, -208.21611001964635, -
419.4145383104125, -74.62082514734769, 253.4734774066798, -380.1218074656189, -
348.6876227897839, -367.35166994106095, -262.2436149312377, -167.94106090373285, -
223.93320235756374, 91.39096267190583, 374.2986247544204, 206.32220039292747, 149.34
77406679764, -4.876227897838927, -508.8055009823182, -679.7288801571709, -
139.45383104125742, -648.294695481336, 844.8290766208253, -543.1866404715128, -
63.81532416502944, 403.7681728880158, 251.50884086444015, 686.6758349705304, 83.5324
1650294696, -373.24557956777994, -70.69155206286837, 125.77210216110024, -
241.61493123772104, -336.89980353634576, 252.49115913556008, 153.27701375245573, -
132.5776031434185, -370.2986247544204, 335.00589390962676, -522.5579567779961, -
232.77406679764246, -122.75442043221996, 313.39489194499026, 179.7996070726915, -
493.0884086444007, -187.58742632612967, -588.3732809430255, -
601.1434184675835, 315.3595284872297, 115.94891944990171, 77.63850687622798, -
181.6935166994107, 223.0216110019645, 108.09037328094306, 333.0412573673869, -
458.70726915520623, -
803.5009823182712, 874.2986247544204, 504.9469548133596, 473.51277013752457, -
510.77013752455787, -615.8781925343811, -
710.180746561886, 348.7583497053046, 33.43418467583503, -
249.4734774066798, 921.4499017681727, -24.522593320235764, 129.70137524557956, -
369.3163064833006, -198.39292730844792, 264.27897838899804, -
229.82711198428296, 282.94302554027496, -334.9351669941061, 188.6404715127701, -
278.9430255402751, -
126.6836935166994, 441.09626719056985, 41.29273084479368, 498.07072691552065, 195.51
6699410609, -0.9469548133594925, 366.44007858546155, -
541.222003929273, 875.2809430255402, -28.45186640471509, 257.4027504911591, -
507.8231827111984, -210.18074656188602, -
355.56385068762285, 357.599214145383, 247.5795677799606, -188.56974459724952, -
54.974459724950854, -385.033398821218, 312.4125736738703, -289.7485265225934, -
```

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

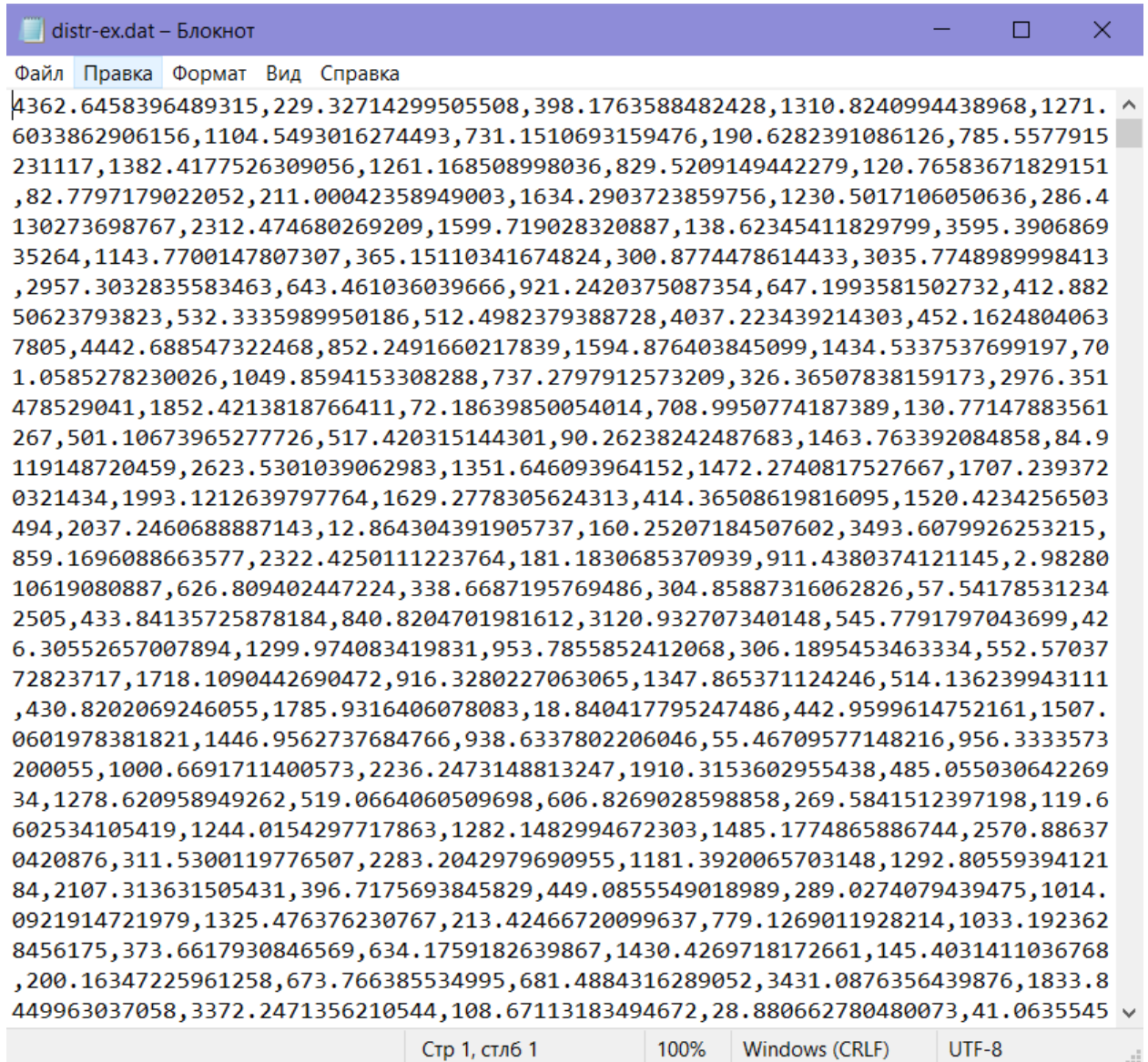
$$Y = -b \ln(U) + a.$$

Пример ввода:

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

Пример работы программы:

```
D:\тгпсч>rnc.exe -f lc.dat -d ex -p1 2 -p2 1000
```



Файл Правка Формат Вид Справка

4362.6458396489315,229.32714299505508,398.1763588482428,1310.8240994438968,1271.6033862906156,1104.5493016274493,731.1510693159476,190.6282391086126,785.5577915231117,1382.4177526309056,1261.168508998036,829.5209149442279,120.76583671829151,82.7797179022052,211.00042358949003,1634.2903723859756,1230.5017106050636,286.4130273698767,2312.474680269209,1599.719028320887,138.62345411829799,3595.390686935264,1143.7700147807307,365.15110341674824,300.8774478614433,3035.7748989998413,2957.3032835583463,643.461036039666,921.2420375087354,647.1993581502732,412.88250623793823,532.3335989950186,512.4982379388728,4037.223439214303,452.16248040637805,4442.688547322468,852.2491660217839,1594.876403845099,1434.5337537699197,701.0585278230026,1049.8594153308288,737.2797912573209,326.36507838159173,2976.351478529041,1852.4213818766411,72.18639850054014,708.9950774187389,130.77147883561267,501.10673965277726,517.420315144301,90.26238242487683,1463.763392084858,84.9119148720459,2623.5301039062983,1351.646093964152,1472.2740817527667,1707.2393720321434,1993.1212639797764,1629.2778305624313,414.36508619816095,1520.4234256503494,2037.2460688887143,12.864304391905737,160.25207184507602,3493.6079926253215,859.1696088663577,2322.4250111223764,181.1830685370939,911.4380374121145,2.9828010619080887,626.809402447224,338.6687195769486,304.85887316062826,57.541785312342505,433.84135725878184,840.8204701981612,3120.932707340148,545.7791797043699,426.30552657007894,1299.974083419831,953.7855852412068,306.1895453463334,552.5703772823717,1718.1090442690472,916.3280227063065,1347.865371124246,514.136239943111,430.8202069246055,1785.9316406078083,18.840417795247486,442.9599614752161,1507.0601978381821,1446.9562737684766,938.6337802206046,55.46709577148216,956.3333573200055,1000.6691711400573,2236.2473148813247,1910.3153602955438,485.05503064226934,1278.620958949262,519.0664060509698,606.8269028598858,269.5841512397198,119.6602534105419,1244.0154297717863,1282.1482994672303,1485.1774865886744,2570.886370420876,311.5300119776507,2283.2042979690955,1181.3920065703148,1292.8055939412184,2107.313631505431,396.7175693845829,449.0855549018989,289.0274079439475,1014.0921914721979,1325.476376230767,213.42466720099637,779.1269011928214,1033.1923628456175,373.6617930846569,634.1759182639867,1430.4269718172661,145.4031411036768,200.16347225961258,673.766385534995,681.4884316289052,3431.0876356439876,1833.8449963037058,3372.2471356210544,108.67113183494672,28.880662780480073,41.0635545

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

2.4 Нормальное распределение

Описание алгоритма:

Если стандартные равномерные случайные числа U_1 и U_2 независимо сгенерированы методом, установленным в разделе 2, то два независимых нормальных случайных числа Z_1, Z_2 получают в соответствии со следующей процедурой

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

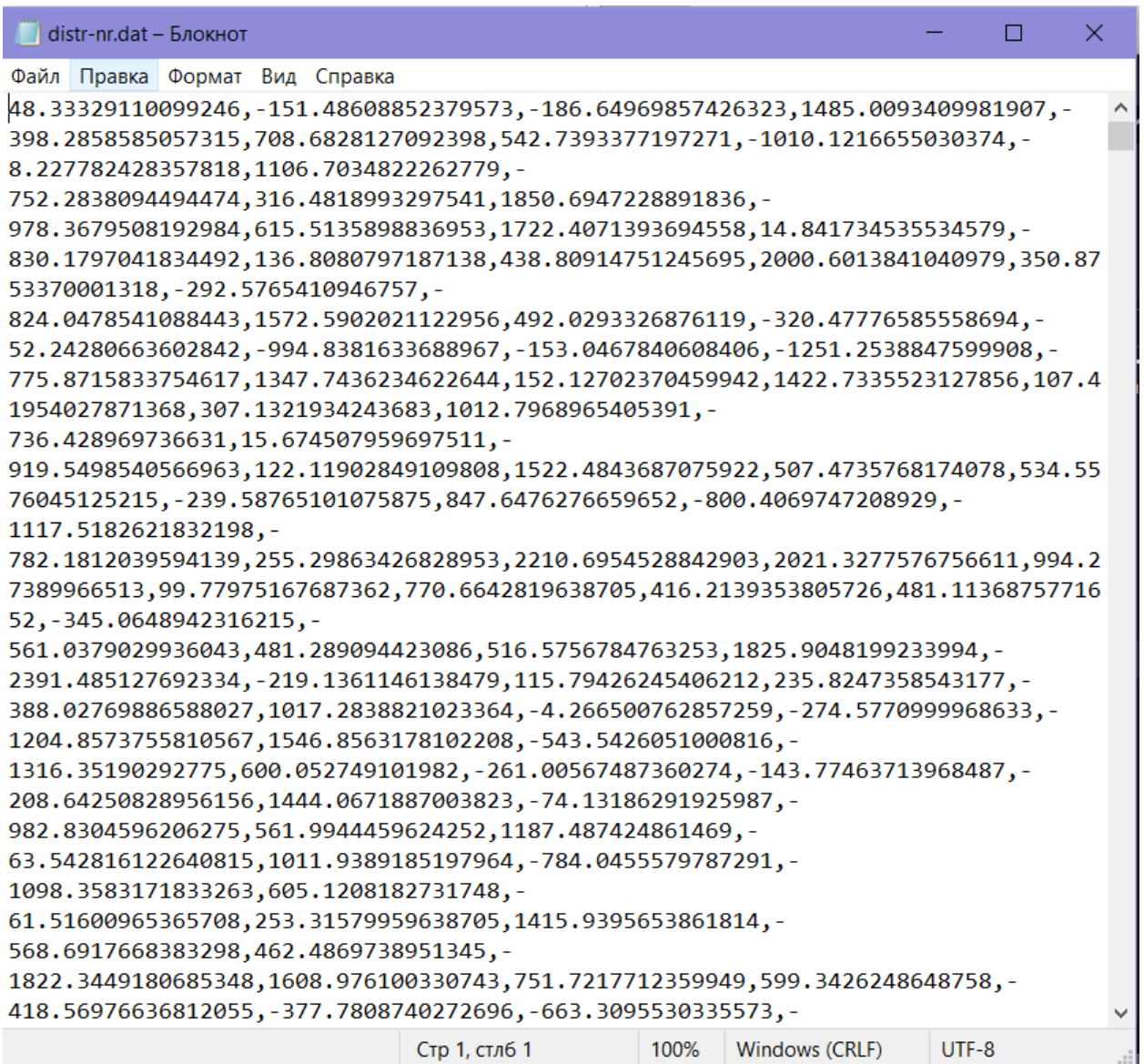
$$Z_2 = \mu + \sigma \sqrt{-2 \ln(1 - U_1)} \sin(2\pi U_2).$$

Пример ввода:

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

Пример работы программы:

```
D:\тгпсч>rnc.exe -f lc.dat -d nr -p1 2 -p2 1000
```



The screenshot shows a Notepad window with the title "distr-nr.dat - Блокнот". The menu bar includes "Файл", "Правка", "Формат", "Вид", and "Справка". The text area contains a long list of numerical data points, each preceded by a minus sign. The status bar at the bottom indicates "Стр 1, столб 1", "100%", "Windows (CRLF)", and "UTF-8".

```

-48.33329110099246, -151.48608852379573, -186.64969857426323, 1485.0093409981907, -
-398.2858585057315, 708.6828127092398, 542.7393377197271, -1010.1216655030374, -
8.227782428357818, 1106.7034822262779, -
752.2838094494474, 316.4818993297541, 1850.6947228891836, -
978.3679508192984, 615.5135898836953, 1722.4071393694558, 14.841734535534579, -
830.1797041834492, 136.8080797187138, 438.80914751245695, 2000.6013841040979, 350.87
53370001318, -292.5765410946757, -
824.0478541088443, 1572.5902021122956, 492.0293326876119, -320.47776585558694, -
52.24280663602842, -994.8381633688967, -153.0467840608406, -1251.2538847599908, -
775.8715833754617, 1347.7436234622644, 152.12702370459942, 1422.7335523127856, 107.4
1954027871368, 307.1321934243683, 1012.7968965405391, -
736.428969736631, 15.674507959697511, -
919.5498540566963, 122.11902849109808, 1522.4843687075922, 507.4735768174078, 534.55
76045125215, -239.58765101075875, 847.6476276659652, -800.4069747208929, -
1117.5182621832198, -
782.1812039594139, 255.29863426828953, 2210.6954528842903, 2021.3277576756611, 994.2
7389966513, 99.77975167687362, 770.6642819638705, 416.2139353805726, 481.11368757716
52, -345.0648942316215, -
561.0379029936043, 481.289094423086, 516.5756784763253, 1825.9048199233994, -
2391.485127692334, -219.1361146138479, 115.79426245406212, 235.8247358543177, -
388.02769886588027, 1017.2838821023364, -4.266500762857259, -274.5770999968633, -
1204.8573755810567, 1546.8563178102208, -543.5426051000816, -
1316.35190292775, 600.052749101982, -261.00567487360274, -143.77463713968487, -
208.64250828956156, 1444.0671887003823, -74.13186291925987, -
982.8304596206275, 561.9944459624252, 1187.487424861469, -
63.542816122640815, 1011.9389185197964, -784.0455579787291, -
1098.3583171833263, 605.1208182731748, -
61.51600965365708, 253.31579959638705, 1415.9395653861814, -
568.6917668383298, 462.4869738951345, -
1822.3449180685348, 1608.976100330743, 751.7217712359949, 599.3426248648758, -
418.56976636812055, -377.7808740272696, -663.3095530335573, -

```

2.5 Гамма распределение (для параметра c=k)

Описание алгоритма:

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

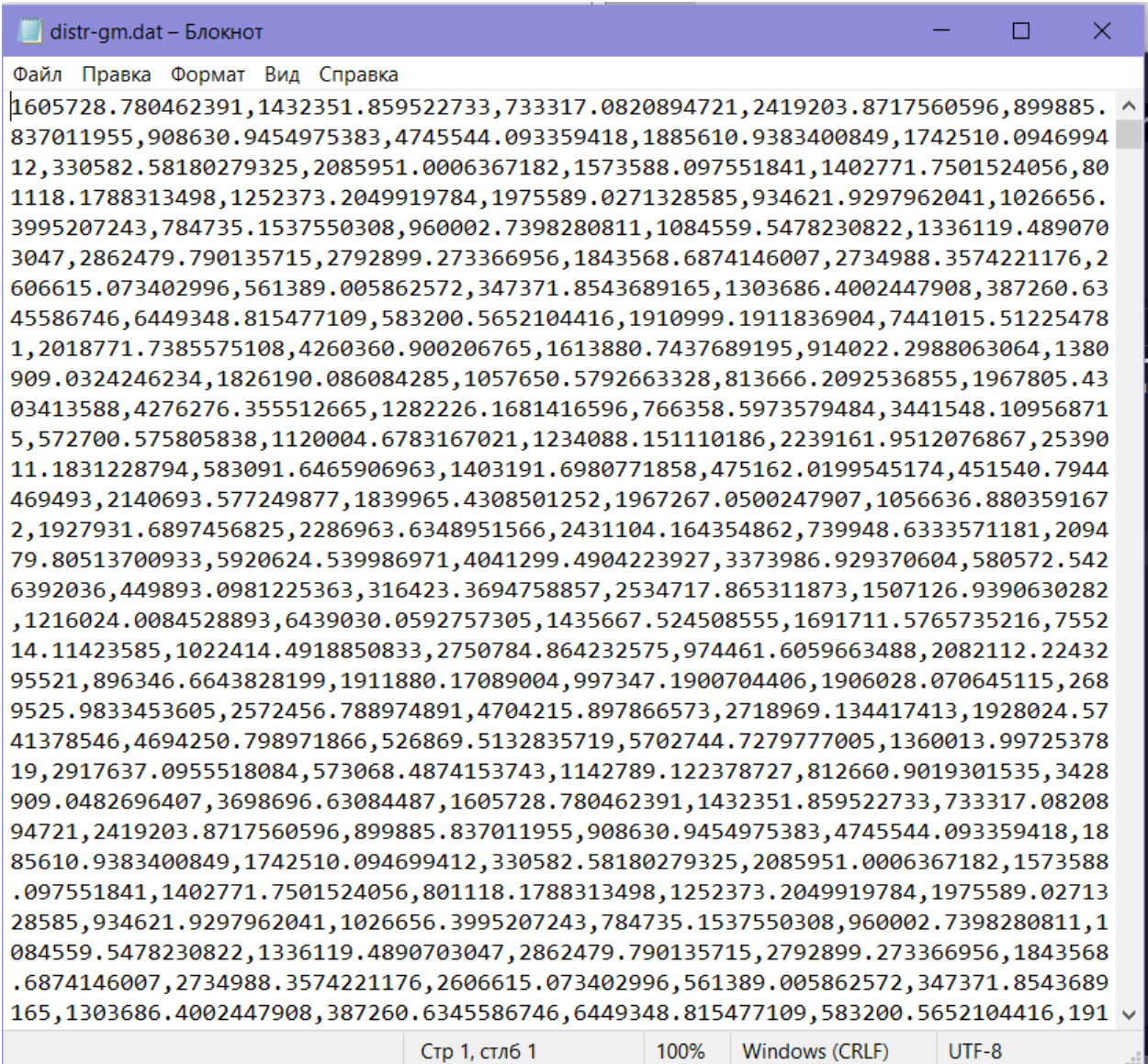
$$Y = a - b * \ln\{(1 - U_1)(1 - U_2) \dots (1 - U_k)\}$$

Пример ввода:

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

Пример работы программы:

```
D:\тгпсч>rnc.exe -f lc.dat -d gm -p1 0 -p2 1000000 -p3 2
```



The screenshot shows a Notepad window with the title "distr-gm.dat - Блокнот". The menu bar includes "Файл", "Правка", "Формат", "Вид", and "Справка". The text area contains a single line of 1000 random numbers, separated by commas, arranged in 10 columns. The numbers are displayed in a monospaced font. At the bottom of the window, the status bar shows "Стр 1, столб 1", "100%", "Windows (CRLF)", and "UTF-8".

2.6 Логнормальное распределение

Описание алгоритма:

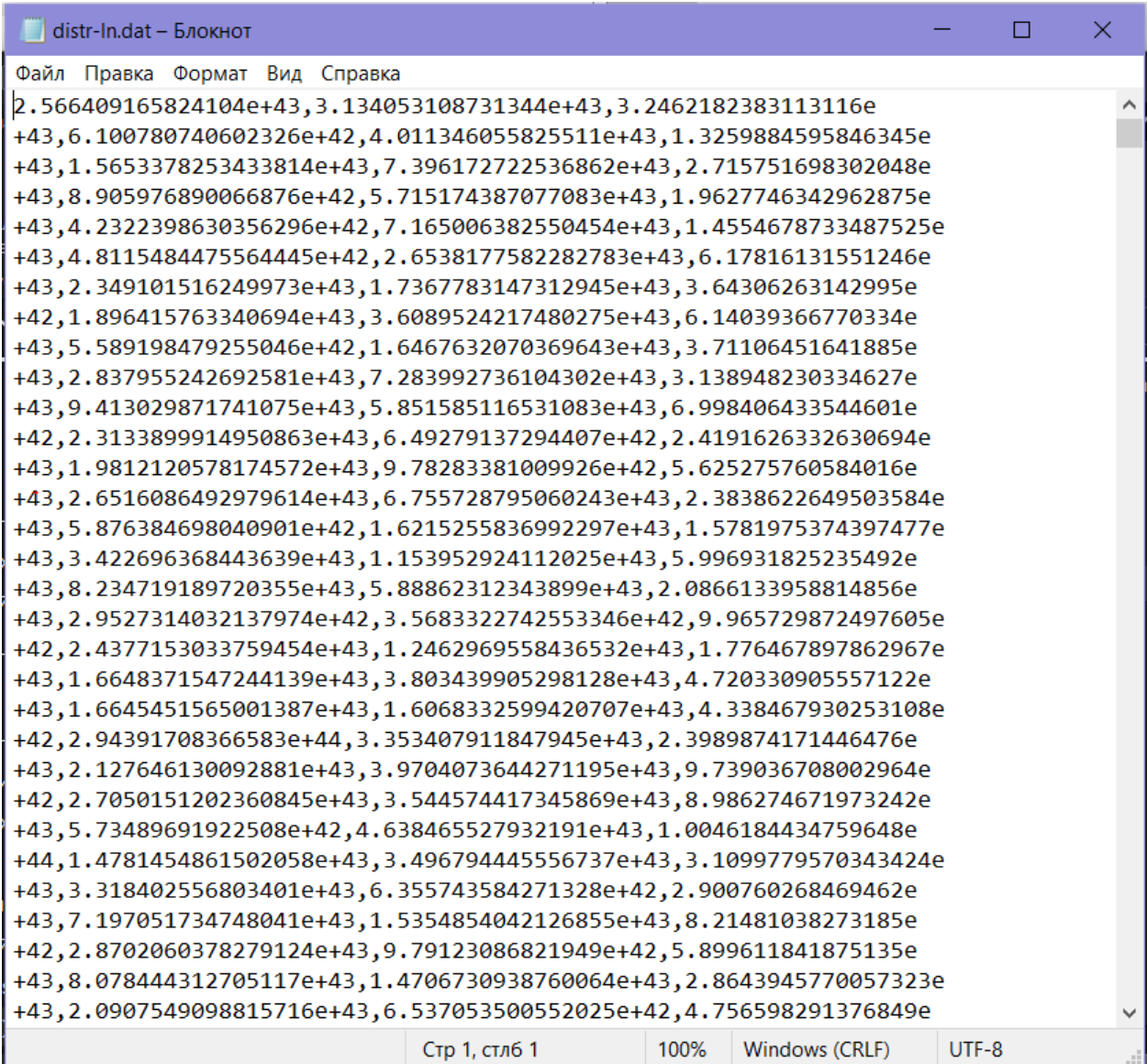
Используя стандартные нормальные случайные числа Z , применяют формулу $Y = a + \exp(b - Z)$ для получения случайных чисел, соответствующих логнормальному распределению.

Пример ввода:

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

Пример работы программы:

```
D:\тгпсч>rnc.exe -f lc.dat -d ln -p1 2 -p2 100
```



```
distr-ln.dat - Блокнот
Файл  Правка  Формат  Вид  Справка
2.566409165824104e+43,3.134053108731344e+43,3.2462182383113116e
+43,6.100780740602326e+42,4.011346055825511e+43,1.3259884595846345e
+43,1.5653378253433814e+43,7.396172722536862e+43,2.715751698302048e
+43,8.905976890066876e+42,5.715174387077083e+43,1.9627746342962875e
+43,4.2322398630356296e+42,7.165006382550454e+43,1.4554678733487525e
+43,4.8115484475564445e+42,2.6538177582282783e+43,6.17816131551246e
+43,2.349101516249973e+43,1.7367783147312945e+43,3.64306263142995e
+42,1.896415763340694e+43,3.6089524217480275e+43,6.14039366770334e
+43,5.589198479255046e+42,1.6467632070369643e+43,3.71106451641885e
+43,2.837955242692581e+43,7.283992736104302e+43,3.138948230334627e
+43,9.413029871741075e+43,5.851585116531083e+43,6.998406433544601e
+42,2.3133899914950863e+43,6.49279137294407e+42,2.4191626332630694e
+43,1.9812120578174572e+43,9.78283381009926e+42,5.625275760584016e
+43,2.6516086492979614e+43,6.755728795060243e+43,2.3838622649503584e
+43,5.876384698040901e+42,1.6215255836992297e+43,1.5781975374397477e
+43,3.422696368443639e+43,1.153952924112025e+43,5.996931825235492e
+43,8.234719189720355e+43,5.88862312343899e+43,2.0866133958814856e
+43,2.9527314032137974e+42,3.5683322742553346e+42,9.965729872497605e
+42,2.4377153033759454e+43,1.2462969558436532e+43,1.776467897862967e
+43,1.6648371547244139e+43,3.803439905298128e+43,4.720330905557122e
+43,1.6645451565001387e+43,1.6068332599420707e+43,4.338467930253108e
+42,2.94391708366583e+44,3.353407911847945e+43,2.3989874171446476e
+43,2.127646130092881e+43,3.9704073644271195e+43,9.739036708002964e
+42,2.7050151202360845e+43,3.544574417345869e+43,8.986274671973242e
+43,5.73489691922508e+42,4.638465527932191e+43,1.0046184434759648e
+44,1.4781454861502058e+43,3.496794445556737e+43,3.1099779570343424e
+43,3.318402556803401e+43,6.355743584271328e+42,2.900760268469462e
+43,7.197051734748041e+43,1.5354854042126855e+43,8.21481038273185e
+42,2.8702060378279124e+43,9.79123086821949e+42,5.899611841875135e
+43,8.078444312705117e+43,1.4706730938760064e+43,2.8643945770057323e
+43,2.0907549098815716e+43,6.537053500552025e+42,4.756598291376849e
```

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

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

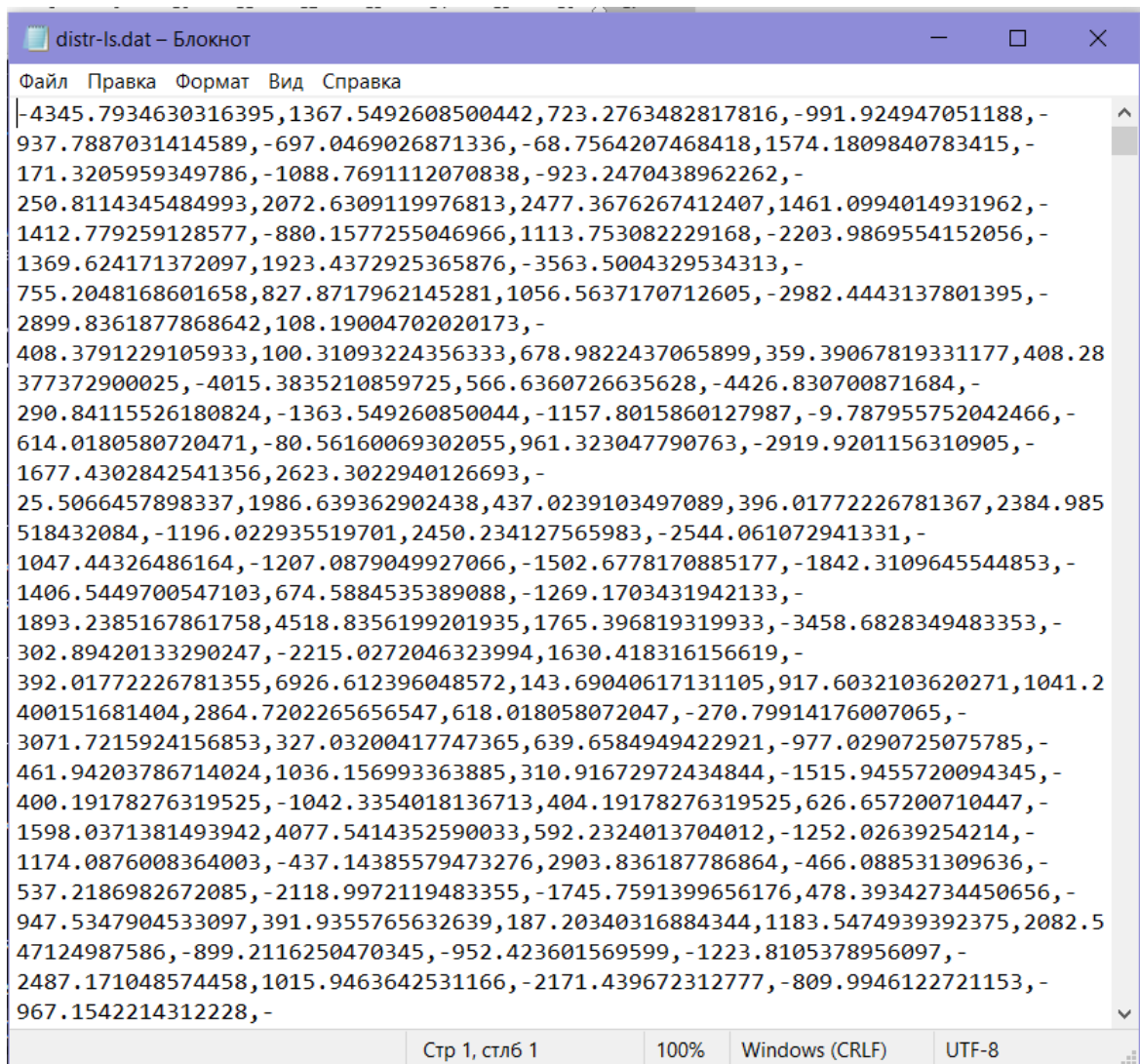
$$Y = a + b \ln\left(\frac{U}{1-U}\right).$$

Пример ввода:

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

Пример работы программы:

```
D:\тгпсч>rnc.exe -f lc.dat -d ls -p1 2 -p2 1000
```



```
distr-ls.dat - Блокнот
Файл  Правка  Формат  Вид  Справка
-4345.7934630316395,1367.5492608500442,723.2763482817816,-991.924947051188,-
937.7887031414589,-697.0469026871336,-68.7564207468418,1574.1809840783415,-
171.3205959349786,-1088.7691112070838,-923.2470438962262,-
250.8114345484993,2072.6309119976813,2477.3676267412407,1461.0994014931962,-
1412.779259128577,-880.1577255046966,1113.753082229168,-2203.9869554152056,-
1369.624171372097,1923.4372925365876,-3563.5004329534313,-
755.2048168601658,827.8717962145281,1056.5637170712605,-2982.4443137801395,-
2899.8361877868642,108.19004702020173,-
408.3791229105933,100.31093224356333,678.9822437065899,359.39067819331177,408.28
377372900025,-4015.3835210859725,566.6360726635628,-4426.830700871684,-
290.84115526180824,-1363.549260850044,-1157.8015860127987,-9.787955752042466,-
614.0180580720471,-80.56160069302055,961.323047790763,-2919.9201156310905,-
1677.4302842541356,2623.3022940126693,-
25.5066457898337,1986.639362902438,437.0239103497089,396.01772226781367,2384.985
518432084,-1196.022935519701,2450.234127565983,-2544.061072941331,-
1047.44326486164,-1207.0879049927066,-1502.6778170885177,-1842.3109645544853,-
1406.5449700547103,674.5884535389088,-1269.1703431942133,-
1893.2385167861758,4518.8356199201935,1765.396819319933,-3458.6828349483353,-
302.89420133290247,-2215.0272046323994,1630.418316156619,-
392.01772226781355,6926.612396048572,143.69040617131105,917.6032103620271,1041.2
400151681404,2864.7202265656547,618.018058072047,-270.79914176007065,-
3071.7215924156853,327.03200417747365,639.6584949422921,-977.0290725075785,-
461.94203786714024,1036.156993363885,310.91672972434844,-1515.9455720094345,-
400.19178276319525,-1042.3354018136713,404.19178276319525,626.657200710447,-
1598.0371381493942,4077.5414352590033,592.2324013704012,-1252.02639254214,-
1174.0876008364003,-437.14385579473276,2903.836187786864,-466.088531309636,-
537.2186982672085,-2118.9972119483355,-1745.7591399656176,478.39342734450656,-
947.5347904533097,391.9355765632639,187.20340316884344,1183.5474939392375,2082.5
47124987586,-899.2116250470345,-952.423601569599,-1223.8105378956097,-
2487.171048574458,1015.9463642531166,-2171.439672312777,-809.9946122721153,-
967.1542214312228,-
```

2.8 Биномиальное распределение

Описание алгоритма:

Вычисляют функцию распределения

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

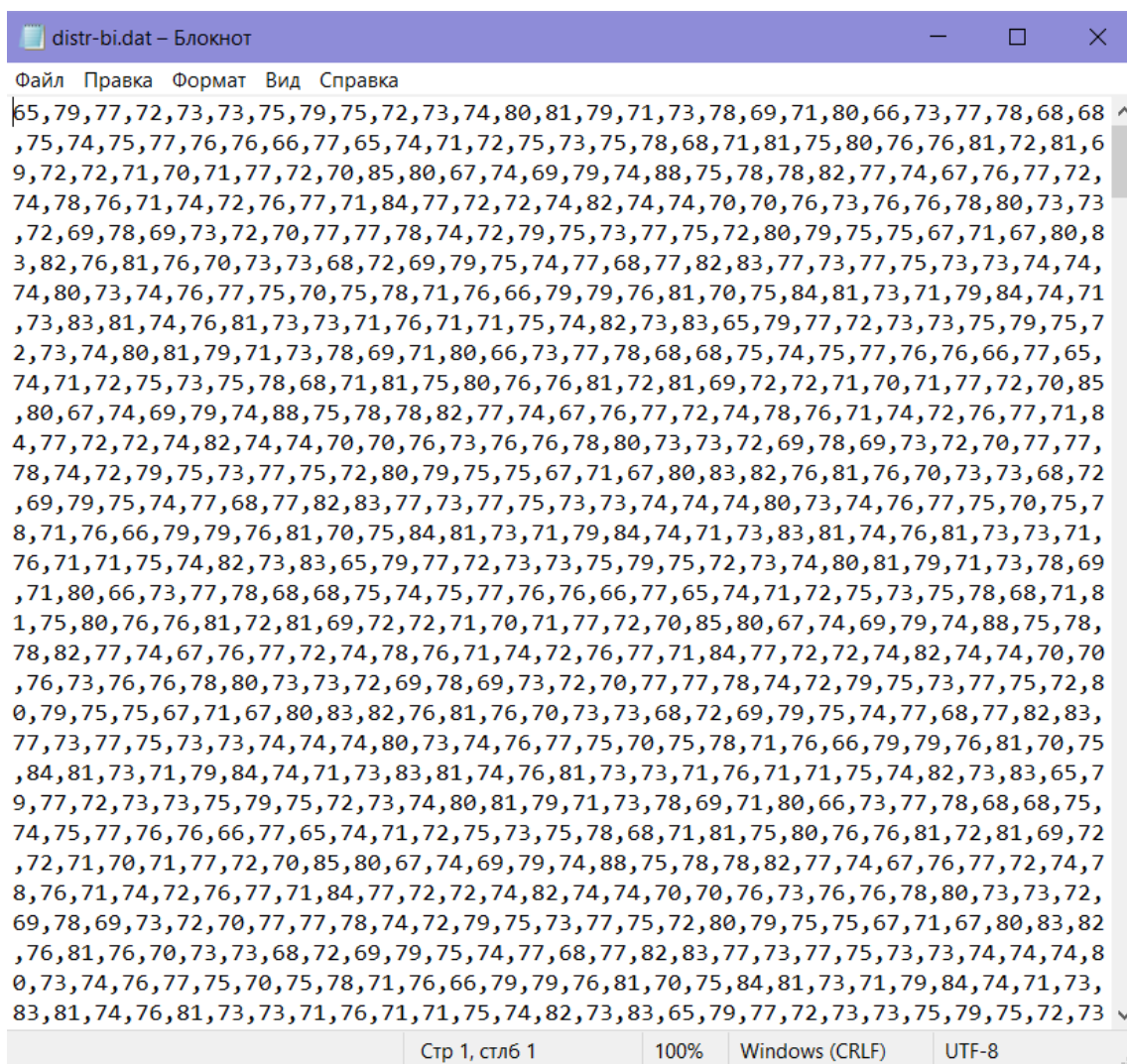
Для получения случайного числа Y генерируют стандартное равномерное случайное число U . Случайное число Y является наименьшим значением y , для которого $U \leq F(y)$.

Пример ввода:

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

Пример работы программы:

```
D:\тггсч>rnc.exe -f lc.dat -d bi -p1 0.75 -p2 100
```



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
,670,8,907,951,685,989,222,931,169,286,289,556,731,902,688,701,771,533,990,630,70
8,884,255,683,25,214,792,348,34,758,9,781,946,580,615,955,585,5,886,563,81,38,809
,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
,25,642,286,489,478,184,812,770,846,241,141,266,500,375,827,633,761,154,663,461,2
06,529,212,667,342,360,165,523,749,582,803,553,345,786,990,361,702,256,380,234,23
8,73,965,266,300,847,755,969,681,146,843,125,306,845,752,879,458,788,833,727,817,
122,239,765,877,827,327,733,658,644,880,150,474,493,689,670,368,611,263,113,417,8
34,103,725,754,117,824,623,338,540,337,879,521,183,370,808,120,571,871,301,210,79
6,744,398,106,845,745,842,876,399,27,105,601,802,831,53,266,157,352,175,303,505,4
84,994,425,292,729,654,584,860,420,412,49,281,417,703,400,48,404,772,389,733,152,
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:
 $1 < e < (p - 1) * (q - 1)$ ,  $\text{НОД}(e, (p - 1) * (q - 1)) = 1$ ,  $p * q = n$ , 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\nраспределения"
    helpp2 = "2-й параметр, необходимый, для генерации ПСЧ заданного\nраспределения"
    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))

```