МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

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

Кафедра теоретических основ компьютерной безопасности и криптографии

**ТЕОРИЯ ПСЕВДОСЛУЧАЙНЫХ ГЕНЕРАТОРОВ**

ОТЧЁТ ПО ПРАКТИЧЕСКОМУ КУРСУ

студента 4 курса 431 группы

специальности 10.05.01 Компьютерная безопасность

факультета компьютерных наук и информационных технологий

Алексеева Александра Александровича

|  |  |  |
| --- | --- | --- |
| Преподаватель  Ассистент | \_\_\_\_\_\_\_\_\_\_ | Н.А. Артемова |
|  | подпись, дата |  |

Саратов 2023

**Задание 1. Генерация псевдослучайных чисел**

***Описание задания***: создать программу, генерирующую псевдослучайные числа из заданного диапазона. Входные параметры алгоритмы передаются программе через строку параметров. Выходные значения записываются в файл, указанный в параметре запуска программы.

Алгоритм 1. Линейный конгруэнтный метод

**Описание алгоритма**

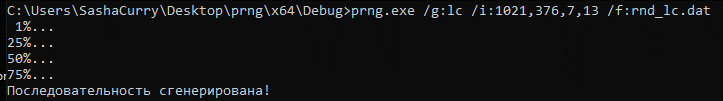
Последовательность ПСЧ получаемая по формуле:

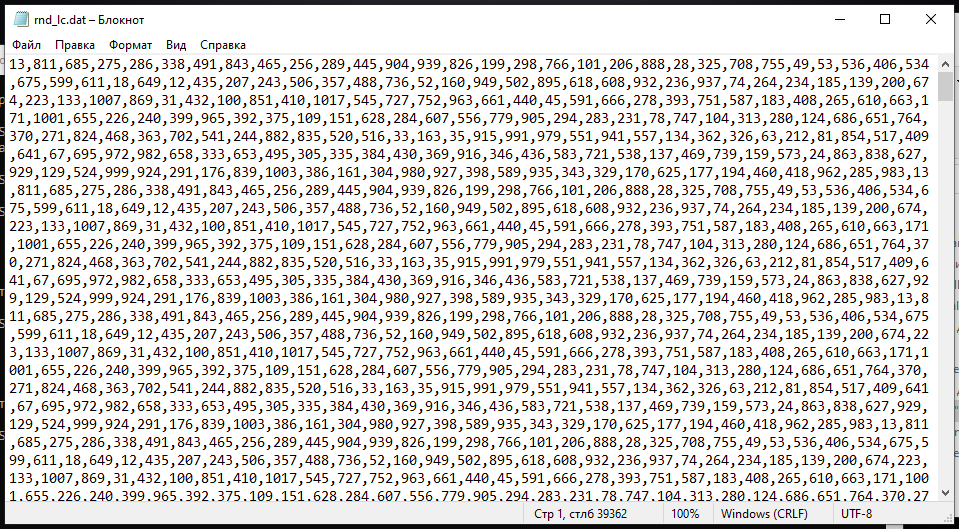
называется линейной конгруэнтной последовательностью. Ключом служит .

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

**Параметры запуска программы**

prng.exe /g:lc /i:1021,376,7,13 /f:rnd\_lc.dat





**Исходный текст программы**

void algLC(vector <int> init, int count, string outFile) {

ofstream fout;

fout.open(outFile);

long long x0 = rand() % 1000, a = 1103515245, c = 12345, m = 2147483647, mod = 1000;

if (!init.empty())

m = init[0], a = init[1], c = init[2], x0 = init[3], mod = init[0];

if (m <= 0 || a < 0 || a > m || c < 0 || c > m || x0 < 0 || x0 > m) { //Проверка на дурака

cout << "Некорректный ввод данных! \n";

return;

}

fout << x0 << endl;

for (int i = 2; i <= count; i++) {

x0 = (a \* x0 + c) % m;

fout << x0 % mod << endl;

if (i / (float)count \* 100 == 1) cout << " 1%... \n";

if (i / (float)count \* 100 == 25) cout << "25%... \n";

if (i / (float)count \* 100 == 50) cout << "50%... \n";

if (i / (float)count \* 100 == 75) cout << "75%... \n";

if (i / (float)count \* 100 == 100) cout << "Последовательность сгенерирована! \n";

}

fout.close();

}

Алгоритм 2. Аддитивный метод

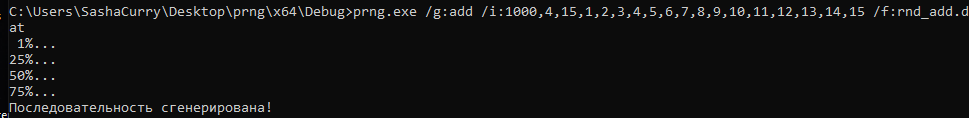
**Описание алгоритма**

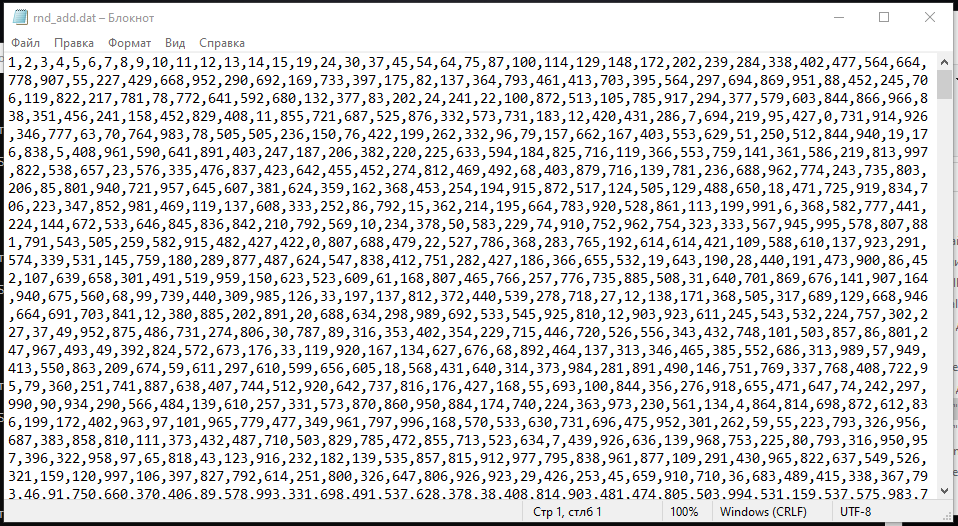
Для получения последовательности ПСЧ необходимо воспользоваться следующей формулой:

где k, j – целые числа, называемые запаздываниями

**Параметры запуска программы**

prng.exe /g:add /i:1000,4,15,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15 /f:rnd\_add.dat





**Исходный текст программы**

void algADD(vector <int> init, int count, string outFile) {

ofstream fout;

fout.open(outFile);

int m, iYoung, iOld; //Инициализируем начальные значения

vector <int> startVal;

if (init.empty()) {

m = 1000, iYoung = 55, iOld = 24;

for (int i = 0; i < 56; i++) {

startVal.push\_back(rand() % 1000);

fout << startVal[i] << endl;

}

}

else {

m = init[0], iYoung = init[1], iOld = init[2];

for (int i = 3; i < init.size(); i++) {

startVal.push\_back(init[i] % m);

fout << startVal[i - 3] << endl;

}

}

for (int i = startVal.size(); i <= count; i++) {

int xNew = (startVal[iOld - 1] + startVal[iYoung - 1]) % m;

fout << xNew << endl;

startVal.push\_back(xNew);

startVal.erase(startVal.begin());

if (i / (float)count \* 100 == 1) cout << " 1%... \n";

if (i / (float)count \* 100 == 25) cout << "25%... \n";

if (i / (float)count \* 100 == 50) cout << "50%... \n";

if (i / (float)count \* 100 == 75) cout << "75%... \n";

if (i / (float)count \* 100 == 100) cout << "Последовательность сгенерирована! \n";

}

fout.close();

}

Алгоритм 3. Пятипараметрический метод

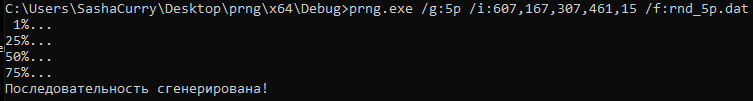
**Описание алгоритма**

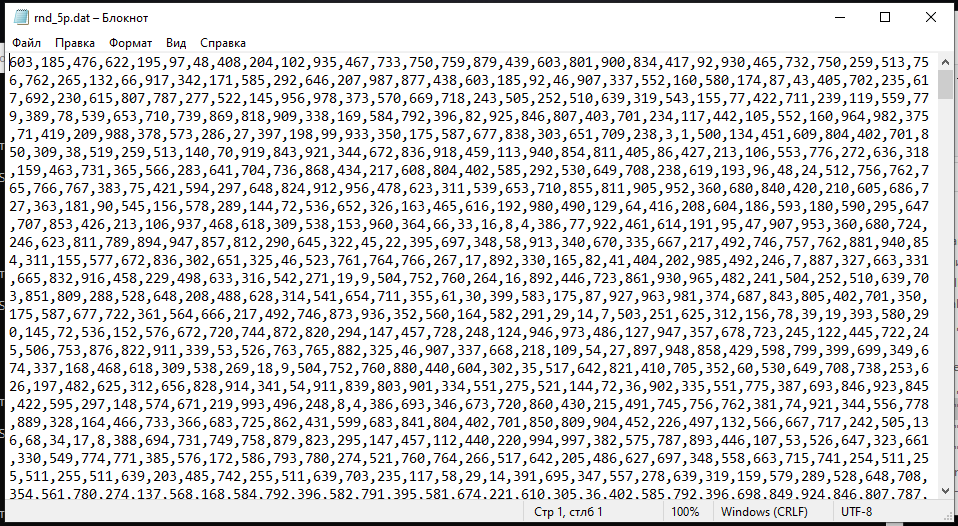
Данный метод является частным случаем РСЛОС, использует характеристический многочлен из 5 членов и позволяет генерировать последовательности *w*-битовых двоичных целых чисел в соответствии со следующей рекуррентной формулой (использует характеристический многочлен из 5 членов: p, q1, q2, q3, w).

После вычисления берётся w битов из регистра p.

**Параметры запуска программы**

prng.exe /g:5p /i:607,167,307,461,15 /f:rnd\_5p.dat

****

****

**Исходный текст программы**

void alg5P(vector <int> init, int count, string outFile) {

ofstream fout;

fout.open(outFile);

int p = 607, q1 = 167, q2 = 307, q3 = 461, w = 15;

if (!init.empty())

p = init[0], q1 = init[1], q2 = init[2], q3 = init[3], w = init[4];

if (q1 > p || q2 > p || q3 > p || w > p) { //Проверка на дурака

cout << "Некорректный ввод данных: q1/q2/q3/w > p! \n";

return;

}

vector <int> buffer(p); //Инициализируем рандомный вектор

for (int i = 0; i < p; i++)

buffer[i] = rand() % 2;

for (int i = 1; i <= count; i++) {

int bit = 0;

if (buffer[q1] == 1) bit += 1; //Вычисляем новый бит

if (buffer[q2] == 1) bit += 1;

if (buffer[q3] == 1) bit += 1;

bit = bit & 1;

buffer.push\_back(bit);

buffer.erase(buffer.begin());

int x = 0, deg = 1; //Вычисляем сгенерированное число размером w

if (buffer[0] == 1)

x += 1;

for (int j = 1; j < w; j++) {

deg = (deg \* 2) % 1000;

if (buffer[j] == 1)

x += deg;

}

fout << x % 1000 << endl;

if (i / (float)count \* 100 == 1) cout << " 1%... \n";

if (i / (float)count \* 100 == 25) cout << "25%... \n";

if (i / (float)count \* 100 == 50) cout << "50%... \n";

if (i / (float)count \* 100 == 75) cout << "75%... \n";

if (i / (float)count \* 100 == 100) cout << "Последовательность сгенерирована! \n";

}

fout.close();

}

Алгоритм 4. Регистр сдвига с обратной связью (РСЛОС).

**Описание алгоритма**

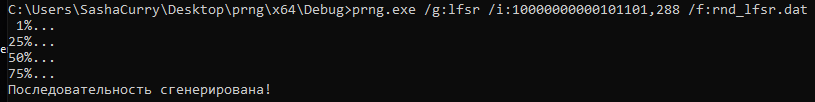
Для натурального числа и , принимающих значения 0 или 1 определяют рекуррентную формулу

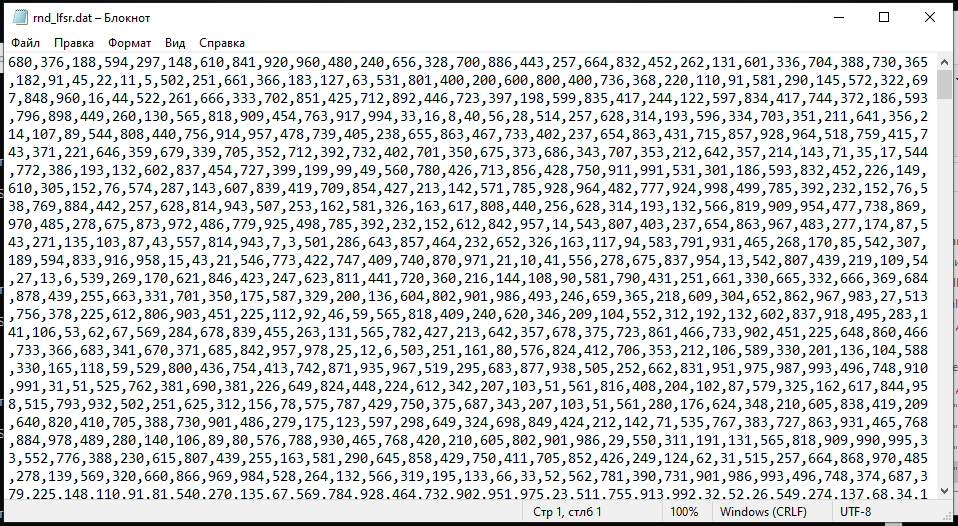
Одна итерация алгоритма, генерирующего последовательность состоит из следующих шагов:

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

**Параметры запуска программы**

prng.exe /g:lfsr /i:10000000000101101,288 /f:rnd\_lfsr.dat





**Исходный текст программы**

void algLFSR(vector <int> init, int count, string outFile, string polynom) {

ofstream fout;

fout.open(outFile);

int x0 = rand() % 1000; //Задаём начальное значение

if (!init.empty())

x0 = init[0];

for (int i = 1; i <= count; i++) {

string bitter = binForm(x0); //Бинарная форма значения регистра

while (bitter.length() < polynom.length())

bitter = "0" + bitter;

int bit = 0; //Сложение битов, указанных в полиноме

for (int j = 0; j < polynom.length(); j++)

if (polynom[polynom.length() - j - 1] == '1' && bitter[bitter.length() - j - 1] == '1')

bit++;

x0 = x0 >> 1; //Вычисляем новое значение последовательности

if (bit % 2 == 1) {

int bits = polynom.length();

x0 = (x0 + pow(2, bits - 1)) % pow(2, bits);

}

fout << x0 % 1000 << endl;

if (i / (float)count \* 100 == 1) cout << " 1%... \n";

if (i / (float)count \* 100 == 25) cout << "25%... \n";

if (i / (float)count \* 100 == 50) cout << "50%... \n";

if (i / (float)count \* 100 == 75) cout << "75%... \n";

if (i / (float)count \* 100 == 100) cout << "Последовательность сгенерирована! \n";

}

fout.close();

}

Алгоритм 5. Нелинейная комбинация РСЛОС

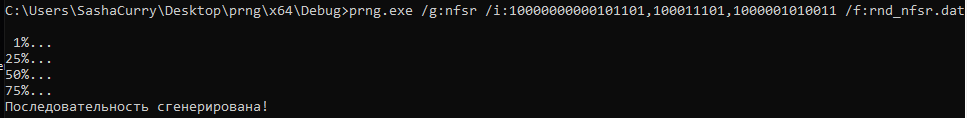
**Описание алгоритма**

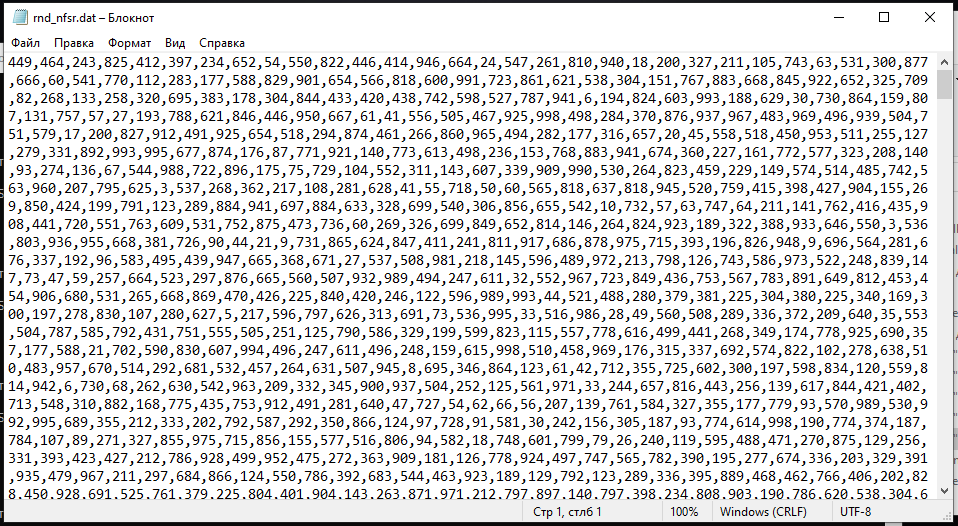
Генератор Геффа является примером нелинейной комбинацией РСЛОС. В этом генераторе используются три РСЛОС, объединённые нелинейным образом.

Нелинейная функция генератора:

**Параметры запуска программы**

prng.exe /g:nfsr /i:10000000000101101,100011101,1000001010011 /f:rnd\_nfsr.dat



****

**Исходный текст программы**

//Вспомогательная функция для нелинейной комбинации РСЛОС

int lfsrHelp(int x0, int polynom) {

string binX = binForm(x0), binPoly = binForm(polynom);

while (binX.length() < binPoly.length())

binX = "0" + binX;

int bit = 0; //Сложение битов, указанных в полиноме

for (int i = 0; i < binPoly.size(); i++)

if (binX[binX.length() - i - 1] == '1' && binPoly[binPoly.length() - i - 1] == '1')

bit++;

x0 = x0 >> 1;

if (bit % 2 == 1) {

int bits = binPoly.length();

x0 = (x0 + pow(2, bits - 1)) % pow(2, bits);

}

return x0;

}

//Нелинейная комбинация РСЛОС

void algNFSR(vector <string> polynoms, int count, string outFile) {

ofstream fout;

fout.open(outFile);

int R1 = 65581, R2 = 285, R3 = 4179, x0;

if (!polynoms.empty())

R1 = decForm(polynoms[0]), R2 = decForm(polynoms[1]), R3 = decForm(polynoms[2]);

int R1Start = R1, R2Start = R2, R3Start = R3;

for (int i = 1; i <= count; i++) {

x0 = (R1 ^ R2) + (R2 ^ R3) + R3 % 1000000000;

R1 = lfsrHelp(R1, R1Start);

R2 = lfsrHelp(R2, R2Start);

R3 = lfsrHelp(R3, R3Start);

fout << x0 % 1000 << endl;

if (i / (float)count \* 100 == 1) cout << " 1%... \n";

if (i / (float)count \* 100 == 25) cout << "25%... \n";

if (i / (float)count \* 100 == 50) cout << "50%... \n";

if (i / (float)count \* 100 == 75) cout << "75%... \n";

if (i / (float)count \* 100 == 100) cout << "Последовательность сгенерирована! \n";

}

fout.close();

}

Алгоритм 6. Вихрь Мерсенна

**Описание алгоритма**

Метод Вихрь Мерсенна позволяет генерировать последовательность двоичных псевдослучайных целых w-битных чисел в соответствии с рекуррентной формулой:

где *p, q, r* – целые константы, *p* – степень реккурентности, ;

- *w*-битное двоичное целое число;

*a* – двоичное *w*-битное целое число (формирующее матрицу *A*);

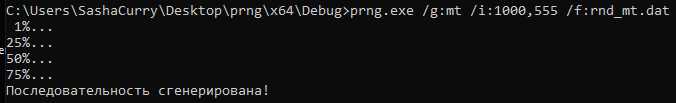
– двоичное целое число, полученное конкатенацией чисел и , когда первые *(w-r)* битов взяты из , а последние *r* битов из в том же порядке;

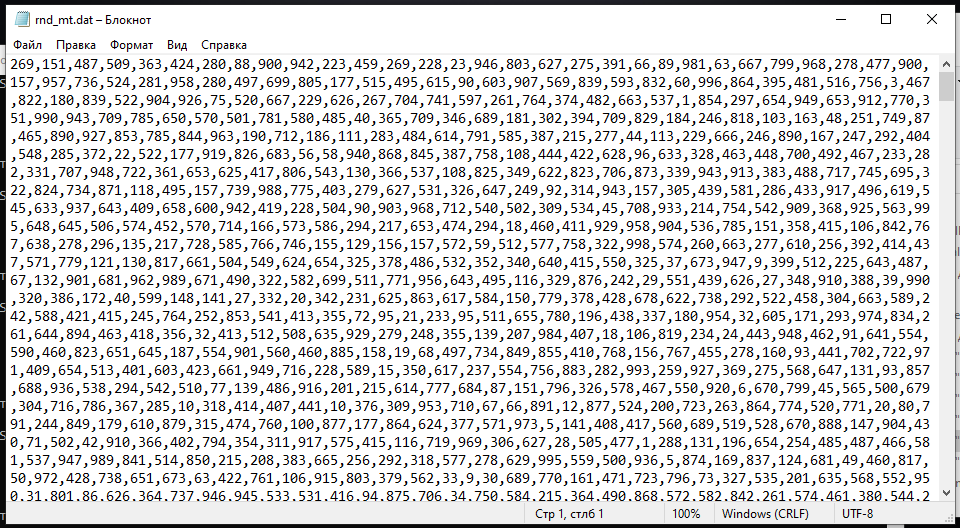
*A* – матрица размера *w*×*w*  состоящая из нулей и единиц, определенная посредством a.

*XA* – произведение, при вычислении которого сначала выполняют операцию *X>>1*, если последний бит *X* равен 0, а затем, когда последний бит *X* равен 1, то вычисляют

**Параметры запуска программы**

prng.exe /g:mt /i:1000,555 /f:rnd\_mt.dat

****

****

**Исходный текст программы**

void algMT(vector <int> init, int count, string outFile) {

ofstream fout;

fout.open(outFile);

int mod = 1000, x0 = 5489;

if (!init.empty())

mod = init[0], x0 = init[1];

int p = 624; //Параметры, наилучшие для генератора

int q = 397;

unsigned int u = 2147483648; //1000 0000 0000 0000 0000 0000 0000 0000

unsigned int h = 2147483647; //0111 1111 1111 1111 1111 1111 1111 1111

unsigned int a = 2567483615;

vector <long long> X(p); //Создаём начальные значения

X[0] = x0;

for (int i = 1; i < p; i++)

X[i] = abs(1812433253 \* (X[i - 1] ^ (X[i - 1] >> 30)) + i);

unsigned int matrixA[2] = { 0, a };

for (int i = 1; i <= count; i++) {

long long y = (X[0] & u) | (X[1] & h);

long long xn = X[q % p] ^ (y >> 1) ^ (matrixA[y & 1]);

X.push\_back(xn);

X.erase(X.begin());

y = xn;

y = y ^ (y >> 11);

y = y ^ ((y << 7) \* 2636928640);

y = y ^ ((y << 15) \* 4022730752);

long long z = y ^ (y >> 18);

fout << z % mod << endl;

if (i / (float)count \* 100 == 1) cout << " 1%... \n";

if (i / (float)count \* 100 == 25) cout << "25%... \n";

if (i / (float)count \* 100 == 50) cout << "50%... \n";

if (i / (float)count \* 100 == 75) cout << "75%... \n";

if (i / (float)count \* 100 == 100) cout << "Последовательность сгенерирована! \n";

}

}

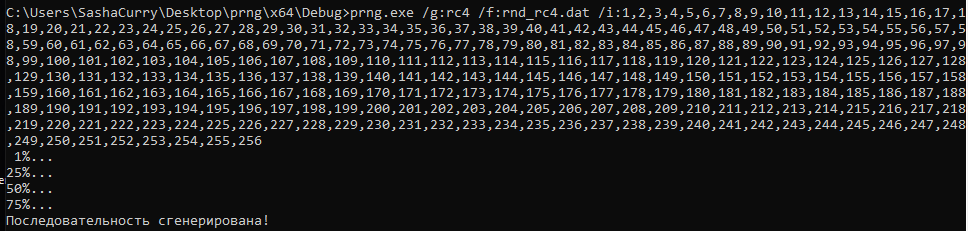
Алгоритм 7. RC4

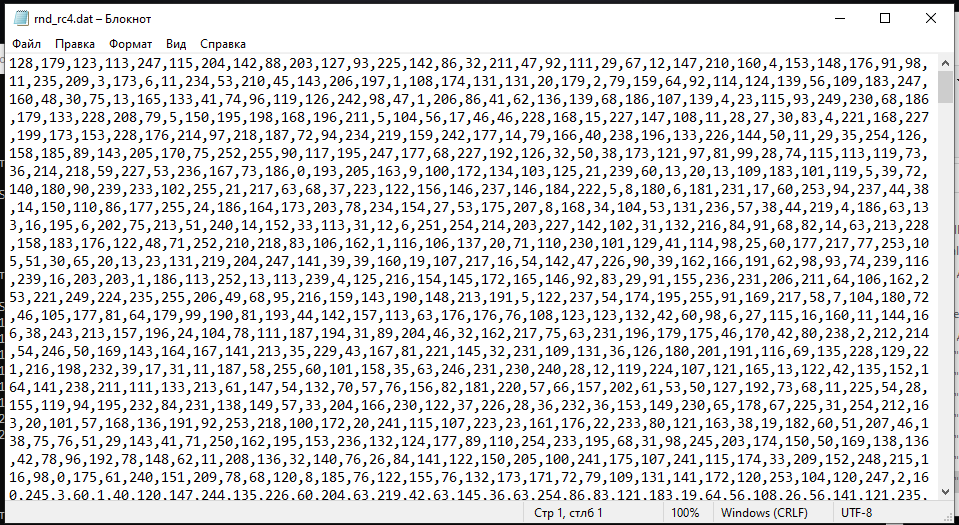
**Описание алгоритма**

1. Инициализация 𝑆𝑖, 𝑖 = 0, 1, …, 255.
   1. 𝑓𝑜𝑟 𝑖 = 0 𝑡𝑜 255: 𝑆𝑖 = 𝑖;
   2. 𝑗 = 0;
   3. 𝑓𝑜𝑟 𝑖 = 0 𝑡𝑜 255: 𝑗=(𝑗+𝑆𝑖+𝐾𝑖) 𝑚𝑜𝑑 256; s𝑤𝑎𝑝(𝑆𝑖, 𝑆𝑗)
2. 𝑖 = 0, 𝑗 = 0.
3. Итерация алгоритма:
   1. 𝑖 = (𝑖 + 1) 𝑚𝑜𝑑 256;
   2. 𝑗 = (𝑗 + 𝑆𝑖) 𝑚𝑜𝑑 256;
   3. s𝑤𝑎𝑝(𝑆𝑖, 𝑆𝑗);
   4. 𝑡 = (𝑆𝑖 + 𝑆𝑗) 𝑚𝑜𝑑 256;
   5. 𝐾=𝑆𝑡;

**Параметры запуска программы**

prng.exe /g:rc4 /f:rnd\_rc4.dat /i:1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51,52,53,54,55,56,57,58,59,60,61,62,63,64,65,66,67,68,69,70,71,72,73,74,75,76,77,78,79,80,81,82,83,84,85,86,87,88,89,90,91,92,93,94,95,96,97,98,99,100,101,102,103,104,105,106,107,108,109,110,111,112,113,114,115,116,117,118,119,120,121,122,123,124,125,126,127,128,129,130,131,132,133,134,135,136,137,138,139,140,141,142,143,144,145,146,147,148,149,150,151,152,153,154,155,156,157,158,159,160,161,162,163,164,165,166,167,168,169,170,171,172,173,174,175,176,177,178,179,180,181,182,183,184,185,186,187,188,189,190,191,192,193,194,195,196,197,198,199,200,201,202,203,204,205,206,207,208,209,210,211,212,213,214,215,216,217,218,219,220,221,222,223,224,225,226,227,228,229,230,231,232,233,234,235,236,237,238,239,240,241,242,243,244,245,246,247,248,249,250,251,252,253,254,255,256

****

****

**Исходный текст программы**

void algRC4(vector <int> init, int count, string outFile) {

ofstream fout;

fout.open(outFile);

vector <int> S(256), K(256); //Создаём и заполняем массивы

for (int i = 0; i < 256; i++)

S[i] = i;

if (init.empty())

for (int i = 0; i < 256; i++)

K[i] = i;

else

K = init;

int i = 0, j = 0; //Перемешиваем значения в таблице S

for (i = 0; i < 256; i++) {

j = (j + S[i] + K[i]) % 256;

swap(S[i], S[j]);

}

for (int n = 1; n <= count; n++) { //Генерируем последовательность

i = (i + 1) % 256;

j = (j + S[i]) % 256;

swap(S[i], S[j]);

int a = (S[i] + S[j]) % 256;

fout << S[a] << endl;

if (n / (float)count \* 100 == 1) cout << " 1%... \n";

if (n / (float)count \* 100 == 25) cout << "25%... \n";

if (n / (float)count \* 100 == 50) cout << "50%... \n";

if (n / (float)count \* 100 == 75) cout << "75%... \n";

if (n / (float)count \* 100 == 100) cout << "Последовательность сгенерирована! \n";

}

fout.close();

}

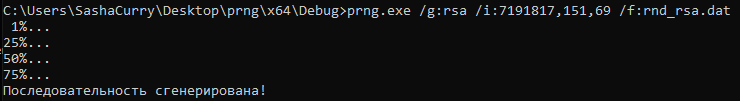
Алгоритм 8. RSA

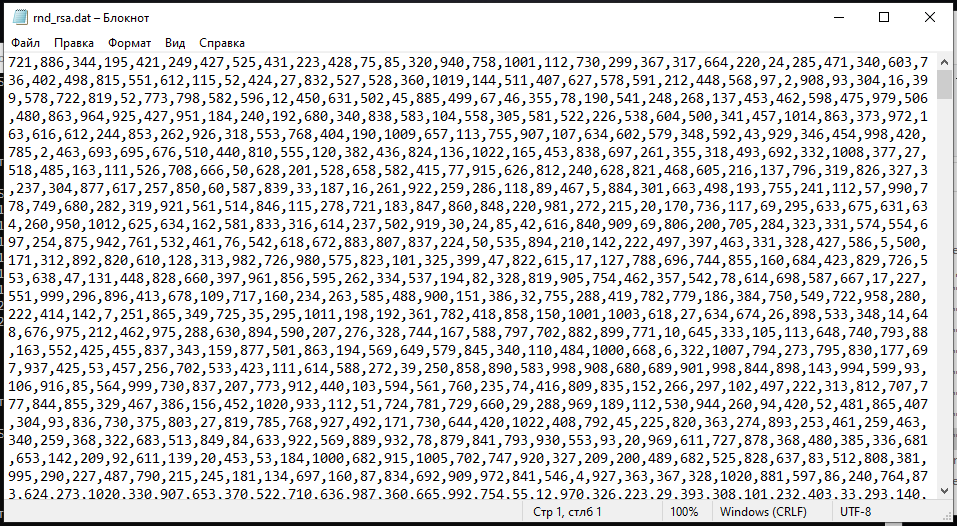
**Описание алгоритма**

1. Сгенерировать два секретных простых числа p и *q*, а также и . Выбрать случайное целое число , такое, что НОД(𝑒, 𝑓) = 1.
2. Выбрать случайное целое –начальный вектор из интервала [1, 𝑛 − 1].
3. 𝐹𝑜𝑟 𝑖 = 1 𝑡𝑜 𝑙 𝑑𝑜
   1. ← 𝑚𝑜𝑑 𝑛.
   2. 𝑧𝑖 ← последний значащий бит 𝑥𝑖
4. Вернуть 𝑧1, 𝑧2, …, 𝑧𝑙.

**Параметры запуска программы**

prng.exe /g:rsa /i:7191817,151,69 /f:rnd\_rsa.dat



****

**Исходный текст программы**

void algRSA(vector <int> init, int count, string outFile) {

ofstream fout;

fout.open(outFile);

int n = 7191817, e = 151, x0 = 69;

if (!init.empty()) {

n = init[0], e = init[1], x0 = init[2];

}

if (x0 < 1 || x0 > n) { //Проверка на дурака

cout << "Начальное значенеи превышает модуль!\n";

return;

}

for (int i = 1; i <= count; i++) {

string z = "";

int x = x0;

for (int j = 0; j < 10; j++) {

x = powClosed(x, e, n);

z = z + binForm(x).back();

}

x0 = x;

fout << decForm(z) << endl;

if (i / (float)count \* 100 == 1) cout << " 1%... \n";

if (i / (float)count \* 100 == 25) cout << "25%... \n";

if (i / (float)count \* 100 == 50) cout << "50%... \n";

if (i / (float)count \* 100 == 75) cout << "75%... \n";

if (i / (float)count \* 100 == 100) cout << "Последовательность сгенерирована! \n";

}

fout.close();

}

Алгоритм 9. Алгоритм Блюм – Блюма - Шуба

**Описание алгоритма**

**На входе:** Длина *l*.

**На выходе:** Последовательность псевдослучайных бит 𝑧1, 𝑧2, …, 𝑧𝑙.

1. Сгенерировать два простых числа *p* и *q*, сравнимых с 3 по модулю 4. Произведение этих чисел, *n*, является целым числом Блюма. Выберем другое случайное целое число *x*, взаимно простое с *n*.

2. Вычислим 𝑥0 = 𝑥2 𝑚𝑜𝑑 𝑛, которое будет начальным вектором.

3. 𝐹𝑜𝑟 𝑖 = 1 𝑡𝑜 𝑙 𝑑𝑜

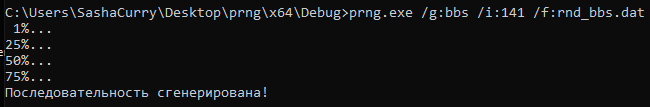
3.1. ← 𝑚𝑜𝑑 𝑛.

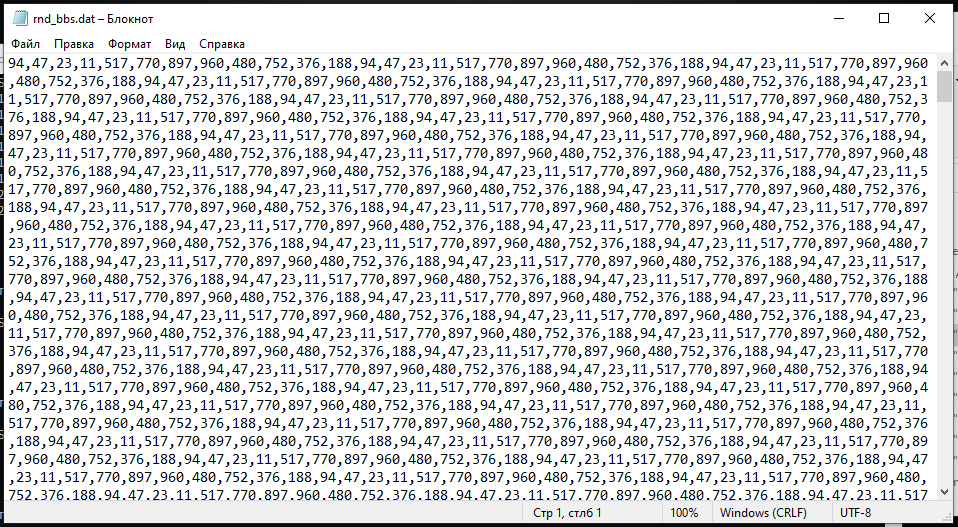
3.2. 𝑧𝑖 ← последний значащий бит

4. Вернуть 𝑧1, 𝑧2, …, 𝑧𝑙.

**Параметры запуска программы**

prng.exe /g:bbs /i:141 /f:rnd\_bbs.dat





**Исходный текст программы**

void algBBS(vector <int> init, int count, string outFile) {

ofstream fout;

fout.open(outFile);

int p = 127, q = 131, n = 16637, x0 = 3137; //Начальные значения

if (!init.empty()) {

if (nod(init[0], n) == 1)

x0 = init[0];

else {

cout << "Начальное значение не взаимно простое с n = 16637! \n";

return;

}

}

for (int i = 1; i <= count; i++) {

string z = "";

int x = (x0 \* x0) % n;

for (int j = 0; j < 10; j++) {

x = (x \* x) % n;

z = z + binForm(x).back();

}

x0 = x;

fout << decForm(z) << endl;

if (i / (float)count \* 100 == 1) cout << " 1%... \n";

if (i / (float)count \* 100 == 25) cout << "25%... \n";

if (i / (float)count \* 100 == 50) cout << "50%... \n";

if (i / (float)count \* 100 == 75) cout << "75%... \n";

if (i / (float)count \* 100 == 100) cout << "Последовательность сгенерирована! \n";

}

fout.close();

}

**Приложение. Программа для запуска генераторов**

#include "iostream"

#include "string"

#include "vector"

#include "fstream"

#include "cmath"

#include "stdlib.h"

#include "time.h"

#include "utility"

using namespace std;

int nod(int x, int y) {

int r = y % x;

if (r == 0)

return x;

else

nod(r, x);

}

int pow(int x, int n) {

int res = 1;

for (int i = 0; i < n; i++)

res \*= x;

return res;

}

//Возведение в степень по модулю

int powClosed(int x, int n, int mod) {

int res = 1;

for (int i = 0; i < n; i++)

res = (res \* x) % mod;

return res;

}

//Представляем число в 2 СС

string binForm(int x) {

string bitter = "";

while (x != 0) {

bitter = to\_string(x % 2) + bitter;

x /= 2;

}

return bitter;

}

//Представляем число в 10 СС

int decForm(string x) {

int res = 0, deg = 1;

if (x[x.length() - 1] == '1')

res += 1;

for (int i = 1; i < x.length(); i++) {

deg = deg \* 2;

if (x[x.length() - i - 1] == '1')

res += deg;

}

return res;

}

//Мануал

void \_\_help() {

cout << "/g:<код\_метода> - параметр указывает на метод генерации ПСЧ, при этом код\_метода может быть одним из следующих: **\n**";

cout << " lc - линейный конгруэнтный метод (вектор инициализации: m, a, c, x0); **\n**";

cout << " add - аддитивный метод (вектор инициализации: модуль, ст.индекс, мл.индекс, последовательность начальных значений); **\n**";

cout << " 5p - пятипараметрический метод (вектор инициализации: p, q1, q2, q3, w); **\n**";

cout << " lfsr - регистр сдвига с обратной связью (вектор инициализации: вектор коэффициентов в 2 СС, x0) **\n**";

cout << " nfsr - нелинейная комбинация РСЛОС (вектор инициализации: R1, R2, R3 - все в 2 СС); **\n**";

cout << " mt - вихрь Мерсенна (вектор инициализации: m, x0); **\n**";

cout << " rc4 - RC4 (вектор инициализации: 256 начальных значений); **\n**";

cout << " rsa - ПСЧ на основе RSA (вектор инициализации: модуль n, число e, начальное значение x); **\n**";

cout << " bbs - алгоритм Блюма-Блюма-Шуба (вектор инициализации: начальоне число x (вз-пр с n = 16637). **\n**";

cout << "/i:<число> - инициализационный вектор генератора. **\n**";

cout << "/n:<длина> - количество генерируемых чисел. Если параметр не указан, - генерируется 10000 чисел. **\n**";

cout << "/f:<полное\_имя\_файла> - полное имя файла, в который будут выводиться данные.";

cout << "Если параметр не указан, данные должны записываться в файл с именем rnd.dat. **\n**";

cout << "/h – информация о допустимых параметрах командной строки программы. **\n**";

}

//Линейный когнруэнтный метод

void algLC(vector <int> init, int count, string outFile) {

ofstream fout;

fout.open(outFile);

long long x0 = rand() % 1000, a = 1103515245, c = 12345, m = 2147483647, mod = 1000;

if (!init.empty())

m = init[0], a = init[1], c = init[2], x0 = init[3], mod = init[0];

if (m <= 0 || a < 0 || a > m || c < 0 || c > m || x0 < 0 || x0 > m) { //Проверка на дурака

cout << "Некорректный ввод данных! **\n**";

return;

}

fout << x0 << endl;

for (int i = 2; i <= count; i++) {

x0 = (a \* x0 + c) % m;

fout << x0 % mod << endl;

if (i / (float)count \* 100 == 1) cout << " 1%... **\n**";

if (i / (float)count \* 100 == 25) cout << "25%... **\n**";

if (i / (float)count \* 100 == 50) cout << "50%... **\n**";

if (i / (float)count \* 100 == 75) cout << "75%... **\n**";

if (i / (float)count \* 100 == 100) cout << "Последовательность сгенерирована! **\n**";

}

fout.close();

}

//Аддитивный метод

void algADD(vector <int> init, int count, string outFile) {

ofstream fout;

fout.open(outFile);

int m, iYoung, iOld; //Инициализируем начальные значения

vector <int> startVal;

if (init.empty()) {

m = 1000, iYoung = 55, iOld = 24;

for (int i = 0; i < 56; i++) {

startVal.push\_back(rand() % 1000);

fout << startVal[i] << endl;

}

}

else {

m = init[0], iYoung = init[1], iOld = init[2];

for (int i = 3; i < init.size(); i++) {

startVal.push\_back(init[i] % m);

fout << startVal[i - 3] << endl;

}

}

for (int i = startVal.size(); i <= count; i++) {

int xNew = (startVal[iOld - 1] + startVal[iYoung - 1]) % m;

fout << xNew << endl;

startVal.push\_back(xNew);

startVal.erase(startVal.begin());

if (i / (float)count \* 100 == 1) cout << " 1%... **\n**";

if (i / (float)count \* 100 == 25) cout << "25%... **\n**";

if (i / (float)count \* 100 == 50) cout << "50%... **\n**";

if (i / (float)count \* 100 == 75) cout << "75%... **\n**";

if (i / (float)count \* 100 == 100) cout << "Последовательность сгенерирована! **\n**";

}

fout.close();

}

//Пятипараметрический метод

void alg5P(vector <int> init, int count, string outFile) {

ofstream fout;

fout.open(outFile);

int p = 607, q1 = 167, q2 = 307, q3 = 461, w = 15;

if (!init.empty())

p = init[0], q1 = init[1], q2 = init[2], q3 = init[3], w = init[4];

if (q1 > p || q2 > p || q3 > p || w > p) { //Проверка на дурака

cout << "Некорректный ввод данных: q1/q2/q3/w > p! **\n**";

return;

}

vector <int> buffer(p); //Инициализируем рандомный вектор

for (int i = 0; i < p; i++)

buffer[i] = rand() % 2;

for (int i = 1; i <= count; i++) {

int bit = 0;

if (buffer[q1] == 1) bit += 1; //Вычисляем новый бит

if (buffer[q2] == 1) bit += 1;

if (buffer[q3] == 1) bit += 1;

bit = bit & 1;

buffer.push\_back(bit);

buffer.erase(buffer.begin());

int x = 0, deg = 1; //Вычисляем сгенерированное число размером w

if (buffer[0] == 1)

x += 1;

for (int j = 1; j < w; j++) {

deg = (deg \* 2) % 1000;

if (buffer[j] == 1)

x += deg;

}

fout << x % 1000 << endl;

if (i / (float)count \* 100 == 1) cout << " 1%... **\n**";

if (i / (float)count \* 100 == 25) cout << "25%... **\n**";

if (i / (float)count \* 100 == 50) cout << "50%... **\n**";

if (i / (float)count \* 100 == 75) cout << "75%... **\n**";

if (i / (float)count \* 100 == 100) cout << "Последовательность сгенерирована! **\n**";

}

fout.close();

}

//Регистр сдвига с обратной связью (РСЛОС)

void algLFSR(vector <int> init, int count, string outFile, string polynom) {

ofstream fout;

fout.open(outFile);

int x0 = rand() % 1000; //Задаём начальное значение

if (!init.empty())

x0 = init[0];

for (int i = 1; i <= count; i++) {

string bitter = binForm(x0); //Бинарная форма значения регистра

while (bitter.length() < polynom.length())

bitter = "0" + bitter;

int bit = 0; //Сложение битов, указанных в полиноме

for (int j = 0; j < polynom.length(); j++)

if (polynom[polynom.length() - j - 1] == '1' && bitter[bitter.length() - j - 1] == '1')

bit++;

x0 = x0 >> 1; //Вычисляем новое значение последовательности

if (bit % 2 == 1) {

int bits = polynom.length();

x0 = (x0 + pow(2, bits - 1)) % pow(2, bits);

}

fout << x0 % 1000 << endl;

if (i / (float)count \* 100 == 1) cout << " 1%... **\n**";

if (i / (float)count \* 100 == 25) cout << "25%... **\n**";

if (i / (float)count \* 100 == 50) cout << "50%... **\n**";

if (i / (float)count \* 100 == 75) cout << "75%... **\n**";

if (i / (float)count \* 100 == 100) cout << "Последовательность сгенерирована! **\n**";

}

fout.close();

}

//Вспомогательная функция для нелинейной комбинации РСЛОС

int lfsrHelp(int x0, int polynom) {

string binX = binForm(x0), binPoly = binForm(polynom);

while (binX.length() < binPoly.length())

binX = "0" + binX;

int bit = 0; //Сложение битов, указанных в полиноме

for (int i = 0; i < binPoly.size(); i++)

if (binX[binX.length() - i - 1] == '1' && binPoly[binPoly.length() - i - 1] == '1')

bit++;

x0 = x0 >> 1;

if (bit % 2 == 1) {

int bits = binPoly.length();

x0 = (x0 + pow(2, bits - 1)) % pow(2, bits);

}

return x0;

}

//Нелинейная комбинация РСЛОС

void algNFSR(vector <string> polynoms, int count, string outFile) {

ofstream fout;

fout.open(outFile);

int R1 = 65581, R2 = 285, R3 = 4179, x0;

if (!polynoms.empty())

R1 = decForm(polynoms[0]), R2 = decForm(polynoms[1]), R3 = decForm(polynoms[2]);

int R1Start = R1, R2Start = R2, R3Start = R3;

for (int i = 1; i <= count; i++) {

x0 = (R1 ^ R2) + (R2 ^ R3) + R3 % 1000000000;

R1 = lfsrHelp(R1, R1Start);

R2 = lfsrHelp(R2, R2Start);

R3 = lfsrHelp(R3, R3Start);

fout << x0 % 1000 << endl;

if (i / (float)count \* 100 == 1) cout << " 1%... **\n**";

if (i / (float)count \* 100 == 25) cout << "25%... **\n**";

if (i / (float)count \* 100 == 50) cout << "50%... **\n**";

if (i / (float)count \* 100 == 75) cout << "75%... **\n**";

if (i / (float)count \* 100 == 100) cout << "Последовательность сгенерирована! **\n**";

}

fout.close();

}

//Вихрь Мерсенна

void algMT(vector <int> init, int count, string outFile) {

ofstream fout;

fout.open(outFile);

int mod = 1000, x0 = 5489;

if (!init.empty())

mod = init[0], x0 = init[1];

int p = 624; //Параметры, наилучшие для генератора

int q = 397;

unsigned int u = 2147483648; //1000 0000 0000 0000 0000 0000 0000 0000

unsigned int h = 2147483647; //0111 1111 1111 1111 1111 1111 1111 1111

unsigned int a = 2567483615;

vector <long long> X(p); //Создаём начальные значения

X[0] = x0;

for (int i = 1; i < p; i++)

X[i] = abs(1812433253 \* (X[i - 1] ^ (X[i - 1] >> 30)) + i);

unsigned int matrixA[2] = { 0, a };

for (int i = 1; i <= count; i++) {

long long y = (X[0] & u) | (X[1] & h);

long long xn = X[q % p] ^ (y >> 1) ^ (matrixA[y & 1]);

X.push\_back(xn);

X.erase(X.begin());

y = xn;

y = y ^ (y >> 11);

y = y ^ ((y << 7) \* 2636928640);

y = y ^ ((y << 15) \* 4022730752);

long long z = y ^ (y >> 18);

fout << z % mod << endl;

if (i / (float)count \* 100 == 1) cout << " 1%... **\n**";

if (i / (float)count \* 100 == 25) cout << "25%... **\n**";

if (i / (float)count \* 100 == 50) cout << "50%... **\n**";

if (i / (float)count \* 100 == 75) cout << "75%... **\n**";

if (i / (float)count \* 100 == 100) cout << "Последовательность сгенерирована! **\n**";

}

}

//Генератор RC4

void algRC4(vector <int> init, int count, string outFile) {

ofstream fout;

fout.open(outFile);

vector <int> S(256), K(256); //Создаём и заполняем массивы

for (int i = 0; i < 256; i++)

S[i] = i;

if (init.empty())

for (int i = 0; i < 256; i++)

K[i] = i;

else

K = init;

int i = 0, j = 0; //Перемешиваем значения в таблице S

for (i = 0; i < 256; i++) {

j = (j + S[i] + K[i]) % 256;

swap(S[i], S[j]);

}

for (int n = 1; n <= count; n++) { //Генерируем последовательность

i = (i + 1) % 256;

j = (j + S[i]) % 256;

swap(S[i], S[j]);

int a = (S[i] + S[j]) % 256;

fout << S[a] << endl;

if (n / (float)count \* 100 == 1) cout << " 1%... **\n**";

if (n / (float)count \* 100 == 25) cout << "25%... **\n**";

if (n / (float)count \* 100 == 50) cout << "50%... **\n**";

if (n / (float)count \* 100 == 75) cout << "75%... **\n**";

if (n / (float)count \* 100 == 100) cout << "Последовательность сгенерирована! **\n**";

}

fout.close();

}

//ГПСЧ на основе RSA

void algRSA(vector <int> init, int count, string outFile) {

ofstream fout;

fout.open(outFile);

int n = 7191817, e = 151, x0 = 69;

if (!init.empty()) {

n = init[0], e = init[1], x0 = init[2];

}

if (x0 < 1 || x0 > n) { //Проверка на дурака

cout << "Начальное значенеи превышает модуль!**\n**";

return;

}

for (int i = 1; i <= count; i++) {

string z = "";

int x = x0;

for (int j = 0; j < 10; j++) {

x = powClosed(x, e, n);

z = z + binForm(x).back();

}

x0 = x;

fout << decForm(z) << endl;

if (i / (float)count \* 100 == 1) cout << " 1%... **\n**";

if (i / (float)count \* 100 == 25) cout << "25%... **\n**";

if (i / (float)count \* 100 == 50) cout << "50%... **\n**";

if (i / (float)count \* 100 == 75) cout << "75%... **\n**";

if (i / (float)count \* 100 == 100) cout << "Последовательность сгенерирована! **\n**";

}

fout.close();

}

//Алгоритм Блюма-Блюма-Шуба (BBS)

void algBBS(vector <int> init, int count, string outFile) {

ofstream fout;

fout.open(outFile);

int p = 127, q = 131, n = 16637, x0 = 3137; //Начальные значения

if (!init.empty()) {

if (nod(init[0], n) == 1)

x0 = init[0];

else {

cout << "Начальное значение не взаимно простое с n = 16637! **\n**";

return;

}

}

for (int i = 1; i <= count; i++) {

string z = "";

int x = (x0 \* x0) % n;

for (int j = 0; j < 10; j++) {

x = (x \* x) % n;

z = z + binForm(x).back();

}

x0 = x;

fout << decForm(z) << endl;

if (i / (float)count \* 100 == 1) cout << " 1%... **\n**";

if (i / (float)count \* 100 == 25) cout << "25%... **\n**";

if (i / (float)count \* 100 == 50) cout << "50%... **\n**";

if (i / (float)count \* 100 == 75) cout << "75%... **\n**";

if (i / (float)count \* 100 == 100) cout << "Последовательность сгенерирована! **\n**";

}

fout.close();

}

//Главная функция

int main(int argc, char\* argv[]) {

setlocale(LC\_ALL, "ru");

srand(time(NULL));

string method = "lfsr";

string strInit = "";

int count = 10000;

string outFile = "rnd.dat";

//Считываем параметры

for (int i = 1; i < argc; i++) {

string str = argv[i];

switch (argv[i][1]) {

case 'g':

method = str.erase(0, 3);

break;

case 'i':

strInit = str.erase(0, 3);

break;

case 'n':

count = stoi(str.erase(0, 3));

break;

case 'f':

outFile = str.erase(0, 3);

break;

case 'h':

\_\_help();

return 0;

default:

cout << str << " - неверный параметр **\n**";

return 0;

}

}

//Разбиваем вектор инициализации на элементы

vector <int> init;

string polynomLFSR = "";

vector <string> polynomNFSR;

string num = "";

for (int i = 0; i <= strInit.length(); i++) {

if (isdigit(strInit[i]))

num += strInit[i];

else if (num != "") {

if (method == "lfsr" && polynomLFSR.empty())

polynomLFSR = num;

else if (method == "nfsr")

polynomNFSR.push\_back(num);

else

init.push\_back(stoi(num));

num = "";

}

}

if (method == "lfsr" && polynomLFSR.empty())

polynomLFSR = "10000000000101101";

//Генерируем последовательность

if (method == "lc" && (init.size() == 4 || init.empty()))

algLC(init, count, outFile);

else if (method == "add" && (init.empty() || init.size() >= init[2] + 3))

algADD(init, count, outFile);

else if (method == "5p" && (init.size() == 5 || init.empty()))

alg5P(init, count, outFile);

else if (method == "lfsr" && (init.size() == 1 || init.empty()))

algLFSR(init, count, outFile, polynomLFSR);

else if (method == "nfsr" && (polynomNFSR.size() == 3 || polynomNFSR.empty()))

algNFSR(polynomNFSR, count, outFile);

else if (method == "rc4" && (init.size() == 256 || init.empty()))

algRC4(init, count, outFile);

else if (method == "mt" && (init.size() == 2 || init.empty()))

algMT(init, count, outFile);

else if (method == "rsa" && (init.size() == 3 || init.empty()))

algRSA(init, count, outFile);

else if (method == "bbs" && (init.size() == 1 || init.empty()))

algBBS(init, count, outFile);

else

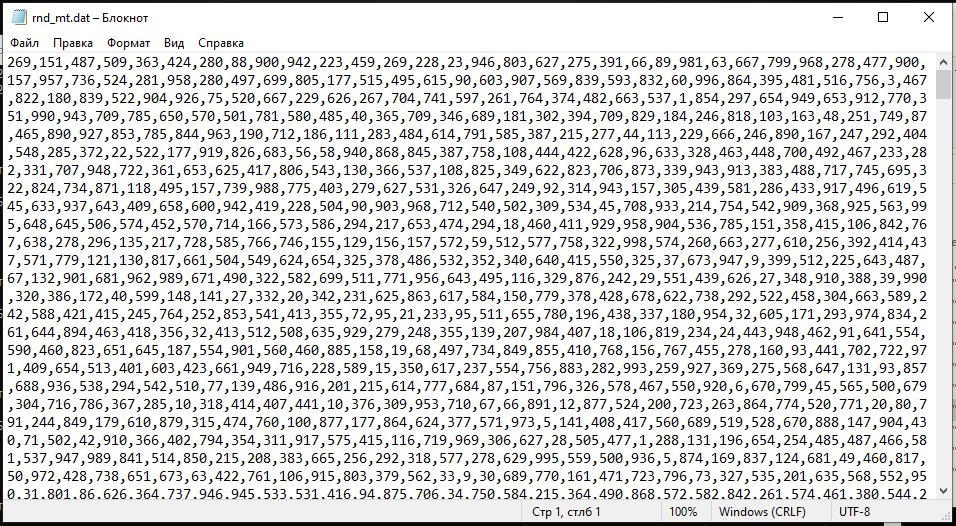
cout << "Некорректный алгоритм или его вектор инициализации! **\n**";

return 0;

}

# Задание 2. Преобразование ПСЧ к заданному распределению

Для получения разбиения использовалась последовательность, полученная с помощью вихря Мерсенна:



Если максимальное значение равномерного целого случайного числа равно , для генерации стандартных равномерных случайных чисел необходимо применять следующую формулу: .

## Распределение 1. Стандартное равномерное с заданным интервалом

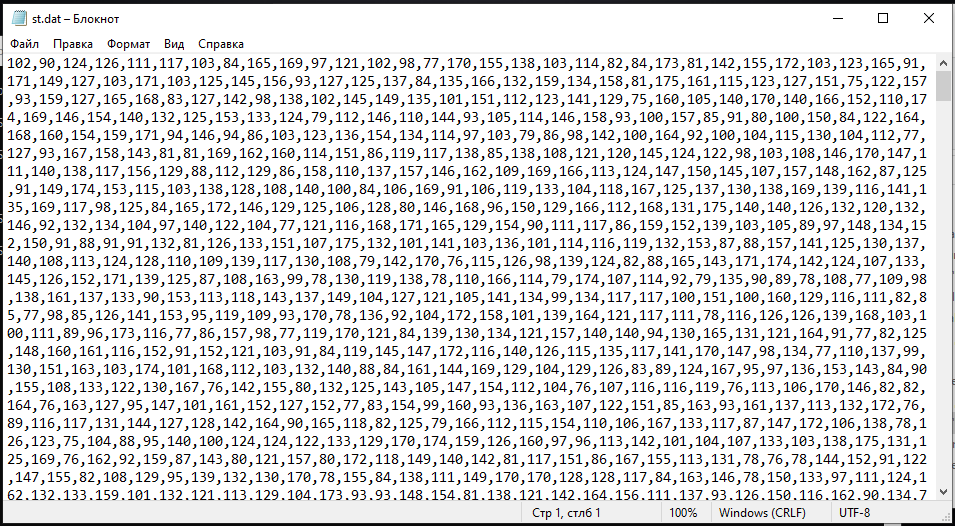
### Описание алгоритма.

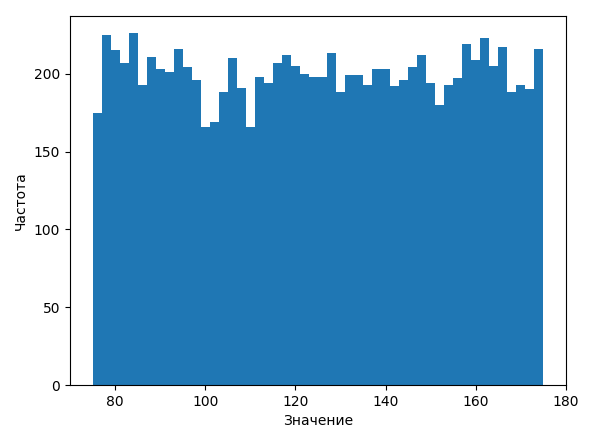
Равномерное случайное число должно быть получено в соответствии со следующей формулой:

, где *U* = *X* / *m* – число, соответствующее стандартному равномерному распределению

### Параметры запуска программы

rnc.exe /d:st /f:rnd.dat /p1:75 /p2:100



****

### Исходный текст программы

int findMaxEl(string inFile) {

int maxEl = 0;

string str = "";

char el;

ifstream fin(inFile);

while (fin.get(el)) { //Ищем наибольшой элемент

if (isdigit(el))

str = str + el;

else if (stoi(str) > maxEl) {

maxEl = stoi(str);

str = "";

}

else

str = "";

}

if (str != "" && stoi(str) > maxEl) {

maxEl = stoi(str);

str = "";

}

fin.close();

return maxEl;

}

//Стандартное равномерное распределние с заданным интервалом

void distST(string inFile, int p1, int p2) {

string str = "";

char el;

ifstream fin(inFile);

if (!fin.is\_open()) {

cout << "Файл " << inFile << " не найден! \n";

return;

}

double m = findMaxEl(inFile) + 1;

ofstream fout("st.dat");

while (fin.get(el)) { //Генерируем распределение

if (isdigit(el))

str = str + el;

else {

fout << round(p1 + (stoi(str) / m) \* p2) << ",";

str = "";

}

}

if (str != "")

fout << round(p1 + (stoi(str) / m) \* p2);

fin.close();

fout.close();

}

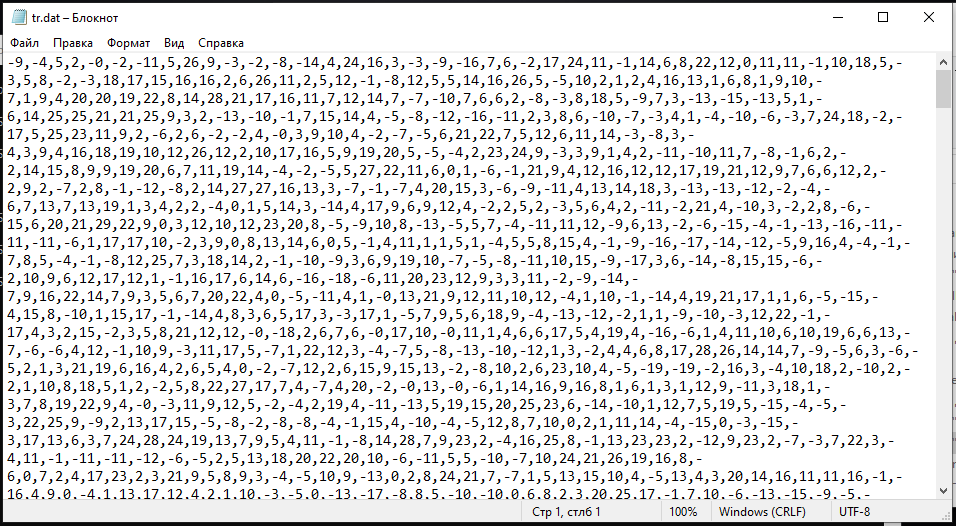
## Распределение 2. Треугольное распределение

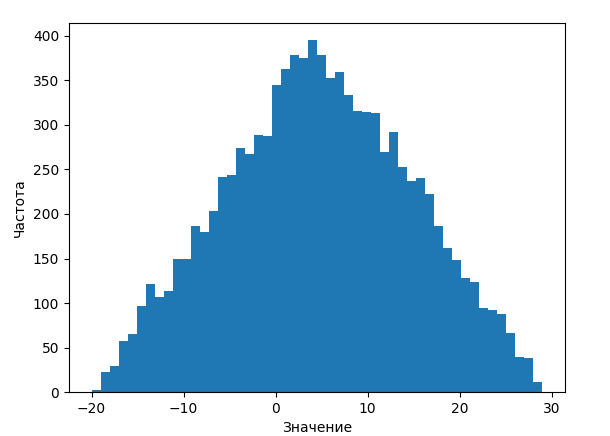
### Описание алгоритма

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

**Параметры запуска программы**

rnc.exe /f:rnd.dat /d:tr /p1:5 /p2:25

****



### Исходный текст программы

void distTR(string inFile, int p1, int p2) {

string str = "";

char el;

ifstream fin(inFile);

if (!fin.is\_open()) {

cout << "Файл " << inFile << " не найден! \n";

return;

}

double m = findMaxEl(inFile) + 1;

int prev = 0;

while (fin.get(el)) { //Первый элемент последовательности

if (isdigit(el))

str = str + el;

else {

prev = stoi(str);

str = "";

break;

}

}

ofstream fout("tr.dat");

while (fin.get(el)) { //Генерируем распределение

if (isdigit(el))

str = str + el;

else {

fout << round(p1 + p2 \* (prev / m + stoi(str) / m - 1)) << ",";

prev = stoi(str);

str = "";

}

}

if (str != "")

fout << round(p1 + p2 \* (prev / m + stoi(str) / m - 1));

fin.close();

fout.close();

}

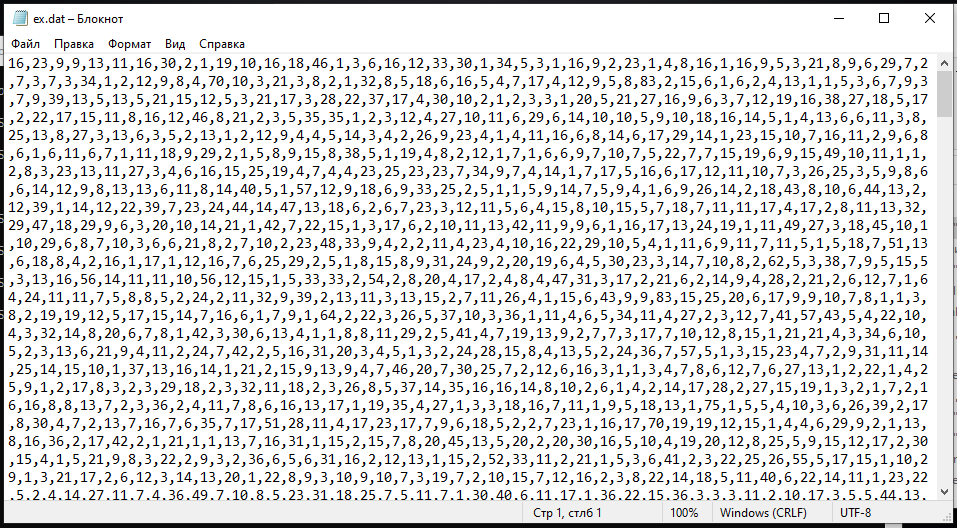
## Распределение 3. Общее экспоненциальное распределение

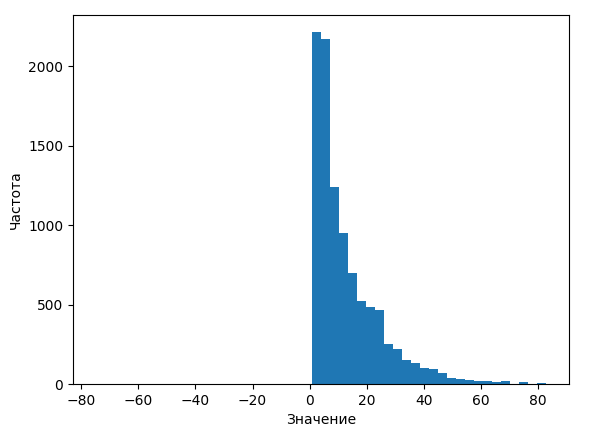
### Описание алгоритма

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

### Параметры запуска программы

rnc.exe /f:rnd.dat /d:ex /p1:1 /p2:12



****

### Исходный текст программы

void distEX(string inFile, int p1, int p2) {

string str = "";

char el;

ifstream fin(inFile);

ofstream fout("ex.dat");

if (!fin.is\_open()) {

cout << "Файл " << inFile << " не найден! \n";

return;

}

double m = findMaxEl(inFile) + 1;

while (fin.get(el)) { //Генерируем распределение

if (isdigit(el))

str = str + el;

else if (str == "0")

str = "";

else {

int res = -p2 \* log(stoi(str) / m) + p1;

fout << res << ",";

str = "";

}

}

if (str != "" && str != "0")

fout << int(-p2 \* log(stoi(str)) + p1);

fin.close();

fout.close();

}

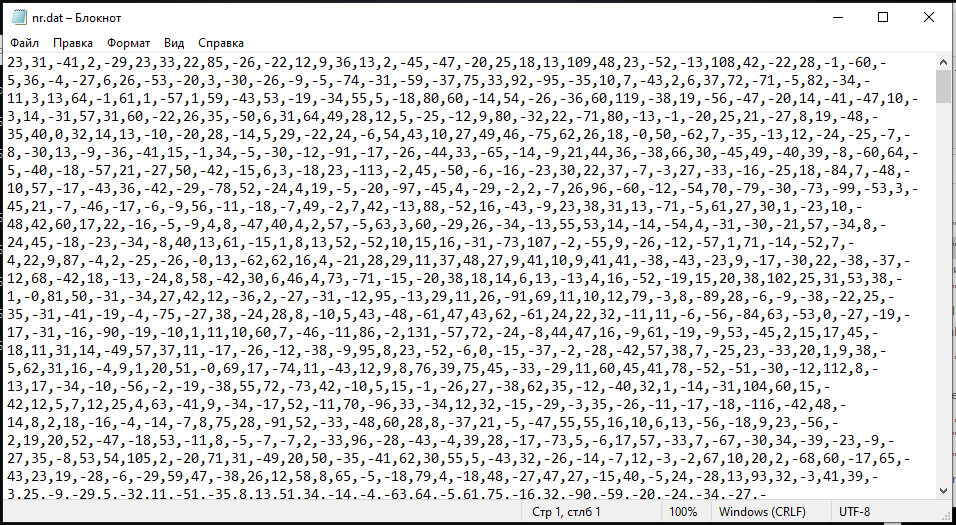
## Распределение 4. Нормальное распределение

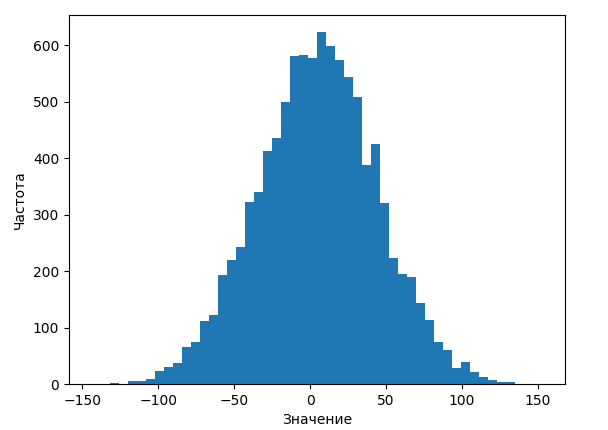
### Описание алгоритма

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

### Параметры запуска программы

rnc.exe /f:rnd.dat /d:nr /p1:5 /p2:40





### Исходный текст программы

void distNR(string inFile, int p1, int p2) {

string str = "";

char el;

ifstream fin(inFile);

ofstream fout("nr.dat");

if (!fin.is\_open()) {

cout << "Файл " << inFile << " не найден! \n";

return;

}

double m = findMaxEl(inFile) + 1;

vector <double> nums;

while (fin.get(el)) { //Генерируем распределение

if (isdigit(el))

str = str + el;

else {

nums.push\_back(stoi(str) / m);

if (nums.size() == 2) {

fout << round(p1 + p2 \* sqrt(-2 \* log(1 - nums[0])) \* cos(2 \* PI \* nums[1])) << ",";

fout << round(p1 + p2 \* sqrt(-2 \* log(1 - nums[0])) \* sin(2 \* PI \* nums[1])) << ",";

nums.clear();

}

str = "";

}

}

if (str != "" && nums.size() == 1) {

fout << round(p1 + p2 \* sqrt(-2 \* log(1 - nums[0])) \* cos(2 \* PI \* stoi(str) / m)) << ",";

fout << round(p1 + p2 \* sqrt(-2 \* log(1 - nums[0])) \* sin(2 \* PI \* stoi(str) / m));

}

fin.close();

fout.close();

}

## Распределение 5. Гамма распределение

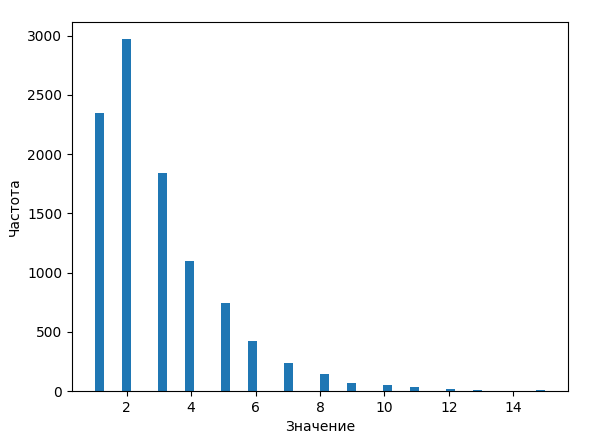
### Описание алгоритма

Используя независимую равномерную случайную величину 𝑈1 применяем формулу (рассмотрен частный случай для *c* = 1):

### Параметры запуска программы

rnc.exe /f:rnd.dat /d:gm /p1:1 /p2:2





### Исходный текст программы

void distGM(string inFile, int p1, int p2) {

string str = "";

char el;

ifstream fin(inFile);

ofstream fout("gm.dat");

if (!fin.is\_open()) {

cout << "Файл " << inFile << "не найден! \n";

return;

}

double m = findMaxEl(inFile) + 1;

while (fin.get(el)) { //Генерируем распределение

if (isdigit(el))

str = str + el;

else {

fout << round(p1 - p2 \* log(1 - stoi(str) / m)) << ",";

str = "";

}

}

if (str != "") {

fout << round(p1 - p2 \* log(1 - stoi(str) / m));

}

fin.close();

fout.close();

}

## Распределение 6. Логнормальное распределение

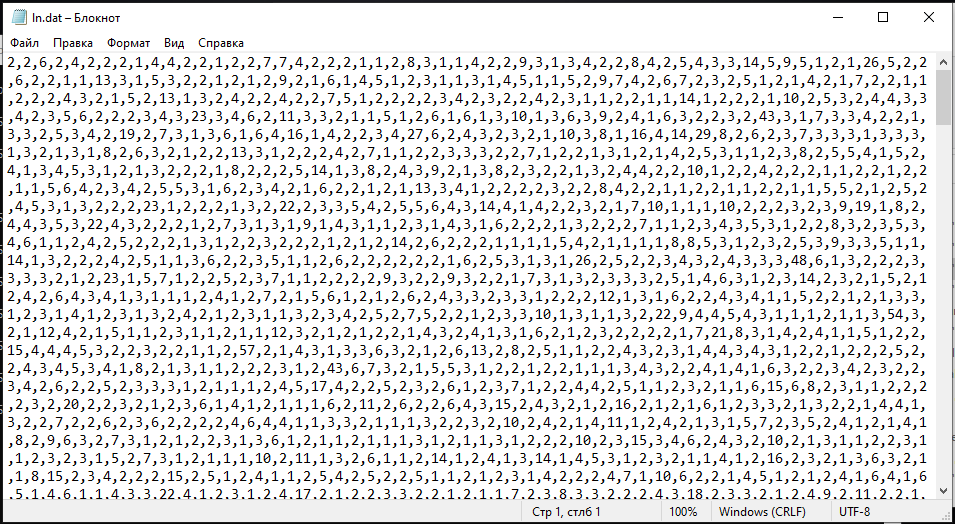
### Описание алгоритма

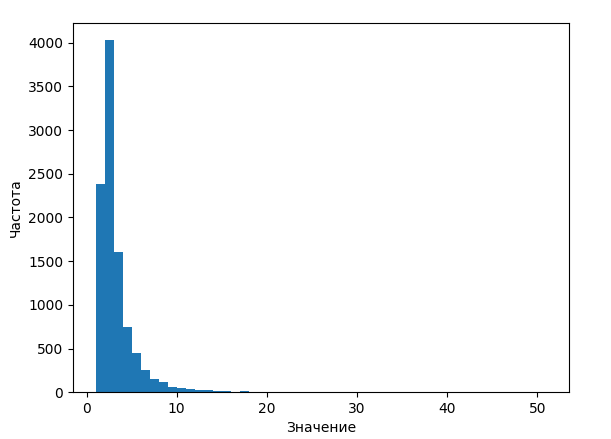
Используя стандартные нормальные случайные числа *Z*, применяют формулу

для получения случайных чисел, соответствующих логнормальному распределению.

### Параметры запуска программы

rnc.exe /f:rnd.dat /d:ln /p1:1 /p2:1,2





### Исходный текст программы

void distLN(string inFile, int p1, double p2) {

string str = "";

char el;

ifstream fin(inFile);

ofstream fout("ln.dat");

if (!fin.is\_open()) {

cout << "Файл " << inFile << " не найден! \n";

return;

}

double m = findMaxEl(inFile) + 1;

vector <double> nums;

while (fin.get(el)) { //Генерируем распределение

if (isdigit(el))

str = str + el;

else {

nums.push\_back(stoi(str) / m);

if (nums.size() == 2) {

double nrNum = p1 + p2 \* sqrt(-2 \* log(1 - nums[0])) \* cos(2 \* PI \* nums[1]);

fout << round(p1 + exp(p2 - nrNum)) << ",";

nrNum = p1 + p2 \* sqrt(-2 \* log(1 - nums[0])) \* sin(2 \* PI \* nums[1]);

fout << round(p1 + exp(p2 - nrNum)) << ",";

nums.clear();

}

str = "";

}

}

if (str != "" && nums.size() == 1) {

int nrNum = p1 + p2 \* sqrt(-2 \* log(1 - nums[0])) \* (int)cos(2 \* PI \* stoi(str) / m);

fout << round(p1 + exp(p2 - nrNum)) << ",";

nrNum = p1 + p2 \* sqrt(-2 \* log(1 - nums[0])) \* (int)sin(2 \* PI \* stoi(str) / m);

fout << round(p1 + exp(p2 - nrNum));

}

fin.close();

fout.close();

}

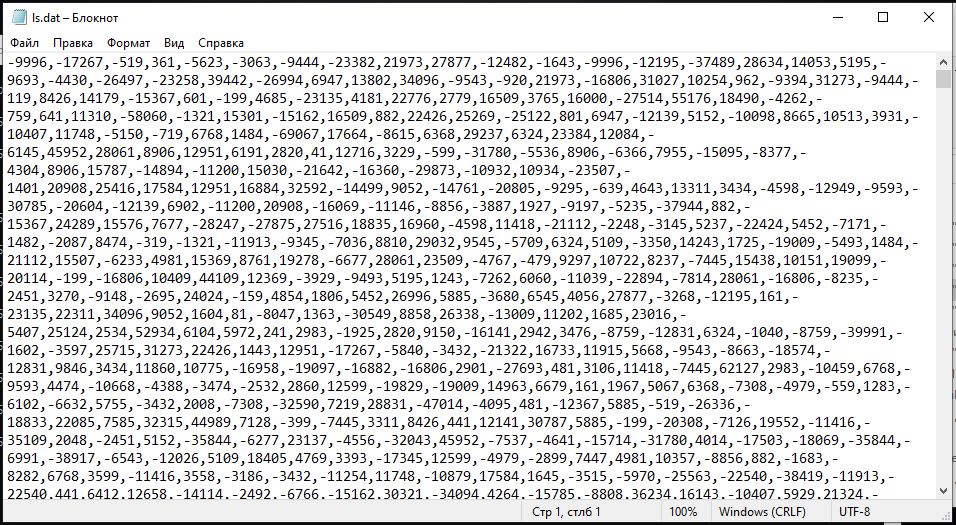
## Распределение 7. Логистическое распределение

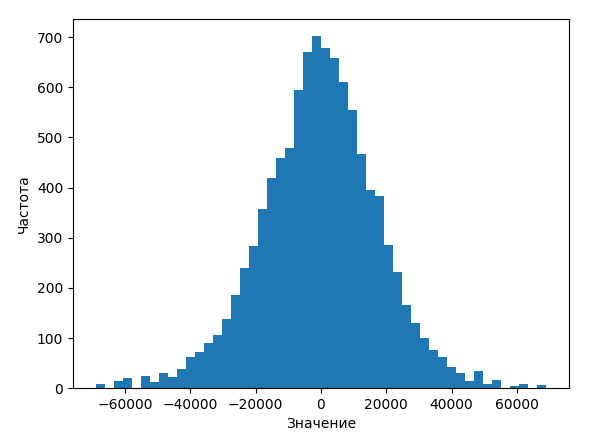
### Описание алгоритма

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

### Параметры запуска программы

rnc.exe /f:rnd.dat /d:ls /p1:1 /p2:10000





### Исходный текст программы

void distLS(string inFile, int p1, int p2) {

string str = "";

char el;

ifstream fin(inFile);

ofstream fout("ls.dat");

if (!fin.is\_open()) {

cout << "Файл " << inFile << "не найден! \n";

return;

}

double m = findMaxEl(inFile) + 1;

while (fin.get(el)) { //Генерируем распределение

if (isdigit(el))

str = str + el;

else {

double u = stoi(str) / m;

int res = 0;

if (str != "0")

res = round(p1 + p2 \* log(u / (1 - u)));

fout << res << ",";

str = "";

}

}

if (str != "") {

double u = stoi(str) / m;

fout << round(p1 + p2 \* log(u / (1 - u)));

}

fin.close();

fout.close();

}

## Распределение 8. Биномиальное распределение

### Описание алгоритма

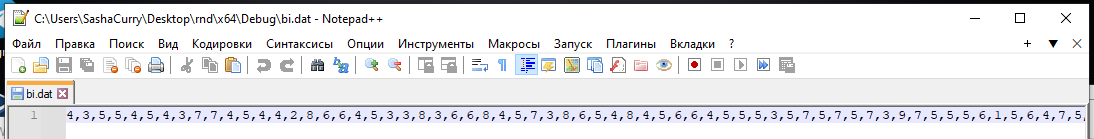
Вычисление функции распределения

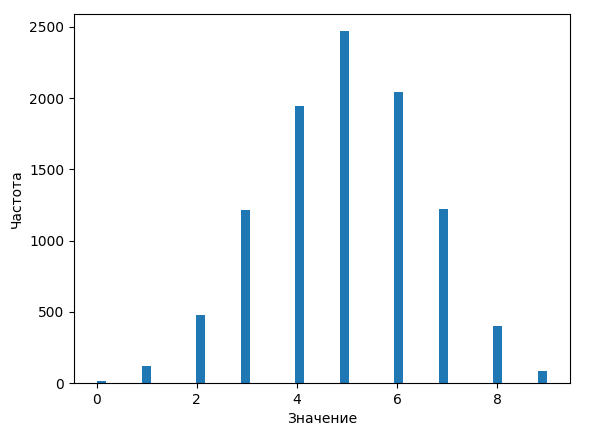
, *y* = 0, 1, …, *n*

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

### Параметры запуска программы

rnc.exe /f:rnd.dat /d:bi /p1:10 /p2:0,5





### Исходный текст программ

void distBI(string inFile, int n, double p) {

string str = "";

char el;

ifstream fin(inFile);

ofstream fout("bi.dat");

if (!fin.is\_open()) {

cout << "Файл " << inFile << "не найден! \n";

return;

}

vector <long long> bnc;

for (int k = 0; k <= n; k++) { //Вычисляем все возможные биномиальные коэффициенты

long long num = 1;

for (int i = k + 1; i <= n; i++)

num \*= i;

bnc.push\_back(num / fact(n - k));

}

double m = findMaxEl(inFile) + 1;

while (fin.get(el)) { //Генерируем распределение

if (isdigit(el))

str = str + el;

else {

double acc = 0, u = stoi(str) / m;

for (int k = 0; k < n; k++) {

acc += bnc[k] \* pow(p, k) \* pow((1 - p), n - k);

if (u <= acc) {

fout << k << ",";

break;

}

}

str = "";

}

}

if (str != "") {

double acc = 0, u = stoi(str) / m;

for (int k = 0; k < n; k++) {

acc += bnc[k] \* pow(p, k) \* pow((1 - p), n - k);

if (u <= acc) {

fout << k;

break;

}

}

}

fin.close();

fout.close();

}

## Приложение. Полный код программы

#include "iostream"

#include "string"

#include "vector"

#include "fstream"

#include "cmath"

#include "stdlib.h"

#include "time.h"

#include "utility"

#define PI 3.14

using namespace std;

int findMaxEl(string inFile) {

int maxEl = 0;

string str = "";

char el;

ifstream fin(inFile);

while (fin.get(el)) { //Ищем наибольшой элемент

if (isdigit(el))

str = str + el;

else if (stoi(str) > maxEl) {

maxEl = stoi(str);

str = "";

}

else

str = "";

}

if (str != "" && stoi(str) > maxEl) {

maxEl = stoi(str);

str = "";

}

fin.close();

return maxEl;

}

//Стандартное равномерное распределние с заданным интервалом

void distST(string inFile, int p1, int p2) {

string str = "";

char el;

ifstream fin(inFile);

if (!fin.is\_open()) {

cout << "Файл " << inFile << " не найден! **\n**";

return;

}

double m = findMaxEl(inFile) + 1;

ofstream fout("st.dat");

while (fin.get(el)) { //Генерируем распределение

if (isdigit(el))

str = str + el;

else {

fout << round(p1 + (stoi(str) / m) \* p2) << ",";

str = "";

}

}

if (str != "")

fout << round(p1 + (stoi(str) / m) \* p2);

fin.close();

fout.close();

}

//Треугольное распределение

void distTR(string inFile, int p1, int p2) {

string str = "";

char el;

ifstream fin(inFile);

if (!fin.is\_open()) {

cout << "Файл " << inFile << " не найден! **\n**";

return;

}

double m = findMaxEl(inFile) + 1;

int prev = 0;

while (fin.get(el)) { //Первый элемент последовательности

if (isdigit(el))

str = str + el;

else {

prev = stoi(str);

str = "";

break;

}

}

ofstream fout("tr.dat");

while (fin.get(el)) { //Генерируем распределение

if (isdigit(el))

str = str + el;

else {

fout << round(p1 + p2 \* (prev / m + stoi(str) / m - 1)) << ",";

prev = stoi(str);

str = "";

}

}

if (str != "")

fout << round(p1 + p2 \* (prev / m + stoi(str) / m - 1));

fin.close();

fout.close();

}

//Общее экспоненциальное распределение

void distEX(string inFile, int p1, int p2) {

string str = "";

char el;

ifstream fin(inFile);

ofstream fout("ex.dat");

if (!fin.is\_open()) {

cout << "Файл " << inFile << " не найден! **\n**";

return;

}

double m = findMaxEl(inFile) + 1;

while (fin.get(el)) { //Генерируем распределение

if (isdigit(el))

str = str + el;

else if (str == "0")

str = "";

else {

int res = -p2 \* log(stoi(str) / m) + p1;

fout << res << ",";

str = "";

}

}

if (str != "" && str != "0")

fout << int(-p2 \* log(stoi(str)) + p1);

fin.close();

fout.close();

}

//Нормальное распределение

void distNR(string inFile, int p1, int p2) {

string str = "";

char el;

ifstream fin(inFile);

ofstream fout("nr.dat");

if (!fin.is\_open()) {

cout << "Файл " << inFile << " не найден! **\n**";

return;

}

double m = findMaxEl(inFile) + 1;

vector <double> nums;

while (fin.get(el)) { //Генерируем распределение

if (isdigit(el))

str = str + el;

else {

nums.push\_back(stoi(str) / m);

if (nums.size() == 2) {

fout << round(p1 + p2 \* sqrt(-2 \* log(1 - nums[0])) \* cos(2 \* PI \* nums[1])) << ",";

fout << round(p1 + p2 \* sqrt(-2 \* log(1 - nums[0])) \* sin(2 \* PI \* nums[1])) << ",";

nums.clear();

}

str = "";

}

}

if (str != "" && nums.size() == 1) {

fout << round(p1 + p2 \* sqrt(-2 \* log(1 - nums[0])) \* cos(2 \* PI \* stoi(str) / m)) << ",";

fout << round(p1 + p2 \* sqrt(-2 \* log(1 - nums[0])) \* sin(2 \* PI \* stoi(str) / m));

}

fin.close();

fout.close();

}

//Гамма распределение

void distGM(string inFile, int p1, int p2) {

string str = "";

char el;

ifstream fin(inFile);

ofstream fout("gm.dat");

if (!fin.is\_open()) {

cout << "Файл " << inFile << "не найден! **\n**";

return;

}

double m = findMaxEl(inFile) + 1;

while (fin.get(el)) { //Генерируем распределение

if (isdigit(el))

str = str + el;

else {

fout << round(p1 - p2 \* log(1 - stoi(str) / m)) << ",";

str = "";

}

}

if (str != "") {

fout << round(p1 - p2 \* log(1 - stoi(str) / m));

}

fin.close();

fout.close();

}

//Логнормальное распределение

void distLN(string inFile, int p1, double p2) {

string str = "";

char el;

ifstream fin(inFile);

ofstream fout("ln.dat");

if (!fin.is\_open()) {

cout << "Файл " << inFile << " не найден! **\n**";

return;

}

double m = findMaxEl(inFile) + 1;

vector <double> nums;

while (fin.get(el)) { //Генерируем распределение

if (isdigit(el))

str = str + el;

else {

nums.push\_back(stoi(str) / m);

if (nums.size() == 2) {

double nrNum = p1 + p2 \* sqrt(-2 \* log(1 - nums[0])) \* cos(2 \* PI \* nums[1]);

fout << round(p1 + exp(p2 - nrNum)) << ",";

nrNum = p1 + p2 \* sqrt(-2 \* log(1 - nums[0])) \* sin(2 \* PI \* nums[1]);

fout << round(p1 + exp(p2 - nrNum)) << ",";

nums.clear();

}

str = "";

}

}

if (str != "" && nums.size() == 1) {

int nrNum = p1 + p2 \* sqrt(-2 \* log(1 - nums[0])) \* (int)cos(2 \* PI \* stoi(str) / m);

fout << round(p1 + exp(p2 - nrNum)) << ",";

nrNum = p1 + p2 \* sqrt(-2 \* log(1 - nums[0])) \* (int)sin(2 \* PI \* stoi(str) / m);

fout << round(p1 + exp(p2 - nrNum));

}

fin.close();

fout.close();

}

//Логистическое распределение

void distLS(string inFile, int p1, int p2) {

string str = "";

char el;

ifstream fin(inFile);

ofstream fout("ls.dat");

if (!fin.is\_open()) {

cout << "Файл " << inFile << "не найден! **\n**";

return;

}

double m = findMaxEl(inFile) + 1;

while (fin.get(el)) { //Генерируем распределение

if (isdigit(el))

str = str + el;

else {

double u = stoi(str) / m;

int res = 0;

if (str != "0")

res = round(p1 + p2 \* log(u / (1 - u)));

fout << res << ",";

str = "";

}

}

if (str != "") {

double u = stoi(str) / m;

fout << round(p1 + p2 \* log(u / (1 - u)));

}

fin.close();

fout.close();

}

//Факториал

long long fact(int n) {

long long res = 1;

for (int i = 2; i <= n; i++)

res \*= i;

return res;

}

//Биномиальное распределение

void distBI(string inFile, int n, double p) {

string str = "";

char el;

ifstream fin(inFile);

ofstream fout("bi.dat");

if (!fin.is\_open()) {

cout << "Файл " << inFile << "не найден! **\n**";

return;

}

vector <long long> bnc;

for (int k = 0; k <= n; k++) { //Вычисляем все возможные биномиальные коэффициенты

long long num = 1;

for (int i = k + 1; i <= n; i++)

num \*= i;

bnc.push\_back(num / fact(n - k));

}

double m = findMaxEl(inFile) + 1;

while (fin.get(el)) { //Генерируем распределение

if (isdigit(el))

str = str + el;

else {

double acc = 0, u = stoi(str) / m;

for (int k = 0; k < n; k++) {

acc += bnc[k] \* pow(p, k) \* pow((1 - p), n - k);

if (u <= acc) {

fout << k << ",";

break;

}

}

str = "";

}

}

if (str != "") {

double acc = 0, u = stoi(str) / m;

for (int k = 0; k < n; k++) {

acc += bnc[k] \* pow(p, k) \* pow((1 - p), n - k);

if (u <= acc) {

fout << k;

break;

}

}

}

fin.close();

fout.close();

}

//Главная функция

int main(int argc, char\* argv[]) {

setlocale(LC\_ALL, "ru");

srand(time(NULL));

string file = "rnd.dat";

string distribution = "bi";

double p1 = 10, p2 = 0.25;

//Считываем параметры

for (int i = 1; i < argc; i++) {

string str = argv[i];

switch (argv[i][1]) {

case 'f':

file = str.erase(0, 3);

break;

case 'd':

distribution = str.erase(0, 3);

break;

case 'p':

if (argv[i][2] == '1')

p1 = stoi(str.erase(0, 4));

else if (argv[i][2] == '2') {

int len = str.length();

char\* strP2 = new char[len - 4];

for (int i = 0; i <= len - 4; i++)

strP2[i] = str[i + 4];

p2 = atof(strP2);

}

else {

cout << "Некорректно задан параметр " << str << "!";

return 0;

}

break;

default:

cout << str << " - неверный параметр **\n**";

return 0;

}

}

//Преобразуем ПСЧ к распределению

if (distribution == "st")

distST(file, p1, p2);

else if (distribution == "tr")

distTR(file, p1, p2);

else if (distribution == "ex")

distEX(file, p1, p2);

else if (distribution == "nr")

distNR(file, p1, p2);

else if (distribution == "gm")

distGM(file, p1, p2);

else if (distribution == "ln")

distLN(file, p1, p2);

else if (distribution == "ls")

distLS(file, p1, p2);

else if (distribution == "bi")

distBI(file, p1, p2);

else

cout << "Неверный параметр /d **\n**";

return 0;

}