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
Яхина Шамиля Илдусовича

Преподаватель
Ассистент _____ Н.А. Артемова
подпись, дата

Саратов 2023

1 Генератор псевдослучайных чисел

Описание задания:

Создайте программу для генерации псевдослучайных величин следующими алгоритмами:

1. Линейный конгруэнтный метод;
2. Аддитивный метод;
3. Пятипараметрический метод;
4. Регистр сдвига с обратной связью (РСЛОС);
5. Нелинейная комбинация РСЛОС;
6. Вихрь Мерсенна;
7. RC4;
8. ГПСЧ на основе RSA;
9. Алгоритм Блюма-Блюма-Шуба.

1.1 Линейный конгруэнтный метод

Описание алгоритма:

Последовательность ПСЧ, получаемая по формуле:

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

называется *линейной конгруэнтной последовательностью (ЛКП)*.

В его основе лежит выбор четырех ключевых чисел:

— $m > 0$, модуль;
— $0 \leq a \leq m$, множитель;
— $0 \leq c \leq m$, приращение;
— $0 \leq X_0 \leq m$, начальное значение.

Инициализирующий вектор: модуль, множитель, приращение, начальное значение.

Параметры запуска программы:

prng.exe /n:16000 /g:lc /i:12960,1741,2731,1 /f:prngTestLc.txt

```
D:\cpp_projects\prng\prng>prng.exe /n:16000 /g:lc /i:12960,1741,2731,1 /f:prngTestLc.txt

The process of generating pseudo-random numbers:

10% completed
20% completed
30% completed
40% completed
50% completed
60% completed
70% completed
80% completed
90% completed
100% completed

Generation method: lc
Count of generated numbers: 16000
Full name of the output file: prngTestLc.txt
```

Рисунок 1 – Запуск программы

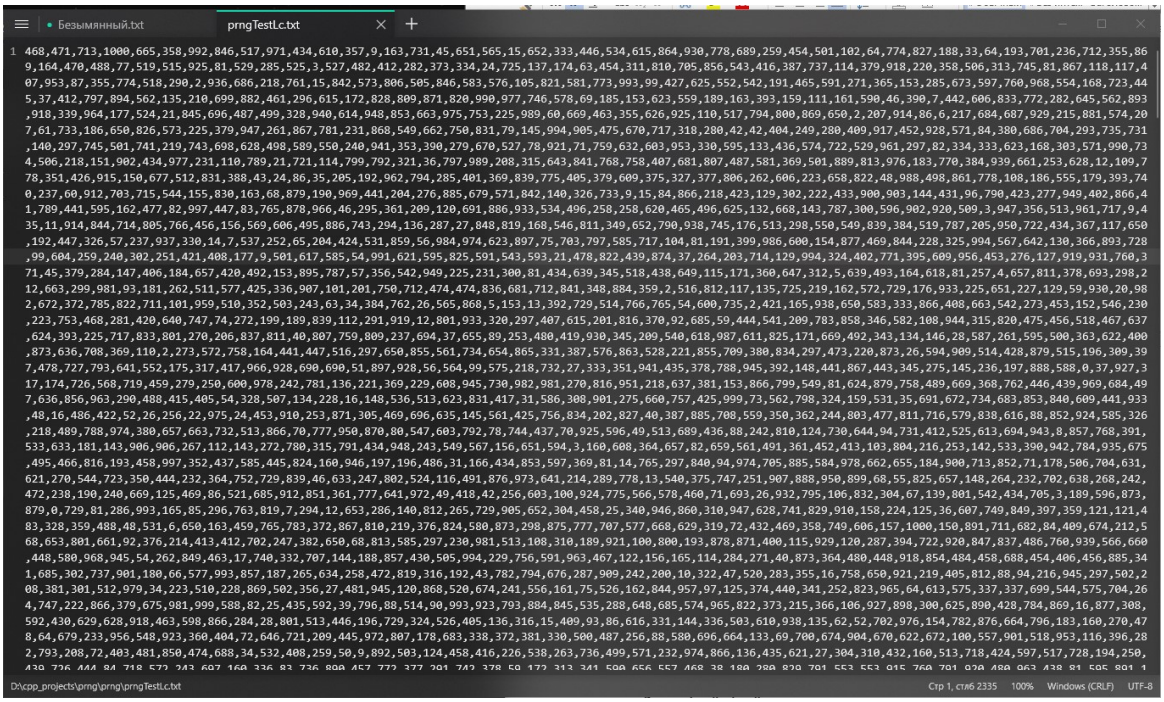

Рисунок 2 – Сгенерированная последовательность

Исходный код программы:

```
void lcFunc(string codeMethod, vector <int> genVec, int NumCount, const char*
↪  fileName, bool defaultVec) {
        long long m = 12960;
        long long a = 1741;
        long long c = 2731;
        long long x0 = 1;
        if (!defaultVec) {
                m = genVec[0];
                a = genVec[1];
                c = genVec[2];
                x0 = genVec[3];
        }
        if (m <= 0 || a > m || a < 0 || c > m || c < 0 || x0 > m || x0 < 0) {
```

```
                cout << "\nError. The parameters do not meet the requirements.\n";
                return;
        }
        if (nod(c, m) != 1) {
                cout << "\nThe parameters do not meet the requirements of Theorem
                ↪  3.1 (Item 1).\n";
        }
        if (!(kratP(a - 1, m))) {
                cout << "\nThe parameters do not meet the requirements of Theorem
                ↪  3.1 (Item 2).\n";
        }
        if (m % 4 == 0 && (a - 1) % 4 != 0) {
        cout << "\nThe parameters do not meet the requirements of Theorem 3.1 (Item
        ↪  3).\n";
    }
    vector<int> showsVector = progressVector(NumCount);
    int check_progress_index = 0;

    ofstream f;
    f.open(fileName, ios::out);
    for (int i = 0; i < NumCount; i++)
    {
            x0 = (x0 * a + c) % m;
            if (defaultVec)
            f << x0 % 1001 << ",";
            else
            f << x0 << ",";

            if (i == showsVector[check_progress_index]) {
                    cout << (check_progress_index + 1) * 10 << "% completed\n";
                    check_progress_index++;
            }
    }
    f.close();
    return;
}
```

1.2 Аддитивный метод

Описание алгоритма:

Последовательность определяется следующим образом:

$$X_n = (X_{n-k} + X_{n-j}) \mod m,\ j > k \geq 1$$

Инициализирующий вектор: модуль, младший индекс, старший индекс, последовательность начальных значений.

Параметры запуска программы:

prng.exe /n:16000 /g:add /i:8001,18,65,816,159,798,290,168,441,691,655,874,220,125,977,586,381,868,294,948,437,581,181,701,536,11,672,103,601,794,189,12,130,386,828,288,183,117,456,624,807,110,498,27,234,474,613,615,341,906,562,778,486,155,276,894,441,226,762,234,762,98,458,399,445,765,223,879 /f:prngTestAdd.txt

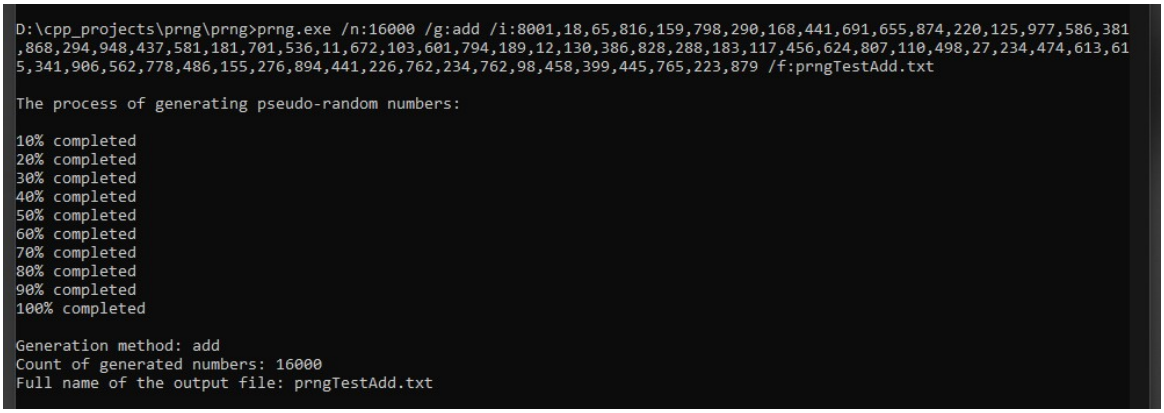

Рисунок 3 – Запуск программы

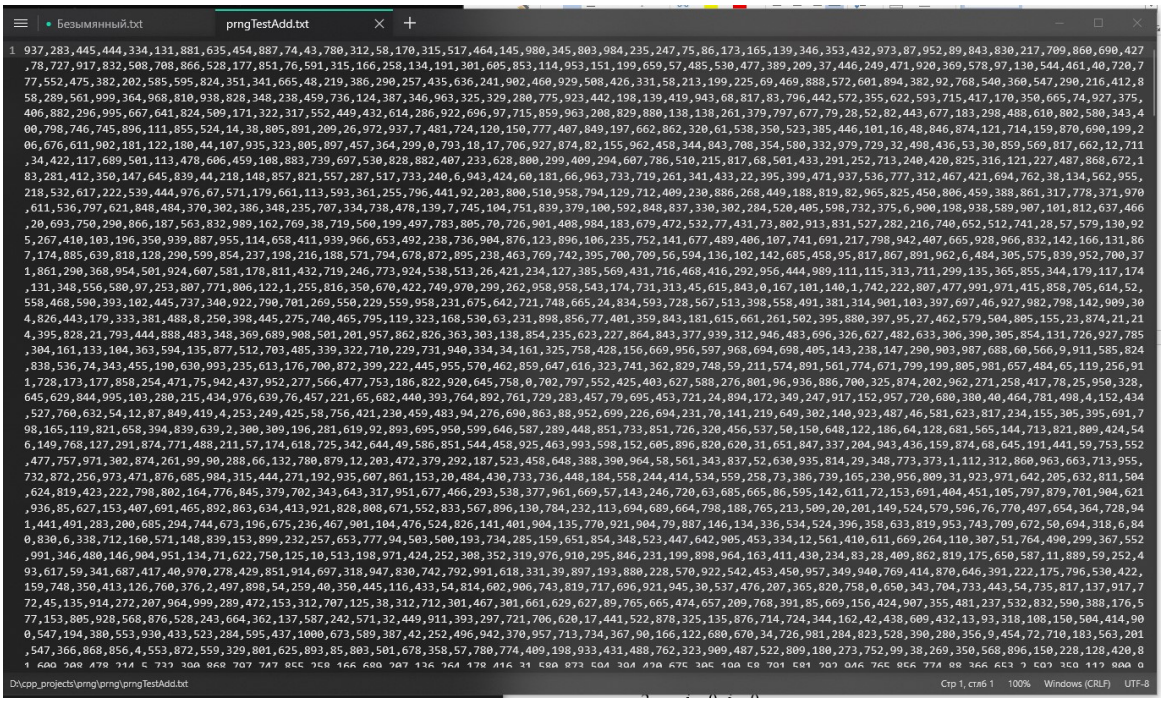

Рисунок 4 – Сгенерированная последовательность

Исходный код программы:

```
void addFunc(string codeMethod, vector <int> genVec, int NumCount, const char* fileName,
↪  bool defaultVec) {
        // Input: module, low index, high index, sequence of initial values
        long long m = 8001;
        long long k = 18;
        long long j = 65;
```

```
long long x0;
vector <long long> xNs;
if (!defaultVec) {
        m = genVec[0];
        k = genVec[1];
        j = genVec[2];
        for (int i = 3; i < genVec.size(); i++)
        {
                xNs.push_back(genVec[i]);
        }
}
else {
        xNs = { 816, 159, 798, 290, 168, 441, 691, 655, 874, 220, 125, 977, 586,
        ↪  381, 868, 294, 948, 437, 581, 181, 701, 536, 11, 672, 103, 601, 794,
        ↪  189, 12, 130, 386, 828, 288, 183, 117, 456, 624, 807, 110, 498, 27, 234,
        ↪  474, 613, 615, 341, 906, 562, 778, 486, 155, 276, 894, 441, 226, 762,
        ↪  234, 762, 98, 458, 399, 445, 765, 223, 879 }
        ;
}

if (k < 1 || k >= j || j < 1) {
        cout << "\nError. The parameters do not meet the requirements.\n";
        return;
}

vector<int> showsVector = progressVector(NumCount);
int check_progress_index = 0;

ofstream f;
f.open(fileName, ios::out);
long long maxkj = max(k, j);
if (xNs.size() < maxkj) {
        cout << "\nError. The parameters do not meet the requirements (The
        ↪  transferred initial values are insufficient).\n";
        return;
}
for (int i = maxkj + 1; i < NumCount + maxkj + 1; i++)
{
        x0 = (xNs[i - k] + xNs[i - j]) % m;
        xNs.push_back(x0);
        if (defaultVec)
        f << x0 % 1001 << ",";
        else
        f << x0 << ",";

        if (i == showsVector[check_progress_index]) {
                cout << (check_progress_index + 1) * 10 << "% completed\n";
                check_progress_index++;
```

```
                }
            }
            f.close();
            return;
}
```

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:

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

Инициализирующий вектор: p, q_1, q_2, q_3, w.

Параметры запуска программы:

prng.exe /n:16000 /g:5p /i:4253,1093,2254,3297,16 /f:prngTest5p.txt

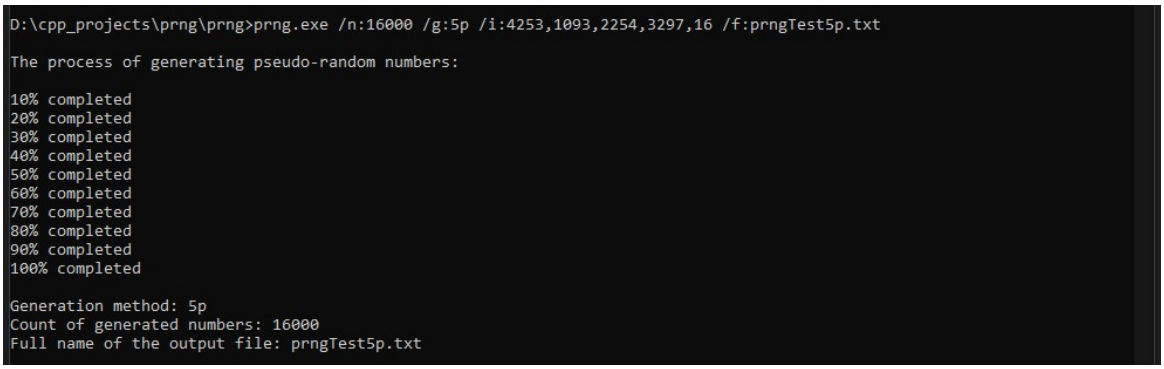

Рисунок 5 – Запуск программы

Рисунок 6 – Сгенерированная последовательность

Исходный код программы:

```
void fivePFunc(string codeMethod, vector <int> genVec, int NumCount, const char* fileName,
↪  bool defaultVec) {
        // Input: p, q1, q2, q3, w
        int p = 4253;
        long long q1 = 1093;
        long long q2 = 2254;
        long long q3 = 3297;
        int w = 16;
        if (!defaultVec) {
                p = genVec[0];
                q1 = genVec[1];
                q2 = genVec[2];
                q3 = genVec[3];
                w = genVec[4];
        }

        if (q1 >= p || q2 >= p || q3 >= p) {
                cout << "\nError. The parameters do not meet the requirements.\n";
                return;
        }

        int regLength = p;
        deque <bool> reg = gen_p_lc(p);
        vector<int> showsVector = progressVector(NumCount);
        int check_progress_index = 0;
```

```
        ofstream f;
        f.open(fileName, ios::out);
        for (int i = 0; i < NumCount; i++)
        {
                bool xNplusP = reg[regLength - q1 - 1] ^ reg[regLength - q2 - 1] ^
                ↪  reg[regLength - q3 - 1];
                reg.push_front(xNplusP);
                reg.pop_back();
                long long ReginDec = 0;
                for (int h = 0; h < w; h++)
                {
                        ReginDec *= 2;
                        ReginDec += reg[h];
                }
                if (defaultVec)
                f << ReginDec % 1001 << ",";
                else
                f << ReginDec << ",";

                if (i == showsVector[check_progress_index]) {
                        cout << (check_progress_index + 1) * 10 << "% completed\n";
                        check_progress_index++;
                }
        }
        f.close();
        return;
}
```

1.4 Регистр сдвига с обратной связью (РСЛОС)

Описание алгоритма:

Для натурального числа p и $c_1, c_2, \ldots, c_(p-1)$, принимающих значения 0 или 1 определяют рекуррентную формулу

$$X_{n+p} = a_{p-1}X_{n+p-1} + a_{p-2}X_{n+p-2} + \ldots + a_1X_{n+1} + X_n$$

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 $a_1, \ldots, a_{p-1}$.
3. Содержимое каждого i-го бита перемещается в $(i+1)$-й, $0 \leq i < p-1$.
4. В ячейку 0 записывается новое содержимое, вычисленное на шаге 2.

Инициализирующий вектор: двоичное представление вектора коэффициентов, начальное значение регистра.

Параметры запуска программы:

prng.exe /n:16000 /g:lfsr /i:1000000000101011,121 /f:prngTestLfsr.txt

```
D:\cpp_projects\prng\prng>prng.exe /n:16000 /g:lfsr /i:1000000000101011,121 /f:prngTestLfsr.txt

The process of generating pseudo-random numbers:

10% completed
20% completed
30% completed
40% completed
50% completed
60% completed
70% completed
80% completed
90% completed
100% completed

Generation method: lfsr
Count of generated numbers: 16000
Full name of the output file: prngTestLfsr.txt
```

Рисунок 7 – Запуск программы

Рисунок 8 – Сгенерированная последовательность

Исходный код программы:

```
void lfsrFunc(string codeMethod, vector <int> genVec, int NumCount, const char* fileName,
↪  bool defaultVec) {
        // Input: binary representation of the coefficient vector, initial value of the
        ↪  register
        if (defaultVec) {
                genVec = { 121, 14, 5, 3, 1, 0 }; // x0, coeffs (1000000000101011)
        }
```

```
int x0 = genVec[0];
int x0Length = log2(x0) + 1;
int regLength = x0Length - 1;
bool bigCoeff = false;
for (int i = 1; i < genVec.size(); i++)
{
        if (genVec[i] > regLength) {
                regLength = genVec[i] + 1;
                bigCoeff = true;
        }
}
if (x0 < 0) {
        cout << "\nError. The parameters do not meet the requirements.\n";
        return;
}

vector<int> showsVector = progressVector(NumCount);
int check_progress_index = 0;

deque<bool> reg;
if (!bigCoeff) {
        regLength++;
        reg = binary(x0, regLength);
}
else {
        reg.assign(regLength - x0Length, 0);
        deque<bool> regDop = binary(x0, x0Length);
        for (int i = 0; i < regDop.size(); i++)
        {
                reg.push_back(regDop[i]);
        }
}
ofstream f;
f.open(fileName, ios::out);
for (int i = 0; i < NumCount; i++)
{
        bool xNplusP = reg[regLength - genVec[1] - 1];
        for (int j = 2; j < genVec.size(); j++)
        {
                xNplusP ^= reg[regLength - genVec[j] - 1];
        }
        reg.push_front(xNplusP);
        reg.pop_back();
        long long ReginDec = 0;
        for (int h = 0; h < reg.size(); h++)
        {
                ReginDec *= 2;
                ReginDec += reg[h];
```

```
                }
                if (defaultVec)
                f << ReginDec % 1001 << ",";
                else
                f << ReginDec << ",";

                if (i == showsVector[check_progress_index]) {
                        cout << (check_progress_index + 1) * 10 << "% completed\n";
                        check_progress_index++;
                }
        }
        f.close();
        return;
}
```

1.5 Нелинейная комбинация РСЛОС

Описание алгоритма:

Последовательность получается объединением нелинейным образом трёх РСЛОС. Нелинейная функция генератора:

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

Инициализирующий вектор: двоичное представление векторов коэффициентов для $R1$, $R2$, $R3$.

Параметры запуска программы:

prng.exe /n:16000 /g:nfsr /i:1000000000011011,1000000000000011, 1000000000101101 /f:prngTestNfsr.txt

```
D:\cpp_projects\prng\prng>prng.exe /n:16000 /g:nfsr /i:1000000000011011,1000000000000011,1000000000101101 /f:prngTestNfs
r.txt

The process of generating pseudo-random numbers:

10% completed
20% completed
30% completed
40% completed
50% completed
60% completed
70% completed
80% completed
90% completed
100% completed

Generation method: nfsr
Count of generated numbers: 16000
Full name of the output file: prngTestNfsr.txt
```

Рисунок 9 – Запуск программы

Рисунок 10 – Сгенерированная последовательность

Исходный код программы:

```
void nfsrFunc(string codeMethod, vector <int> genVec, int NumCount, const char* fileName,
↪  vector <deque <bool>> nfsrRegs, bool defaultVec) {
        // Input: binary representation of the coefficient vectors for R1, R2, R3
        if (defaultVec) {
                genVec = { 5, 14, 4, 3, 1, 0, 3, 15, 1, 0, 5, 16, 5, 3, 2, 0 }; //
                ↪  size1,coeffs1,size2,coeffs2,size3,coeffs3
                ↪  (100000000011011,1000000000000011,10000000000101101)
                nfsrRegs = { {1,0,0,0,0,0,0,0,0,0,0,1,1,0,1,1},
                ↪  {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1}, {1,0,0,0,0,0,0,0,0,0,0,0,1,0,1,1,0,1}
                ↪  };
        }
        int nachR1 = 1;
        int endR1 = genVec[0] + 1;
        int nachR2 = endR1 + 1;
        int endR2 = genVec[endR1] + nachR2 + 1;
        int nachR3 = endR2 + 1;
        int endR3 = genVec.size();
        vector<int> startsRslos = { nachR1, nachR2, nachR3 };
        vector<int> endssRslos = { endR1, endR2, endR3 };
        vector<int> regLengths;
        for (int i = 0; i < 3; i++)
        {
                regLengths.push_back(nfsrRegs[i].size());
        }
        vector <deque<bool>> regs = nfsrRegs;
```

```
        vector<int> showsVector = progressVector(NumCount);
        int check_progress_index = 0;

        ofstream f;
        f.open(fileName, ios::out);
        for (int i = 0; i < NumCount; i++)
        {
                for (int j = 0; j < 3; j++)
                {
                        int startH = startsRslos[j];
                        int endH = endssRslos[j];
                        bool xNext = regs[j][regLengths[j] - genVec[startH] - 1];
                        for (int h = startH; h < endH; h++)
                        {
                                xNext ^= regs[j][regLengths[j] - genVec[h] - 1];
                        }
                        regs[j].push_front(xNext);
                        regs[j].pop_back();
                }
                vector <deque<bool>> regs2(2);
                regs2[0] = multRs(regs[0], regs[1]);
                regs2[1] = multRs(regs[1], regs[2]);
                deque<bool> resR = xorRs(xorRs(regs2[0], regs2[1]), regs[2]);

                long long ReginDec = 0;
                for (int h = 0; h < resR.size(); h++)
                {
                        ReginDec *= 2;
                        ReginDec += resR[h];
                }
                if (defaultVec)
                f << ReginDec % 1001 << ",";
                else
                f << ReginDec << ",";

                if (i == showsVector[check_progress_index]) {
                        cout << (check_progress_index + 1) * 10 << "% completed\n";
                        check_progress_index++;
                }
        }
        f.close();
        return;
}
```

1.6 Вихрь Мерсенна

Описание алгоритма:

Метод Вихрь Мерсенна позволяет генерировать последовательность дво-

ичных псевдослучайных целых w-битных чисел в соответствии с рекуррентной формулой:

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

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

X_n - w-битное двоичное целое число;

$(X_n^r | X_{n+1}^l)$ – двоичное целое число, полученное конкатенацией чисел X_n^r и X_{n+1}^l, когда первые $(w - r)$ битов взяты из X_n, а последние r битов из X_{n+1} в том же порядке;

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

XA – произведение, при вычислении которого сначала выполняют операцию $X >> 1$, если последний бит X равен 0, а затем, когда последний бит X равен 1, то вычисляют $XA = (X >> 1) \oplus A$

Инициализирующий вектор: модуль, начальное значение x.

Параметры запуска программы:

prng.exe /n:16000 /g:mt /i:10001,8191 /f:prngTestMt.txt

```
D:\cpp_projects\prng\prng>prng.exe /n:16000 /g:mt /i:10001,8191 /f:prngTestMt.txt

The process of generating pseudo-random numbers:

10% completed
20% completed
30% completed
40% completed
50% completed
60% completed
70% completed
80% completed
90% completed
100% completed

Generation method: mt
Count of generated numbers: 16000
Full name of the output file: prngTestMt.txt
```

Рисунок 11 – Запуск программы

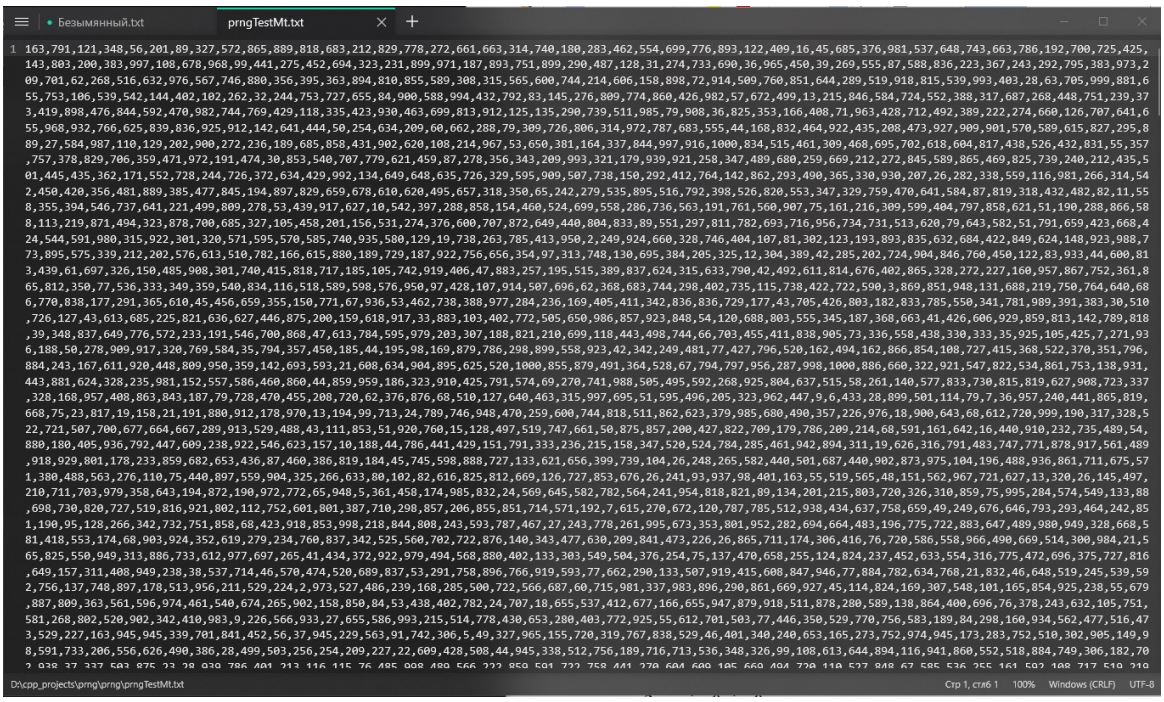

Рисунок 12 – Сгенерированная последовательность

Исходный код программы:

```
void mtFunc(string codeMethod, vector <int> genVec, int NumCount, const char* fileName, bool
↪  defaultVec) {
        // Input: module, initial value of x
        int p = 624;
        int w = 32;
        //string uBin = "10000000000000000000000000000000";
        //string hBin = "01111111111111111111111111111111";
        long long uDec = 2147483647;
        long long hDec = 2147483648;
        long long r = 31;
        long long q = 397;
        long long a = 2567483615;
        long long u = 11;
        long long s = 7;
        long long t = 15;
        long long l = 18;
        long long b = 2636928640;
        long long c = 4022730752;
        if (defaultVec) {
                genVec = { 10001, 8191 };
        }
        long long zMod = genVec[0];
        long long x0 = genVec[1];
        vector <long long> x_zns(1, x0);
        for (int i = 0; i < p; i++)
        {
```

```
        x_zns.push_back(abs(1812433253 * (x_zns[i - 1] ^ (x_zns[i - 1] >> 30)) +
        ↪  i));
    }

    vector<int> showsVector = progressVector(NumCount);
    int check_progress_index = 0;

    ofstream f;
    f.open(fileName, ios::out);
    int n = 0;
    for (int countN = 0; countN < NumCount; countN++)
    {
        long long xn = x_zns[n];
        long long xnPlus1 = x_zns[(n + 1) % p];
        long long yNew = (xn & uDec) | (xnPlus1 & hDec);
        int yNewLength = log2(yNew) + 1;
        deque<bool> yBin = binary(yNew, yNewLength);
        long long Xnew;
        yNew >>= 1;
        long long xnqp = x_zns[(n + q) % p];
        if (yBin[yBin.size() - 1] == 1)
        Xnew = xnqp ^ yNew ^ a;
        else
        Xnew = xnqp ^ yNew ^ 0;
        yNew = Xnew;
        yNew = yNew ^ (yNew >> u);
        yNew = yNew ^ ((yNew << s) & b);
        yNew = yNew ^ ((yNew << t) & c);
        long long zNew = yNew ^ (yNew >> l);
        if (defaultVec)
        f << (zNew % zMod) % 1001 << ",";
        else
        f << zNew % zMod << ",";

        if (countN == showsVector[check_progress_index]) {
            cout << (check_progress_index + 1) * 10 << "% completed\n";
            check_progress_index++;
        }
        x_zns[n] = Xnew;
        n = (n + 1) % p;
    }
    f.close();
    return;
}
```

1.7 RC4

Описание алгоритма:

1. Инициализация $S_i, i = 0, 1, \ldots, 255$.
   а) for i = 0 to 255: $S_i = i$
   б) j = 0;
   в) for i = 0 to 255: $j = (j + S_i + K_i)$ mod 256; $Swap(S_i, S_j)$
2. $i = 0, j = 0$.
3. Итерация алгоритма:
   а) $i = (i + 1) \mod 256$;
   б) $j = (j + S_i) \mod 256$;
   в) $Swap(S_i, S_j)$;
   г) $t = (S_i + S_j) \mod 256$;
   д) $K = S_t$.

Инициализирующий вектор: 256 начальных значений.

Параметры запуска программы:

prng.exe /n:16000 /g:rc4 /i:802,720,341,337,961,882,417,785,198,727,899,372, 374,425,556,615,813,768,840,183,893,568,73,387,18,436,182,125,806,899,485,607, 619,825,944,579,707,360,363,904,87,262,276,460,687,831,75,499,599,915,681,492, 483,754,878,500,189,60,624,994,959,109,600,577,934,544,156,640,903,519,544,990, 781,819,449,468,650,524,967,248,438,647,739,920,400,617,419,588,676,43,581,634, 151,181,211,84,724,367,723,627,886,267,617,667,85,65,134,735,589,100,983,26,747, 721,945,147,337,364,734,13,406,315,647,556,496,858,640,220,224,362,847,110,629, 463,776,713,528,909,448,116,9,430,141,755,151,86,901,488,449,635,500,855,950,147, 410,446,4,49,665,227,411,511,336,39,974,112,752,501,21,200,617,29,629,757,784,779, 843,684,266,292,319,766,146,269,912,556,714,916,605,378,142,15,889,478,54,862,590, 806,363,610,5,979,638,634,736,421,413,578,105,679,869,424,444,14,692,356,569,405, 271,173,783,413,188,671,891,242,533,480,48,895,89,53,873,727,686,608,147,98, 185, 252,776,54,675,220,67,366,576,636,771,846,808,553,259,996,224,149 /f:prngTestRc4.txt

```
D:\cpp_projects\prng\prng>prng.exe /n:16000 /g:rc4 /i:802,720,341,337,961,882,417,785,198,727,899,372,374,425,556,615,81
3,768,840,183,893,568,73,387,18,436,182,125,806,899,485,607,619,825,944,579,707,360,363,904,87,262,276,460,687,831,75,49
9,599,915,681,492,483,754,878,500,189,60,624,994,959,109,600,577,934,544,156,640,903,519,544,990,781,819,449,468,650,524
,967,248,438,647,739,920,400,617,419,588,676,43,581,634,151,181,211,84,724,367,723,627,886,267,617,667,85,65,134,735,589
,100,983,26,747,721,945,147,337,364,734,13,406,315,647,556,496,858,640,220,224,362,847,110,629,463,776,713,528,909,448,1
16,9,430,141,755,151,86,901,488,449,635,500,855,950,147,410,446,4,49,665,227,411,511,336,39,974,112,752,501,21,200,617,2
9,629,757,784,779,843,684,266,292,319,766,146,269,912,556,714,916,605,378,142,15,889,478,54,862,590,806,363,610,5,979,63
8,634,736,421,413,578,105,679,869,424,444,14,692,356,569,405,271,173,783,413,188,671,891,242,533,480,48,895,89,53,873,72
7,686,608,147,98,185,252,776,54,675,220,67,366,576,636,771,846,808,553,259,996,224,149 /f:prngTestRc4.txt

The process of generating pseudo-random numbers:

10% completed
20% completed
30% completed
40% completed
50% completed
60% completed
70% completed
80% completed
90% completed
100% completed

Generation method: rc4
Count of generated numbers: 16000
Full name of the output file: prngTestRc4.txt
```

Рисунок 13 – Запуск программы

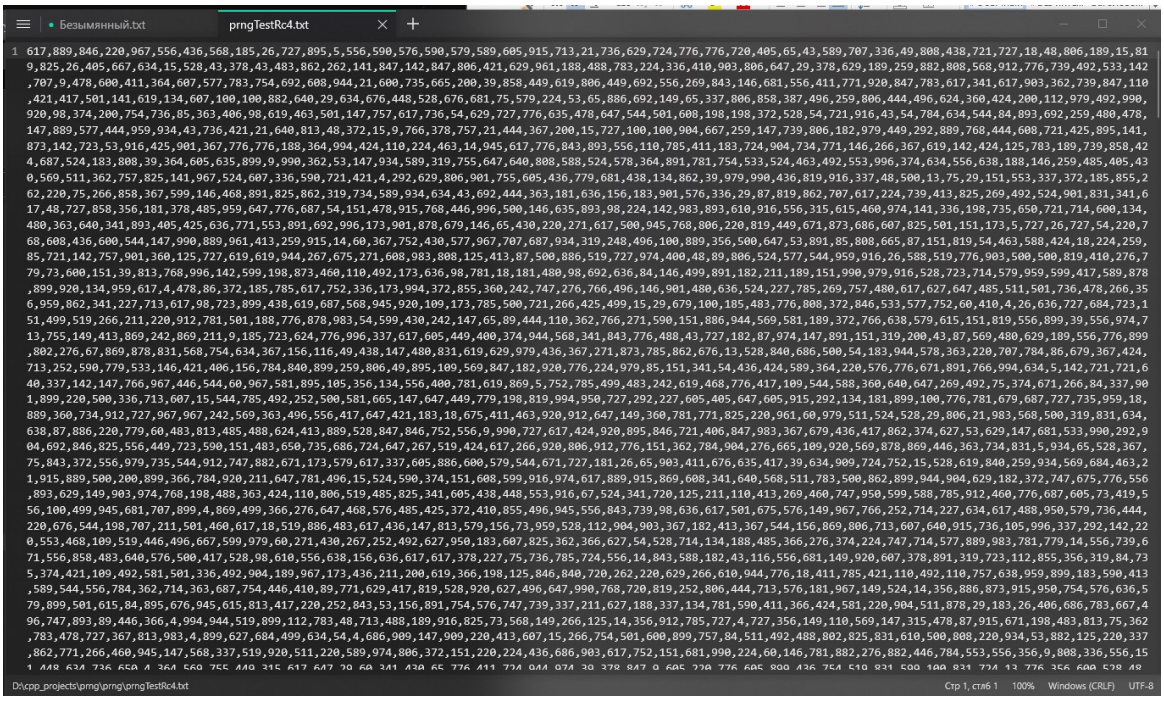

Рисунок 14 – Сгенерированная последовательность

Исходный код программы:

```
void rc4Func(string codeMethod, vector <int> genVec, int NumCount, const char* fileName,
↪  bool defaultVec) {
        // Input: 256 initial values
        if (defaultVec) {
```

```
        genVec = { 802, 720, 341, 337, 961, 882, 417, 785, 198, 727, 899, 372, 374,
        ↪  425, 556, 615, 813, 768, 840, 183, 893, 568, 73, 387, 18, 436, 182, 125,
        ↪  806, 899, 485, 607, 619, 825, 944, 579, 707, 360, 363, 904, 87, 262,
        ↪  276, 460, 687, 831, 75, 499, 599, 915, 681, 492, 483, 754, 878, 500,
        ↪  189, 60, 624, 994, 959, 109, 600, 577, 934, 544, 156, 640, 903, 519,
        ↪  544, 990, 781, 819, 449, 468, 650, 524, 967, 248, 438, 647, 739, 920,
        ↪  400, 617, 419, 588, 676, 43, 581, 634, 151, 181, 211, 84, 724, 367, 723,
        ↪  627, 886, 267, 617, 667, 85, 65, 134, 735, 589, 100, 983, 26, 747, 721,
        ↪  945, 147, 337, 364, 734, 13, 406, 315, 647, 556, 496, 858, 640, 220,
        ↪  224, 362, 847, 110, 629, 463, 776, 713, 528, 909, 448, 116, 9, 430, 141,
        ↪  755, 151, 86, 901, 488, 449, 635, 500, 855, 950, 147, 410, 446, 4, 49,
        ↪  665, 227, 411, 511, 336, 39, 974, 112, 752, 501, 21, 200, 617, 29, 629,
        ↪  757, 784, 779, 843, 684, 266, 292, 319, 766, 146, 269, 912, 556, 714,
        ↪  916, 605, 378, 142, 15, 889, 478, 54, 862, 590, 806, 363, 610, 5, 979,
        ↪  638, 634, 736, 421, 413, 578, 105, 679, 869, 424, 444, 14, 692, 356,
        ↪  569, 405, 271, 173, 783, 413, 188, 671, 891, 242, 533, 480, 48, 895, 89,
        ↪  53, 873, 727, 686, 608, 147, 98, 185, 252, 776, 54, 675, 220, 67, 366,
        ↪  576, 636, 771, 846, 808, 553, 259, 996, 224, 149 };
}
vector <int> k = genVec;
vector <int> s = genVec;
int j = 0;
for (int i = 0; i < 256; i++)
{
        j = (j + s[i] + k[i]) % 256;
        swap(s[i], s[j]);
}
int i = 0;
j = 0;

vector<int> showsVector = progressVector(NumCount);
int check_progress_index = 0;

ofstream f;
f.open(fileName, ios::out);
for (int countN = 0; countN < NumCount; countN++)
{
        i = (i + 1) % 256;
        j = (j + s[j]) % 256;
        swap(s[i], s[j]);
        int t = (s[i] + s[j]) % 256;
        if (defaultVec)
        f << s[t] % 1001 << ",";
        else
        f << s[t] << ",";

        if (countN == showsVector[check_progress_index]) {
                cout << (check_progress_index + 1) * 10 << "% completed\n";
```

```
                    check_progress_index++;
                }
            }
            f.close();
            return;
}
```

1.8 ГПСЧ на основе RSA

Описание алгоритма:

1. Инициализация простых чисел p, q и $n = pq$, а также $f = (p-1)(q-1)$. Выбрать случайное число e: $1 < e < f$, НОД(e, f) = 1.
2. Выбрать случайное целое x_0 из интервала $[1, n-1]$.
3. For i = 1 to w do
   а) $x_i \leftarrow x_{i-1}^e \mod n$.
   б) $z_i \leftarrow$ последний значащий бит x_i
4. Вернуть $z_1, \ldots, z_w$.

Инициализирующий вектор: модуль n, число e, начальное значение x.

Параметры запуска программы:

prng.exe /n:16000 /g:rsa /i:7191817,151,69 /f:prngTestRsa.txt

Рисунок 15 – Запуск программы

Рисунок 16 – Сгенерированная последовательность

Исходный код программы:

```
void rsaFunc(string codeMethod, vector <int> genVec, int NumCount, const char* fileName,
↪  bool defaultVec) {
        // Input: module n, number e, initial value of x; e satisfies conditions: 1 < e <
        ↪  (p-1)(q-1), NOD(e, (p-1)(q-1)) = 1, where p*q=n. x from the interval [1,n]
        if (defaultVec) {
                genVec = { 7191817, 151, 69 };
        }
        int n = genVec[0];
        int e = genVec[1];
        int x = genVec[2];
        int l = 20;

        if (e <= 1 || x < 1 || x > n - 1) {
                cout << "\nError. The parameters do not meet the requirements.\n";
                return;
        }

        vector<int> showsVector = progressVector(NumCount);
        int check_progress_index = 0;

        ofstream f;
        f.open(fileName, ios::out);
        for (int countN = 0; countN < NumCount; countN++)
        {
                string zs = "";
                for (int i = 0; i < l; i++)
```

```
                {
                        x = powmod(x, e, n);
                        char z = '0';
                        if (x % 2 != 0)
                        z = '1';
                        zs.push_back(z);
                }
                if (defaultVec)
                f << binToDec(zs) % 1001 << ",";
                else
                f << binToDec(zs) << ",";

                if (countN == showsVector[check_progress_index]) {
                        cout << (check_progress_index + 1) * 10 << "% completed\n";
                        check_progress_index++;
                }
        }
        f.close();
        return;
}
```

1.9 Алгоритм Блюм-Блюма-Шуба

Описание алгоритма:

На входе: Длина l.

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

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 $x_0 = x^2 \mod n$, которое будет начальным вектором.
3. For i = 1 to w do
   а) $x_i \leftarrow x_{i-1}^2 \mod n$.
   б) $z_i \leftarrow$ последний значащий бит x_i
4. Вернуть $z_1, \ldots, z_w$.

Инициализирующий вектор: начальное значение x (взаимно простое с n).

Параметры запуска программы:

prng.exe /n:16000 /g:bbs /i:8627 /f:prngTestBbs.txt

Рисунок 17 – Запуск программы

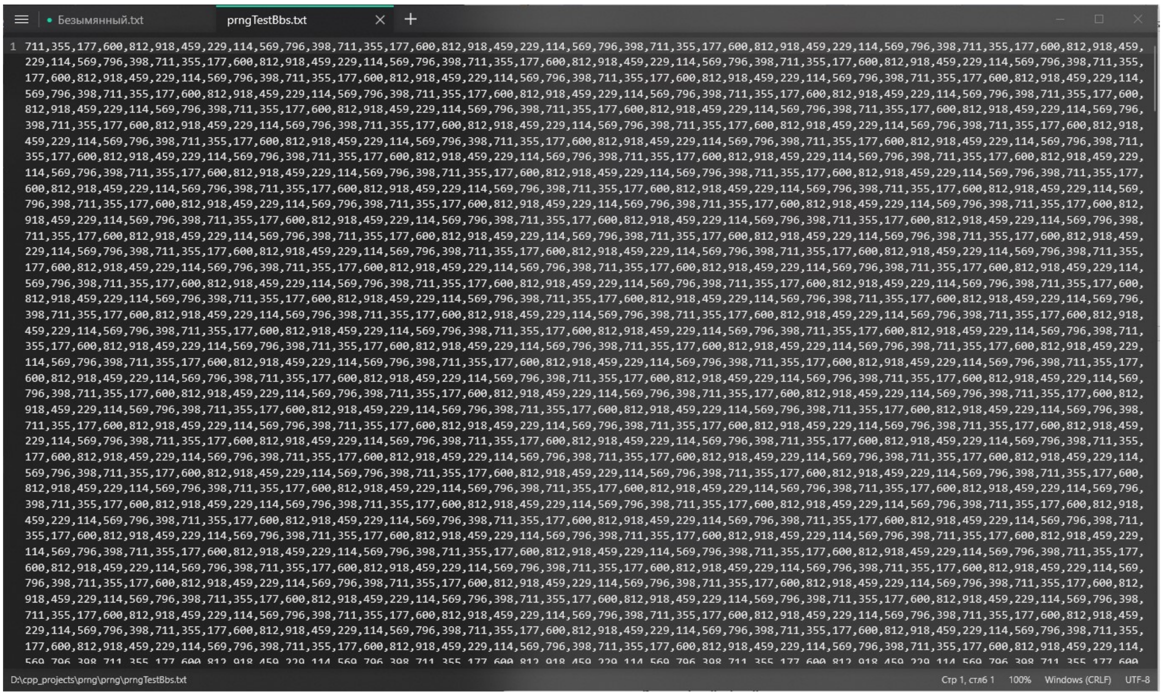

Рисунок 18 – Сгенерированная последовательность

Исходный код программы:

```
void bbsFunc(string codeMethod, vector <int> genVec, int NumCount, const char* fileName,
↪   bool defaultVec) {
        // Input: Initial value of x (mutually prime with n)
        if (defaultVec) {
                genVec = { 8627 };
        }
        int x0 = genVec[0];
        int n = 16637;
        int l = 10;

        if (nod(x0, n) != 1) {
                cout << "\nThe parameters do not meet the requirements (x and n are not
                ↪   mutually prime).\n";
        }
```

```
        vector<int> showsVector = progressVector(NumCount);
        int check_progress_index = 0;

        ofstream f;
        f.open(fileName, ios::out);
        for (int countN = 0; countN < NumCount; countN++)
        {
                x0 = (x0 * x0) % n;
                string zs = "";
                for (int i = 0; i < l; i++)
                {
                        x0 = (x0 * x0) % n;
                        char z = '0';
                        if (x0 % 2 != 0)
                        z = '1';
                        zs.push_back(z);
                }
                if (defaultVec)
                f << binToDec(zs) % 1001 << ",";
                else
                f << binToDec(zs) << ",";

                if (countN == showsVector[check_progress_index]) {
                        cout << (check_progress_index + 1) * 10 << "% completed\n";
                        check_progress_index++;
                }
        }
        f.close();
        return;
}
```

2 Преобразование ПСЧ к заданному распределению

Описание задания:

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

1. Стандартное равномерное с заданным интервалом;
2. Треугольное распределение;
3. Общее экспоненциальное распределение;
4. Нормальное распределение;
5. Гамма распределение (для параметра c=k);
6. Логнормальное распределение;
7. Логистическое распределение;
8. Биномиальное распределение.

Для преобразования последовательности ПСЧ в другую последовательность ПСЧ с заданным распределением использовалась последовательность, полученная с помощью вихря Мерсенна:

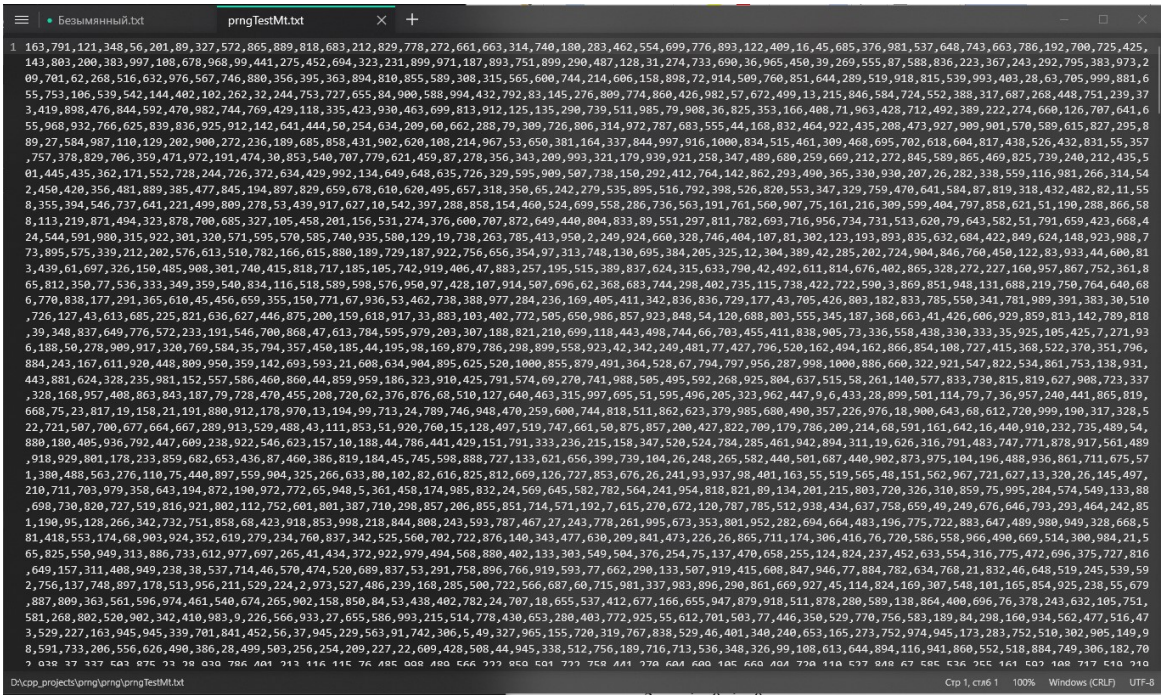

Рисунок 19 – Последовательность, полученная с помощью вихря Мерсенна

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

2.1 Стандартное равномерное с заданным интервалом

Описание алгоритма:

Если число U соответствует стандартному равномерному распределению, то равномерное случайное число должно быть получено в соответствии со следующей формулой:

$$Y = bU + a$$

Параметры запуска программы:

rnc.exe /d:st /f:prngTestMt.txt /p1:75 /p2:200

```
D:\cpp_projects\rnc\rnc>rnc.exe /d:st /f:prngTestMt.txt /p1:75 /p2:200

Distribution method: st
Parameter 1: 75
Parameter 2: 200
Input File: prngTestMt.txt
Output File: stOutput.dat
```

Рисунок 20 – Запуск программы

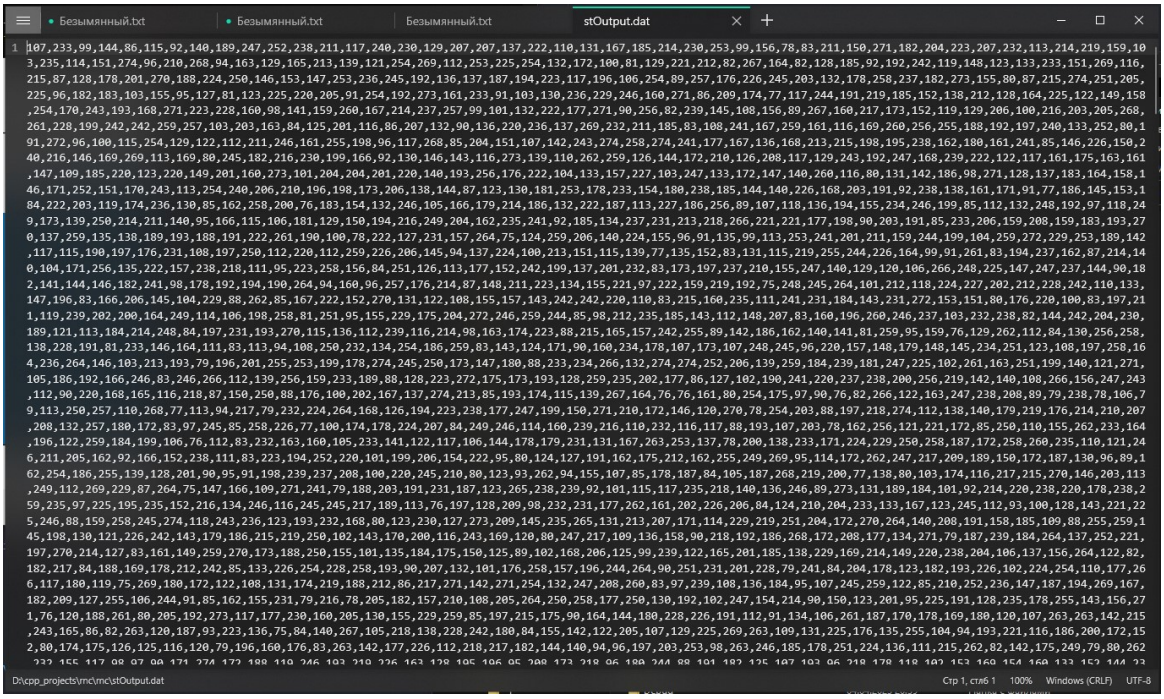

Рисунок 21 – Последовательность с заданным распределением

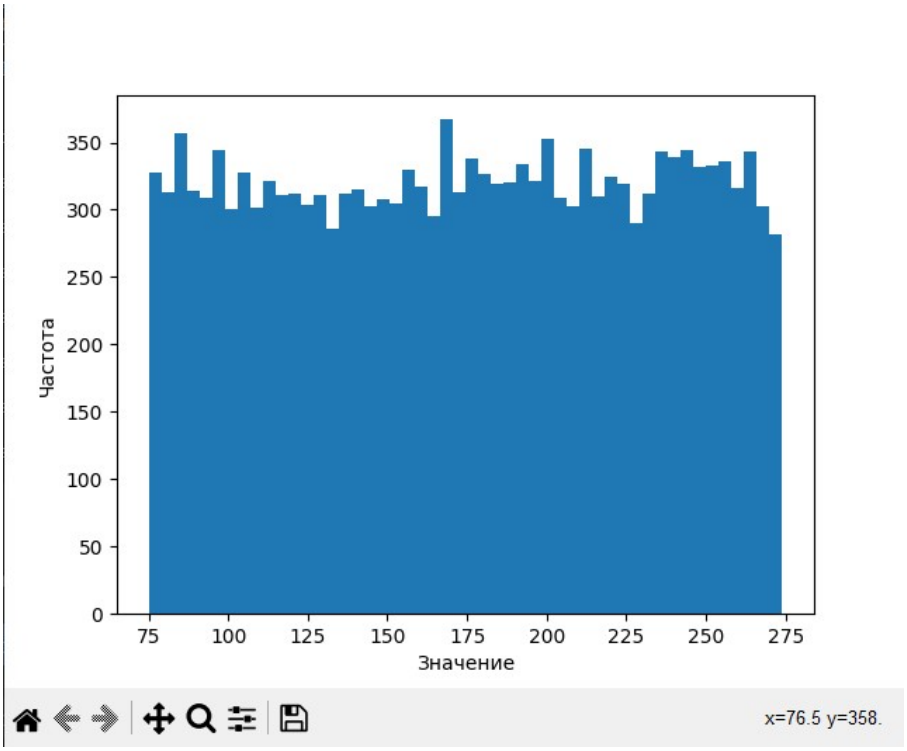

Рисунок 22 – График распределения

Исходный код программы:

```
vector <double> stFunc(string distributionMethod, string inputFileName, string parameter1,
↪  string parameter2, bool defaultVec, bool& errorMessage) {
        ifstream inFile(inputFileName);
        string lineFile;
        vector <double> allElems;
        if (!inFile.is_open()) {
                cout << "Ошибка открытия файла " << inputFileName << endl;
                errorMessage = true;
                return allElems;
        }
        string getlineElem;
        getline(inFile, getlineElem);
        stringstream ss(getlineElem);
        while (getline(ss, getlineElem, ',')) {
                double intElem = stod(getlineElem);
                allElems.push_back(intElem);
        }
        inFile.close();

        vector <double> allU = XtoU(allElems);
        double a = stod(parameter1);
        double b = stod(parameter2);
        for (int i = 0; i < allU.size(); i++)
        {
                allU[i] = (b * allU[i]) + a;
```

```
        }

        return allU;
}
```

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:

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

Параметры запуска программы:

rnc.exe /d:tr /f:prngTestMt.txt /p1:1 /p2:1000

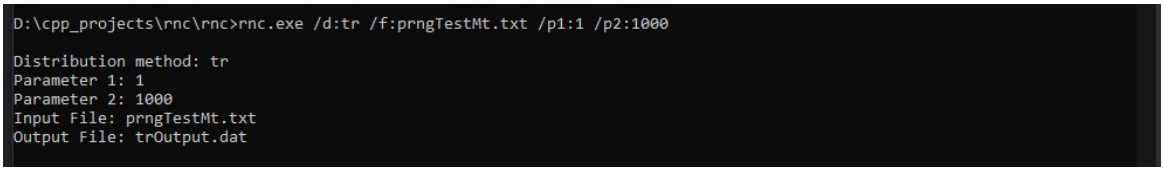

Рисунок 23 – Запуск программы

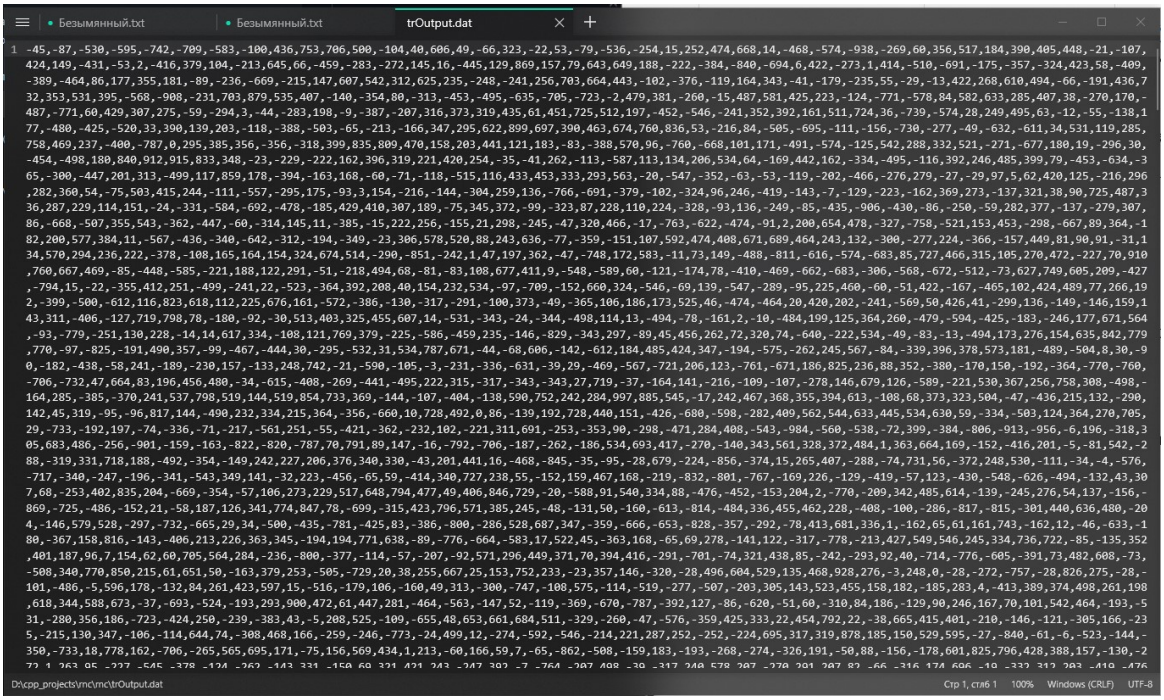

Рисунок 24 – Последовательность с заданным распределением

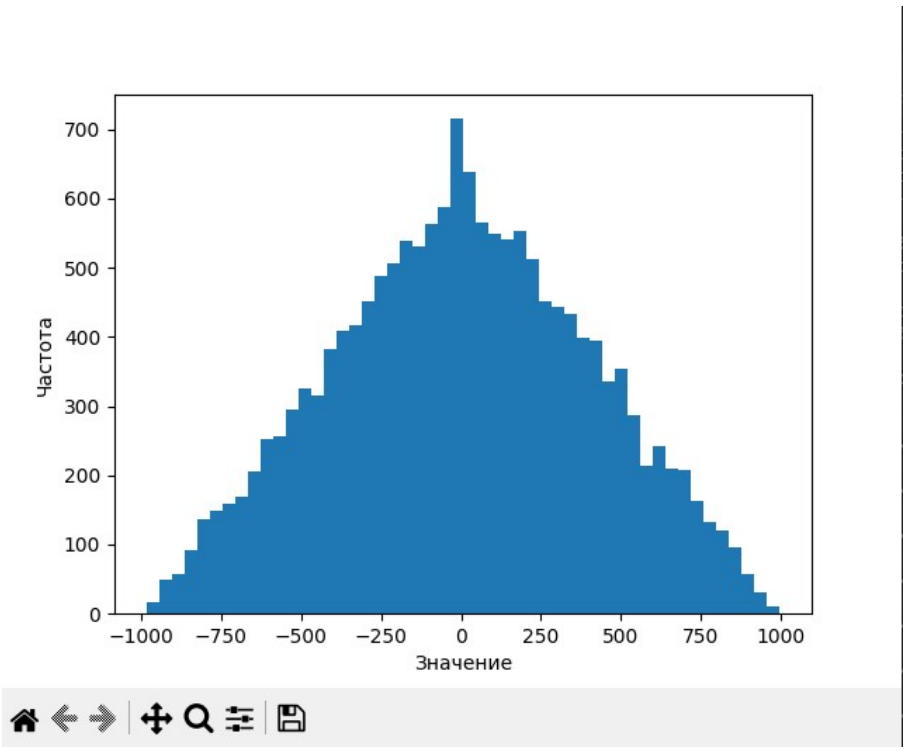

Рисунок 25 – График распределения

Исходный код программы:

```
vector <double> trFunc(string distributionMethod, string inputFileName, string parameter1,
↪  string parameter2, bool defaultVec, bool& errorMessage) {
        ifstream inFile(inputFileName);
        string lineFile;
        vector <double> allElems;
        if (!inFile.is_open()) {
                cout << "Ошибка открытия файла " << inputFileName << endl;
                errorMessage = true;
                return allElems;
        }
        string getlineElem;
        getline(inFile, getlineElem);
        stringstream ss(getlineElem);
        while (getline(ss, getlineElem, ',')) {
                double intElem = stod(getlineElem);
                allElems.push_back(intElem);
        }
        inFile.close();

        vector <double> allU = XtoU(allElems);
        double a = stod(parameter1);
        double b = stod(parameter2);
        for (int i = 0; i < allU.size() - 1; i++)
        {
                allU[i] = a + (b * (allU[i] + allU[i + 1] - 1));
```

```
        }

        return allU;
}
```

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:

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

Параметры запуска программы:

rnc.exe /d:ex /f:D:prngTestMt.txt /p1:7 /p2:12

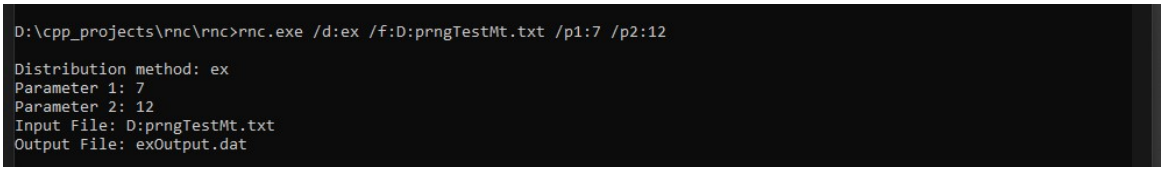

Рисунок 26 – Запуск программы

Рисунок 27 – Последовательность с заданным распределением

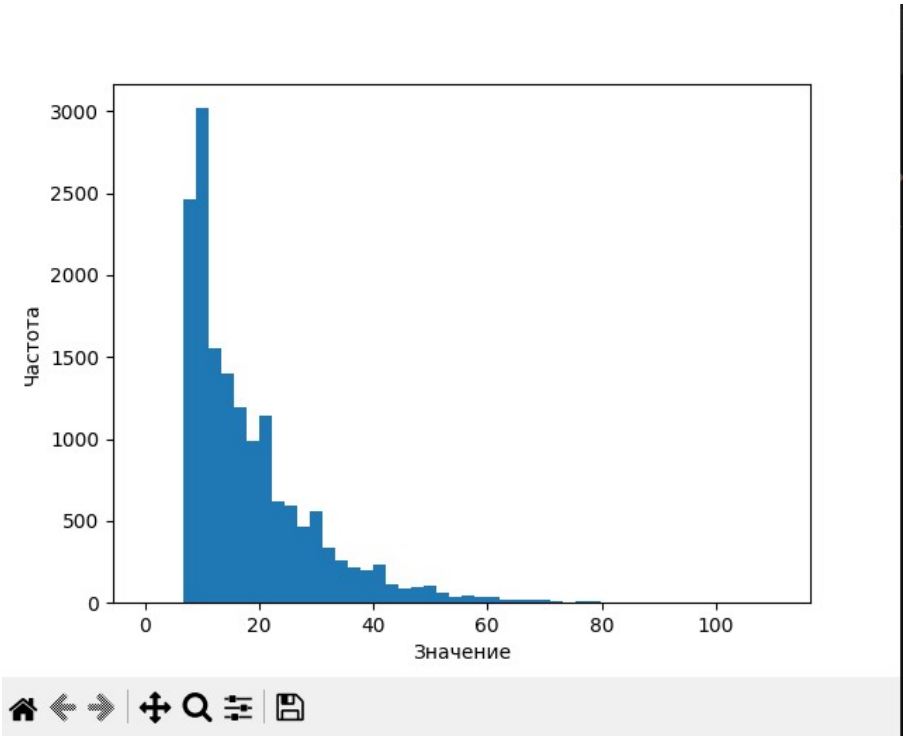

Рисунок 28 – График распределения

Исходный код программы:

```
vector <double> exFunc(string distributionMethod, string inputFileName, string parameter1,
↪  string parameter2, bool defaultVec, bool& errorMessage) {
        ifstream inFile(inputFileName);
        string lineFile;
        vector <double> allElems;
        if (!inFile.is_open()) {
                cout << "Ошибка открытия файла " << inputFileName << endl;
                errorMessage = true;
                return allElems;
        }
        string getlineElem;
        getline(inFile, getlineElem);
        stringstream ss(getlineElem);
        while (getline(ss, getlineElem, ',')) {
                double intElem = stod(getlineElem);
                allElems.push_back(intElem);
        }
        inFile.close();

        vector <double> allU = XtoU(allElems);
        double a = stod(parameter1);
        double b = stod(parameter2);
        for (int i = 0; i < allU.size() - 1; i++)
        {
                if (allU[i] == 0) {
```

```
                    cout << "Одно из случайных чисел U = 0. Ошибка в вычислении ln(U)"
                    ↪  << endl;
                    errorMessage = true;
                    return allU;
                }
                allU[i] = ((b * (-1)) * log(allU[i])) + a;
            }

            return allU;
}
```

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

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

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

Параметры запуска программы:

rnc.exe /d:nr /f:D:prngTestMt.txt /p1:1 /p2:1000

```
D:\cpp_projects\rnc\rnc>rnc.exe /d:nr /f:D:prngTestMt.txt /p1:1 /p2:1000

Distribution method: nr
Parameter 1: 1
Parameter 2: 1000
Input File: D:prngTestMt.txt
Output File: nrOutput.dat
```

Рисунок 29 – Запуск программы

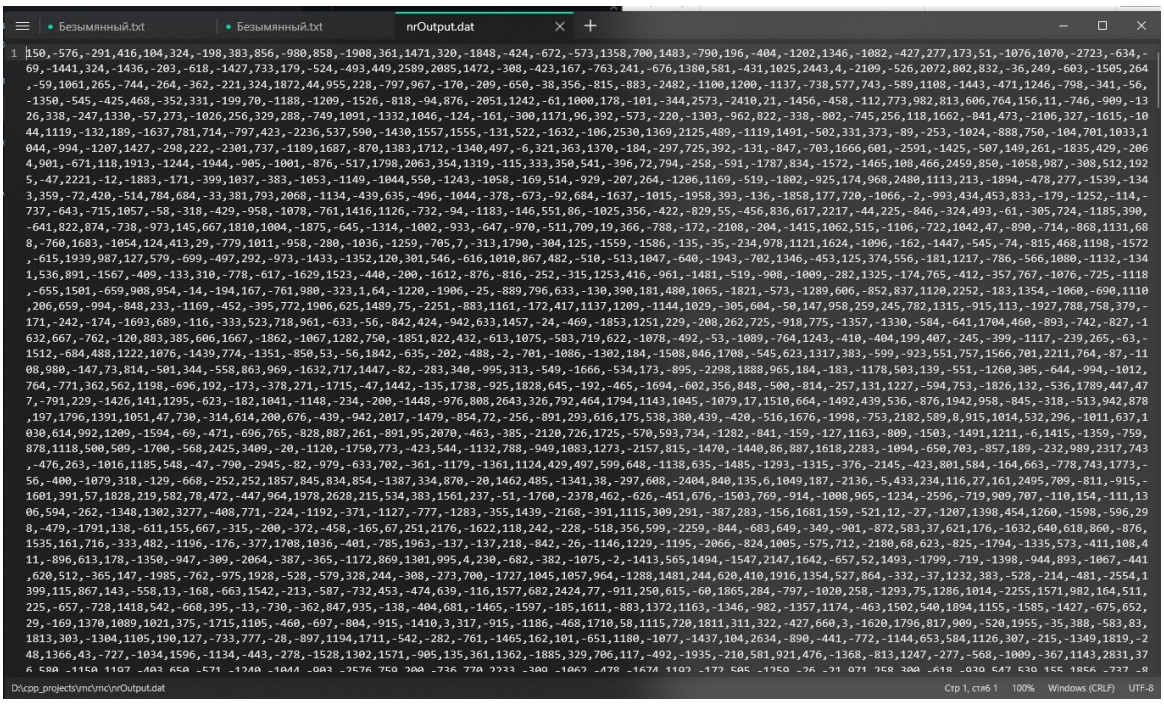

Рисунок 30 – Последовательность с заданным распределением

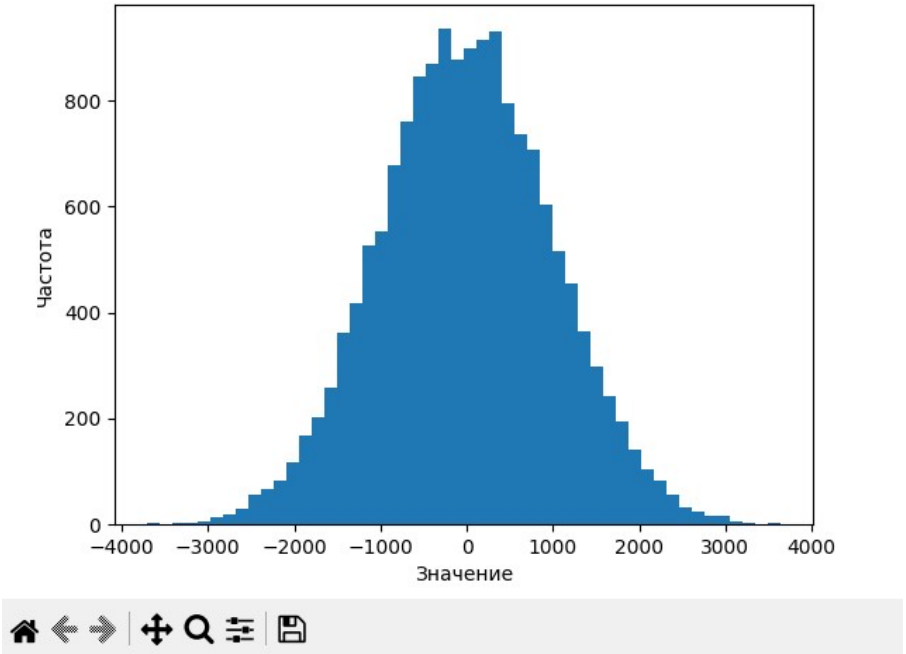

Рисунок 31 – График распределения

Исходный код программы:

```
vector <double> nrFunc(string distributionMethod, string inputFileName, string parameter1,
↪  string parameter2, bool defaultVec, bool& errorMessage) {
        ifstream inFile(inputFileName);
        string lineFile;
```

```
        vector <double> allElems;
        if (!inFile.is_open()) {
                cout << "Ошибка открытия файла " << inputFileName << endl;
                errorMessage = true;
                return allElems;
        }
        string getlineElem;
        getline(inFile, getlineElem);
        stringstream ss(getlineElem);
        while (getline(ss, getlineElem, ',')) {
                double intElem = stod(getlineElem);
                allElems.push_back(intElem);
        }
        inFile.close();

        if (allElems.size() % 2 != 0)
        allElems.erase(allElems.end() - 1);

        vector <double> allU = XtoU(allElems);
        double mu = stod(parameter1);
        double sigma = stod(parameter2);
        for (int i = 0; i < allU.size(); i += 2)
        {
                if (allU[i] == 1) {
                        cout << "Одно из случайных чисел U = 1. Ошибка в вычислении ln(1 -
                        ↪  U)" << endl;
                        errorMessage = true;
                        return allU;
                }
                double U1 = allU[i];
                double U2 = allU[i + 1];
                allU[i] = mu + (sigma * sqrt(-2 * log(1 - U1)) * cos(2 * M_PI * U2)); // Z1
                allU[i + 1] = mu + (sigma * sqrt(-2 * log(1 - U1)) * sin(2 * M_PI * U2)); //
                ↪  Z2
        }

        return allU;
}
```

2.5 Гамма распределение

Описание алгоритма:

Используя независимую равномерную случайную величину U_1, т.к. рассматривается случай $(c = 1)$, применяем формулу

$$Y = a - b \ln(1 - U_1)$$

Параметры запуска программы:

rnc.exe /d:gm /f:D:prngTestMt.txt /p1:7 /p2:12

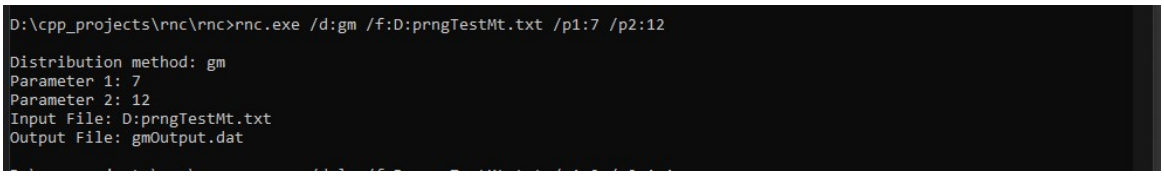

Рисунок 32 – Запуск программы

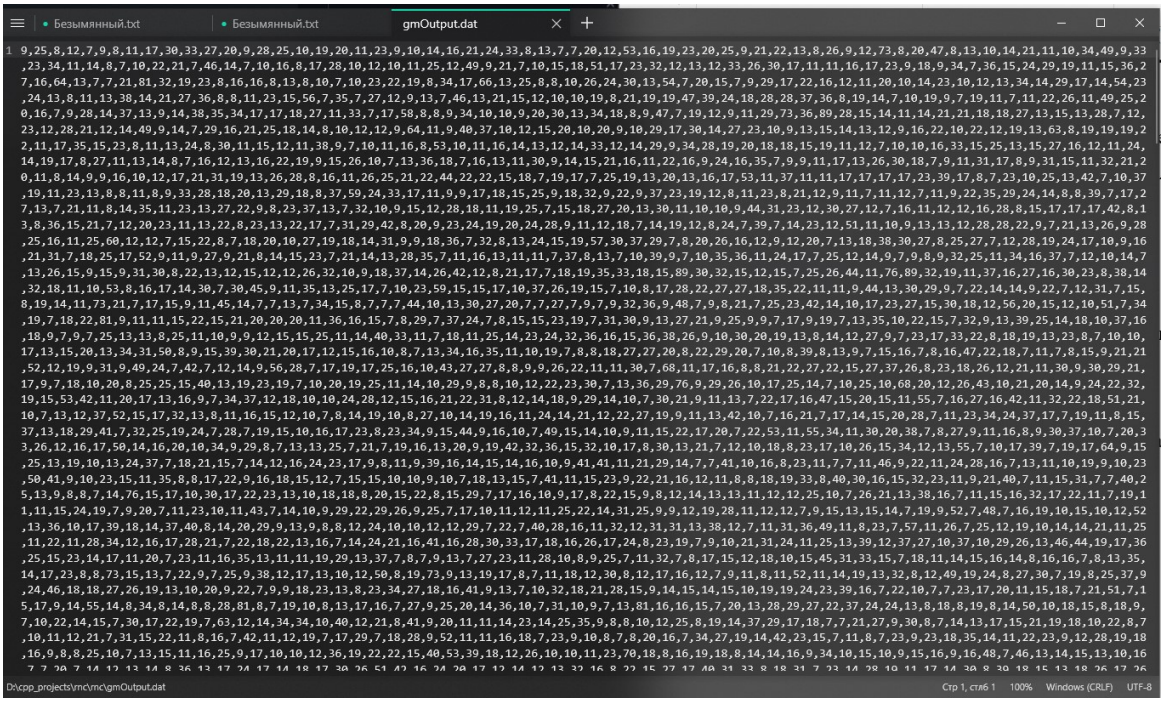

Рисунок 33 – Последовательность с заданным распределением

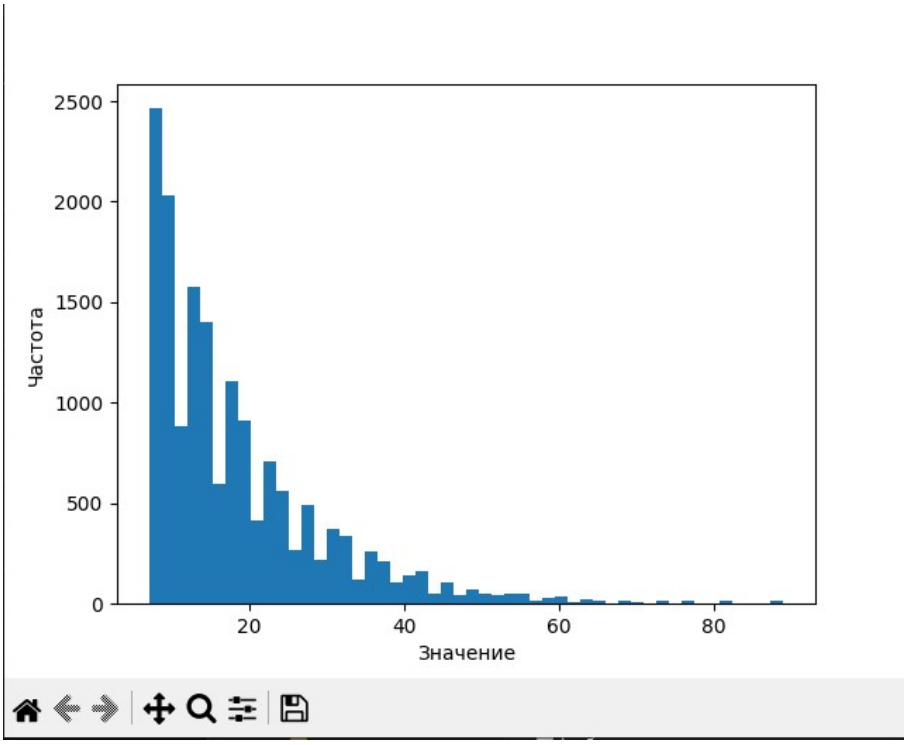

Рисунок 34 – График распределения

Исходный код программы:

```
vector <double> gmFunc(string distributionMethod, string inputFileName, string parameter1,
↪  string parameter2, bool defaultVec, bool& errorMessage) {
        ifstream inFile(inputFileName);
        string lineFile;
        vector <double> allElems;
        if (!inFile.is_open()) {
                cout << "Ошибка открытия файла " << inputFileName << endl;
                errorMessage = true;
                return allElems;
        }
        string getlineElem;
        getline(inFile, getlineElem);
        stringstream ss(getlineElem);
        while (getline(ss, getlineElem, ',')) {
                double intElem = stod(getlineElem);
                allElems.push_back(intElem);
        }
        inFile.close();

        vector <double> allU = XtoU(allElems);
        double a = stod(parameter1);
        double b = stod(parameter2);
        for (int i = 0; i < allU.size(); i++)
        {
                if (allU[i] == 1) {
```

```
                cout << "Одно из случайных чисел U = 1. Ошибка в вычислении ln(1 -
                ↪  U)" << endl;
                errorMessage = true;
                return allU;
            }
            allU[i] = a - (b * log(1 - allU[i]));
        }

        return allU;
}
```

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.

Параметры запуска программы:

rnc.exe /d:ln /f:D:prngTestMt.txt /p1:2 /p2:1,1

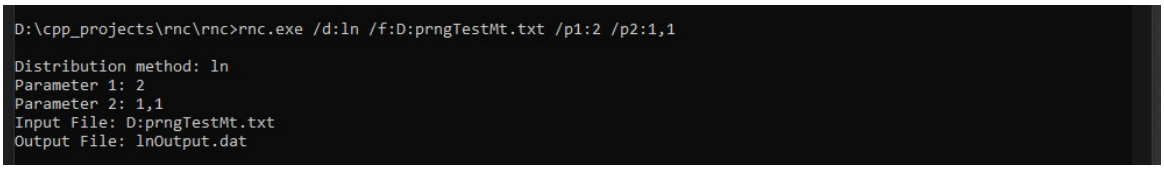

Рисунок 35 – Запуск программы

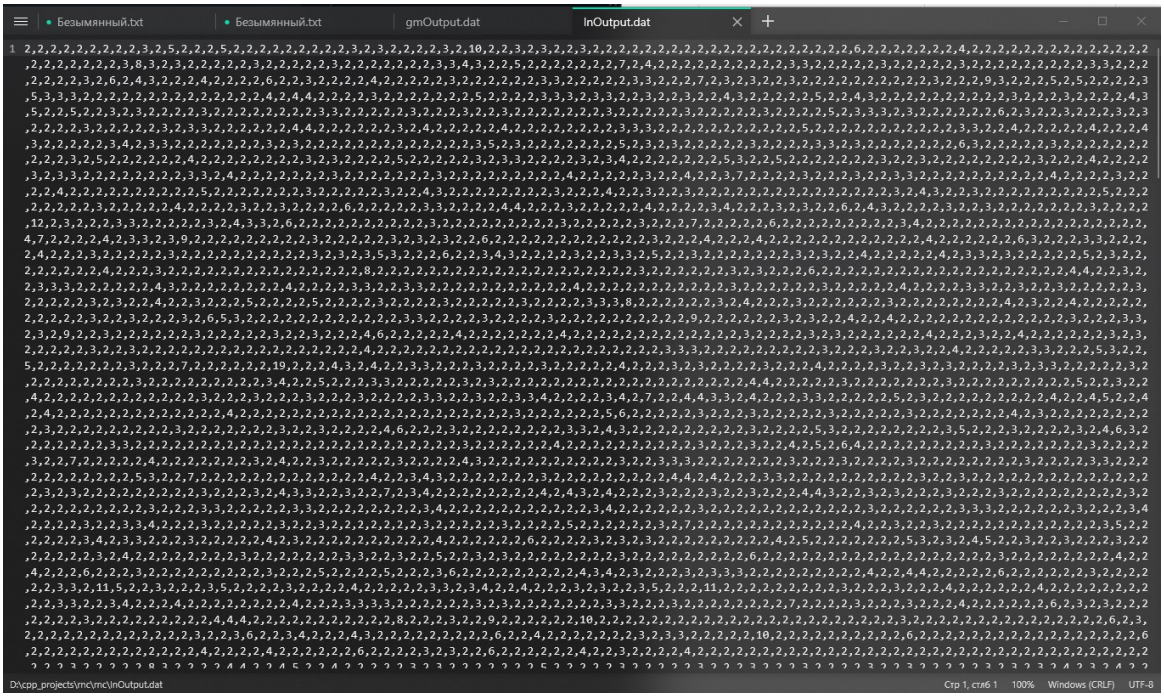

Рисунок 36 – Последовательность с заданным распределением

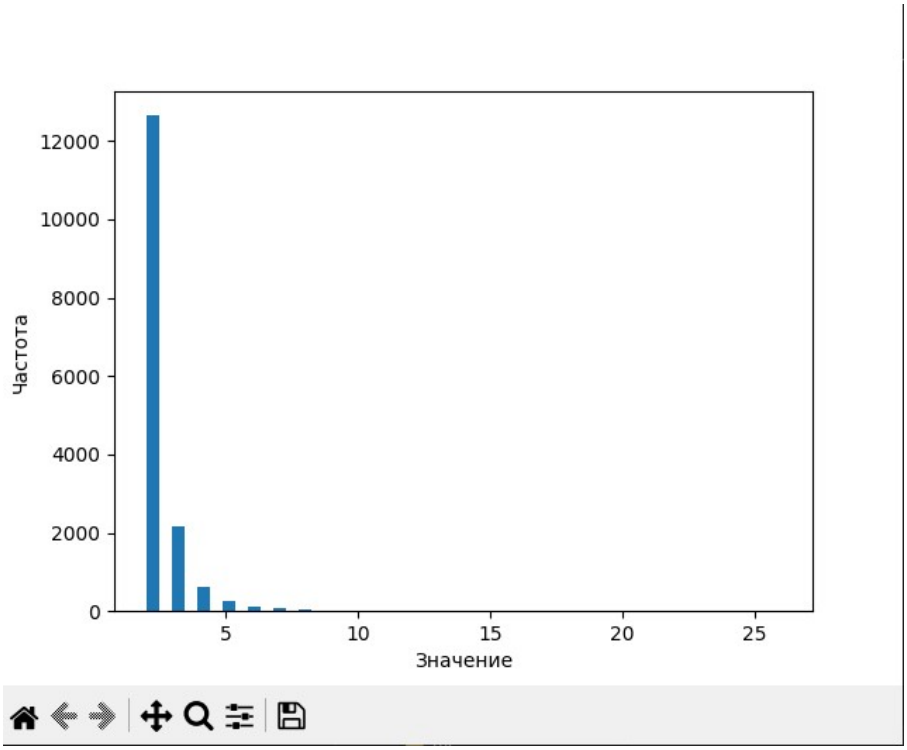

Рисунок 37 – График распределения

Исходный код программы:

```
vector <double> lnFunc(string distributionMethod, string inputFileName, string parameter1,
↪  string parameter2, bool defaultVec, bool& errorMessage) {
        ifstream inFile(inputFileName);
        string lineFile;
```

```
vector <double> allElems;
if (!inFile.is_open()) {
        cout << "Ошибка открытия файла " << inputFileName << endl;
        errorMessage = true;
        return allElems;
}
string getlineElem;
getline(inFile, getlineElem);
stringstream ss(getlineElem);
while (getline(ss, getlineElem, ',')) {
        double intElem = stod(getlineElem);
        allElems.push_back(intElem);
}
inFile.close();

if (allElems.size() % 2 != 0)
allElems.erase(allElems.end() - 1);

vector <double> allU = XtoU(allElems);
double a = stod(parameter1);
double b = stod(parameter2);
for (int i = 0; i < allU.size(); i += 2)
{
        if (allU[i] == 1) {
                cout << "Одно из случайных чисел U = 1. Ошибка в вычислении ln(1 -
                ↪  U)" << endl;
                errorMessage = true;
                return allU;
        }
        double U1 = allU[i];
        double U2 = allU[i + 1];
        double Z1 = a + (b * sqrt(-2 * log(1 - U1)) * cos(2 * M_PI * U2));
        double Z2 = a + (b * sqrt(-2 * log(1 - U1)) * sin(2 * M_PI * U2));
        double bminZ1 = b - Z1;
        if (bminZ1 > 43) {
                cout << "Разность b - Z1 больше 43. Выход за границы double в
                ↪  вычислении exp(b - Z1)" << endl;
                errorMessage = true;
                return allU;
        }
        double bminZ2 = b - Z2;
        if (bminZ1 > 43) {
                cout << "Разность b - Z2 больше 43. Выход за границы double в
                ↪  вычислении exp(b - Z2)" << endl;
                errorMessage = true;
                return allU;
        }
        allU[i] = a + exp(bminZ1);
```

```
            allU[i + 1] = a + exp(bminZ2);
        }

        return allU;
}
```

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:

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

Параметры запуска программы:

rnc.exe /d:ls /f:D:prngTestMt.txt /p1:7 /p2:16

```
D:\cpp_projects\rnc\rnc>rnc.exe /d:ls /f:D:prngTestMt.txt /p1:7 /p2:16

Distribution method: ls
Parameter 1: 7
Parameter 2: 16
Input File: D:prngTestMt.txt
Output File: lsOutput.dat
```

Рисунок 38 – Запуск программы

Рисунок 39 – Последовательность с заданным распределением

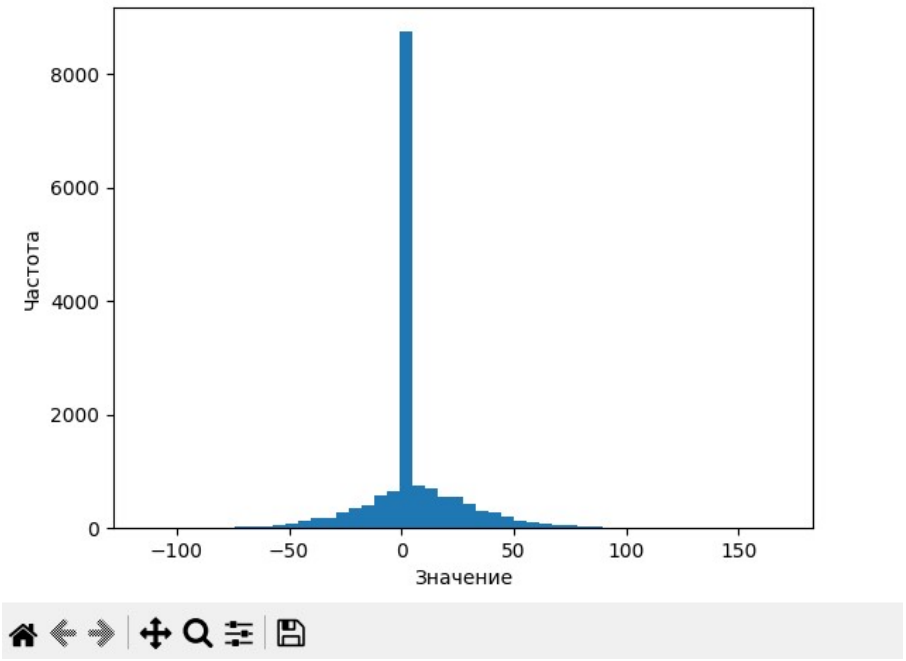

Рисунок 40 – График распределения

Исходный код программы:

```
vector <double> lsFunc(string distributionMethod, string inputFileName, string parameter1,
↪  string parameter2, bool defaultVec, bool& errorMessage) {
        ifstream inFile(inputFileName);
        string lineFile;
        vector <double> allElems;
        if (!inFile.is_open()) {
                cout << "Ошибка открытия файла " << inputFileName << endl;
                errorMessage = true;
                return allElems;
        }
        string getlineElem;
        getline(inFile, getlineElem);
        stringstream ss(getlineElem);
        while (getline(ss, getlineElem, ',')) {
                double intElem = stod(getlineElem);
                allElems.push_back(intElem);
        }
        inFile.close();

        vector <double> allU = XtoU(allElems);
        double a = stod(parameter1);
        double b = stod(parameter2);
        for (int i = 0; i < allU.size(); i += 2)
        {
                if (allU[i] == 1) {
```

```
                cout << "Одно из случайных чисел U = 1. Ошибка в вычислении U/(1 -
                ↪  U). Деление на 0." << endl;
                errorMessage = true;
                return allU;
            }
            else if (allU[i] == 0) {
                cout << "Одно из случайных чисел U = 0. Ошибка в вычислении ln(U/(1
                ↪  - U))" << endl;
                errorMessage = true;
                return allU;
            }
            allU[i] = a + (b * log(allU[i] / (1 - allU[i])));
        }

        return allU;
}
```

2.8 Биномиальное распределение

Описание алгоритма:

Вычисляют функцию распределения

$$F(y) = \sum_{k=0}^{y} C_n^k p^k (1-p)^{n-k},\ y = 0, 1, \ldots, n$$

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

Параметры запуска программы:

rnc.exe /d:bi /f:D:prngTestMt.txt /p1:10 /p2:0,5

```
D:\cpp_projects\rnc\rnc>rnc.exe /d:bi /f:D:prngTestMt.txt /p1:10 /p2:0,5

Distribution method: bi
Parameter 1: 10
Parameter 2: 0,5
Input File: D:prngTestMt.txt
Output File: biOutput.dat
```

Рисунок 41 – Запуск программы

Рисунок 42 – Последовательность с заданным распределением

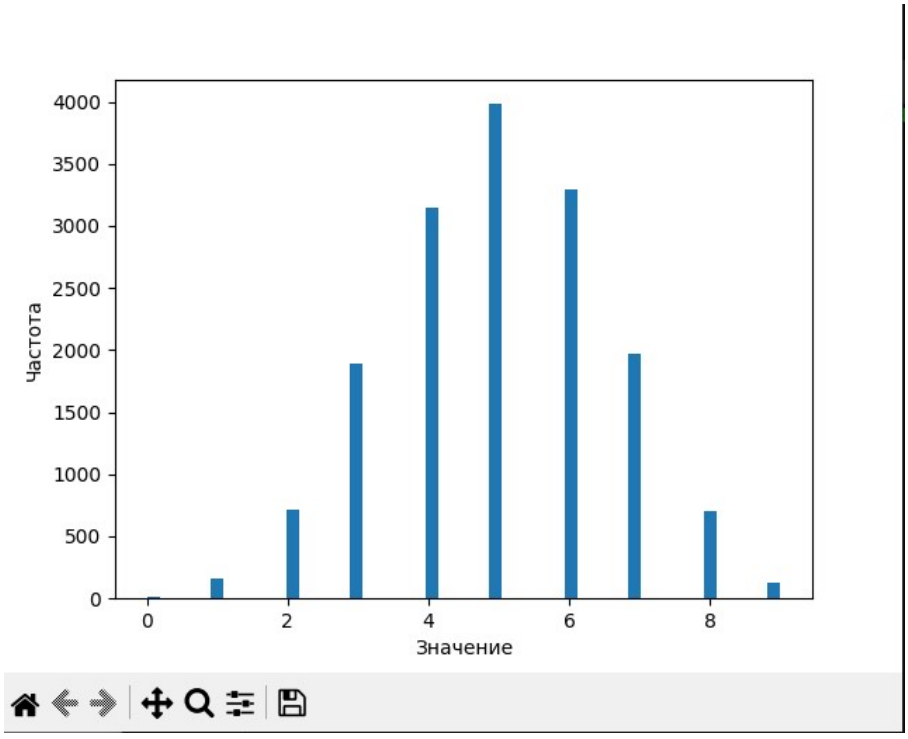

Рисунок 43 – График распределения

Исходный код программы:

```
double Cnk(double n, double k) {
        double nfact = 1;
        for (int i = 1; i <= n; i++) {
                nfact = nfact * i;
```

```
        }
        double nminkfact = 1;
        for (int i = 1; i <= n - k; i++) {
                nminkfact = nminkfact * i;
        }
        double kfact = 1;
        for (int i = 1; i <= k; i++) {
                kfact = kfact * i;
        }
        double res = nfact / (nminkfact * kfact);
        return res;
}

double funcBiF(double y, double n, double p) {
        double resSum = 0;
        for (int k = 0; k <= y; k++)
        {
                resSum = resSum + (Cnk(n, k) * pow(p, k) * pow(1 - p, n - k));
        }
        return resSum;
}

vector <double> biFunc(string distributionMethod, string inputFileName, string parameter1,
↪  string parameter2, bool defaultVec, bool& errorMessage) {
        ifstream inFile(inputFileName);
        string lineFile;
        vector <double> allElems;
        if (!inFile.is_open()) {
                cout << "Ошибка открытия файла " << inputFileName << endl;
                errorMessage = true;
                return allElems;
        }
        string getlineElem;
        getline(inFile, getlineElem);
        stringstream ss(getlineElem);
        while (getline(ss, getlineElem, ',')) {
                double intElem = stod(getlineElem);
                allElems.push_back(intElem);
        }
        inFile.close();

        vector <double> allU = XtoU(allElems);
        double n = stod(parameter1); //n
        double p = stod(parameter2); //p
        if (p > 1 || p < 0) {
                cout << "Ошибка. Параметр p2 не может быть > 1 или < 0, т.к. это
                ↪  вероятность." << endl;
                errorMessage = true;
```

```
                return allElems;
        }
        for (int i = 0; i < allU.size(); i++)
        {
                double y = 0;
                for (int j = 0; j <= n; j++)
                {
                        if (allU[i] > funcBiF(y, n, p)) {
                                y++;
                                if (j == n) {
                                        cout << "Ошибка. Не нашлось такого y = (0,...,n),
                                        ↪  при котором U <= F(y)" << endl;
                                        errorMessage = true;
                                        return allElems;
                                }
                        }
                        else
                        break;
                }
                allU[i] = y;
        }

        return allU;
}
```

ПРИЛОЖЕНИЕ А

Исходный код задания 1

```
#include <iostream>
#include <string>
#include <vector>
#include <fstream>
#include <iomanip>
#include <bitset>
#include <sstream>
#include <deque>
#include <cmath>

using namespace std;

deque<bool> binary(int x0, int num_bits) {
        bitset<sizeof(int) * 8> bits(x0);
        deque<bool> binary_digits;
        for (int i = num_bits - 1; i >= 0; --i) {
                binary_digits.push_back(bits[i]);
        }
        return binary_digits;
}

long long binToDec(string binStr)
{
        short i;
        long long res = 0;
        for (i = 0; i < sizeof(long long) * 8 && binStr[i]; ++i)
        {
                res <<= 1;
                res |= (binStr[i] - '0');
        }
        return res;
}

long long binDequeToDec(deque<bool> binStr)
{
        short i;
        long long res = 0;
        for (i = 0; i < sizeof(long long) * 8 && binStr[i]; ++i)
        {
                res <<= 1;
                res |= (binStr[i] - '0');
        }
        return res;
}
```

```
vector<int> splitStr(string str, char symbol)
{
        vector<int> elems;
        int elem;
        stringstream ss(str);
        while (ss >> elem) {
                elems.push_back(elem);
                if (ss.peek() == symbol) {
                        ss.ignore();
                }
        }
        return elems;
}

pair<vector<int>, vector <deque <bool>>> splitStrFsr(string str, char symbol, string
↪  codeMethod)
{
        vector<int> elems;
        vector <deque <bool>> nfsrRegs;
        if (codeMethod == "lfsr") {
                char elem;
                int k = 0;
                vector<string> elemsLfsr(2);
                stringstream ss(str);
                while (ss >> elem) {
                        elemsLfsr[k].push_back(elem);
                        if (ss.peek() == symbol) {
                                ss.ignore();
                                k++;
                        }
                }
                elems.push_back(stoi(elemsLfsr[1]));
                int elemsLfsrSize = elemsLfsr[0].size();
                for (int i = 0; i < elemsLfsrSize; i++)
                {
                        if (elemsLfsr[0][i] == '1')
                        {
                                elems.push_back(elemsLfsrSize - 1 - i);
                        }
                }
                //x0,coeffs
        }
        else {
                char elem;
                int k = 0;
                vector<string> elemsNfsr(3);
                stringstream ss(str);
                while (ss >> elem) {
```

```
                        elemsNfsr[k].push_back(elem);
                        if (ss.peek() == symbol) {
                                ss.ignore();
                                k++;
                        }
                }
                for (int j = 0; j < elemsNfsr.size(); j++)
                {
                        int elemsNfsrSize = elemsNfsr[j].size();
                        int coeffsCount = 0;
                        vector <int> elemsDop;
                        for (int i = 0; i < elemsNfsrSize; i++)
                        {
                                if (elemsNfsr[j][i] == '1')
                                {
                                        elemsDop.push_back(elemsNfsrSize - 1 - i);
                                        coeffsCount++;
                                }
                        }
                        elems.push_back(coeffsCount);
                        for (int i = 0; i < coeffsCount; i++)
                        {
                                elems.push_back(elemsDop[i]);
                        }
                }
                //size1,coeffs1,size2,coeffs2,size3,coeffs3
                for (int i = 0; i < 3; i++)
                {
                        deque<bool> strInDeq;
                        for (int j = 0; j < elemsNfsr[i].size(); j++)
                        {
                                if (elemsNfsr[i][j] == '1')
                                strInDeq.push_back(1);
                                else
                                strInDeq.push_back(0);
                        }
                        nfsrRegs.push_back(strInDeq);
                }
        }
        pair<vector<int>, vector <deque <bool>>> resPair = { elems , nfsrRegs };
        return resPair;
}

deque<bool> multRs(deque<bool> r1, deque<bool> r2) {
        int r1size = r1.size();
        int r2size = r2.size();
        if (r1size > r2size) {
                while (r2size != 0) {
```

```
                        r1[r1size] &= r2[r2size];
                        r2size--;
                        r1size--;
                }
                return r1;
        }
        else {
                while (r1size != 0) {
                        r1[r2size] &= r2[r1size];
                        r2size--;
                        r1size--;
                }
                return r2;
        }
        return r1;
}

deque<bool> orRs(deque<bool> r1, deque<bool> r2) {
        int r1size = r1.size();
        int r2size = r2.size();
        if (r1size > r2size) {
                while (r2size != 0) {
                        r1[r1size] |= r2[r2size];
                        r2size--;
                        r1size--;
                }
                return r1;
        }
        else {
                while (r1size != 0) {
                        r1[r2size] |= r2[r1size];
                        r2size--;
                        r1size--;
                }
                return r2;
        }
        return r1;
}

deque<bool> xorRs(deque<bool> r1, deque<bool> r2) {
        int r1size = r1.size();
        int r2size = r2.size();
        if (r1size > r2size) {
                while (r2size != 0) {
                        r1[r1size] ^= r2[r2size];
                        r2size--;
                        r1size--;
                }
```

```
                return r1;
        }
        else {
                while (r1size != 0) {
                        r1[r2size] ^= r2[r1size];
                        r2size--;
                        r1size--;
                }
                return r2;
        }
        return r1;
}

string findInStr(string const& str, int n) {
        if (str.length() < n) {
                return str;
        }
        return str.substr(0, n);
}

void dequePrint(deque<bool> reg) {
        cout << "deque: {";
                int regSize = reg.size();
                for (int i = 0; i < regSize; i++)
                {
                        if (i != regSize - 1)
                        cout << reg[i] << ",";
                        else
                        cout << reg[i] << "}";
        }
        cout << "; size: " << regSize << "." << endl;
        return;
}

int powmod(int x, int e, int n) {
        for (int i = 0; i < e - 1; i++)
        {
                x = (x * x) % n;
        }
        return x;
}

long long nod(long long x, long long y) {
        while (x != y) {
                if (x > y) {
                        x = x - y;
                }
                else {
```

```
                        y = y - x;
                }
        }
        return x;
}

bool mal_fermaPrime(long long p) {
        vector <long long> test;
        long long prime = 0;
        long long not_prime = 0;
        for (long long i = 1; i < 100; i++)
        {
                long long pi = p - i;
                if (pi >= 2 && nod(pi, p) == 1)
                {
                        test.push_back(pi);
                }
        }
        for (long long i = 0; i < test.size(); i++)
        {
                long long a_step = test[i];
                long long a_step_dop = test[i];
                for (int j = 1; j < p - 1; j++)
                {
                        a_step *= a_step_dop;
                        a_step = a_step % p;
                }
                if (a_step == 1) {
                        prime++;
                }
                else {
                        not_prime++;
                }
        }
        if (prime <= not_prime)
        return false;
        else
        return true;
}

bool kratP(long long a, long long b) {
        for (int i = 2; i < b; i++)
        {
                if (b % i == 0 && a % i == 0 && mal_fermaPrime(i))
                return true;
        }
        return false;
}
```

```
vector<int> progressVector(int NumCount) {
        vector<int> shows;
        int NumCount10 = NumCount / 10;
        for (int i = 1; i < 10; i++)
        {
                shows.push_back(NumCount10 * i);
        }
        shows.push_back(NumCount - 1);
        cout << "\nThe process of generating pseudo-random numbers: \n\n";
        return shows;
}

deque <bool> gen_p_lc(int p) {
        deque <bool> ps;
        //Модуль, множитель, приращение, начальное значение
        long long m = 7875;
        long long a = 421;
        long long c = 1663;
        long long x0 = 32112;

        for (int i = 0; i < p; i++)
        {
                x0 = (x0 * a + c) % m;
                ps.push_back(x0 % 2);
        }
        return ps;
}

void helpFunc() {
        cout << "The command with /h is introduced. Permissible parameters:";
        cout << "\n\n/g:<method_code> - the parameter specifies the method of DRB
        ↪  generation, where the method_code can be one of the following:\n";
        cout << "\n  lc - linear congruent method (Input: module, multiplier,
        ↪  increment, initial value);\n add - additive method (Input: module, low
        ↪  index, high index, sequence of initial values);\n 5p - five-parameter
        ↪  method (Input: p, q1, q2, q3, w);\n lfsr - Linear-feedback shift
        ↪  register (LFSR) (Input: binary representation of the coefficient vector,
        ↪  initial value of the register);\n nfsr - non-linear combination of LFSR
        ↪  (Input: binary representation of the coefficient vectors for R1, R2,
        ↪  R3);\n mt - Mersenne vortex (Input: module, initial value of x);\n rc4 -
        ↪  RC4 (Input: 256 initial values);\n rsa - RSA-based PRNG (Input: module
        ↪  n, number e, initial value of x; e satisfies conditions: 1 < e <
        ↪  (p-1)(q-1), NOD(e, (p-1)(q-1)) = 1, where p*q=n. x from the interval
        ↪  [1,n]);\n bbs - Blum-Blum-Schube algorithm (Input: Initial value of x
        ↪  (mutually prime with n));\n";
        cout << "\n\n/i:<number> - initialization vector of the generator.";
```

```
        cout << "\n\n/n:<length> - the number of generated numbers. If the parameter
        ↪  is not specified - 10000 numbers are generated.";
        cout << "\n\n/f:<full_file_name> - full name of the file in which the data
        ↪  will be output. If the parameter is not specified, the data should be
        ↪  written to the file named rnd.dat.";
        cout << "\n\n/h - information about the allowed command line parameters of
        ↪  the program.\n";

}

// Linear congruent method
void lcFunc(string codeMethod, vector <int> genVec, int NumCount, const char*
↪  fileName, bool defaultVec) {
        long long m = 12960;
        long long a = 1741;
        long long c = 2731;
        long long x0 = 1;
        if (!defaultVec) {
                m = genVec[0];
                a = genVec[1];
                c = genVec[2];
                x0 = genVec[3];
        }
        if (m <= 0 || a > m || a < 0 || c > m || c < 0 || x0 > m || x0 < 0) {
                cout << "\nError. The parameters do not meet the requirements.\n";
                return;
        }
        if (nod(c, m) != 1) {
                cout << "\nThe parameters do not meet the requirements of Theorem
                ↪  3.1 (Item 1).\n";
        }
        if (!(kratP(a - 1, m))) {
                cout << "\nThe parameters do not meet the requirements of Theorem
                ↪  3.1 (Item 2).\n";
        }
        if (m % 4 == 0 && (a - 1) % 4 != 0) {
        cout << "\nThe parameters do not meet the requirements of Theorem 3.1 (Item
        ↪  3).\n";
}
vector<int> showsVector = progressVector(NumCount);
int check_progress_index = 0;

ofstream f;
f.open(fileName, ios::out);
for (int i = 0; i < NumCount; i++)
{
        x0 = (x0 * a + c) % m;
        if (defaultVec)
```

```
                f << x0 % 1001 << ",";
                else
                f << x0 << ",";

                if (i == showsVector[check_progress_index]) {
                        cout << (check_progress_index + 1) * 10 << "% completed\n";
                        check_progress_index++;
                }
        }
        f.close();
        return;
}

// Additive method
void addFunc(string codeMethod, vector <int> genVec, int NumCount, const char* fileName,
↪  bool defaultVec) {
// Input: module, low index, high index, sequence of initial values
long long m = 8001;
long long k = 18;
long long j = 65;
long long x0;
vector <long long> xNs;
if (!defaultVec) {
        m = genVec[0];
        k = genVec[1];
        j = genVec[2];
        for (int i = 3; i < genVec.size(); i++)
        {
                xNs.push_back(genVec[i]);
        }
}
else {
        xNs = { 816, 159, 798, 290, 168, 441, 691, 655, 874, 220, 125, 977, 586, 381, 868,
        ↪  294, 948, 437, 581, 181, 701, 536, 11, 672, 103, 601, 794, 189, 12, 130, 386,
        ↪  828, 288, 183, 117, 456, 624, 807, 110, 498, 27, 234, 474, 613, 615, 341, 906,
        ↪  562, 778, 486, 155, 276, 894, 441, 226, 762, 234, 762, 98, 458, 399, 445, 765,
        ↪  223, 879 }
        ;
}

if (k < 1 || k >= j || j < 1) {
        cout << "\nError. The parameters do not meet the requirements.\n";
        return;
}

vector<int> showsVector = progressVector(NumCount);
int check_progress_index = 0;
```

```
ofstream f;
f.open(fileName, ios::out);
long long maxkj = max(k, j);
if (xNs.size() < maxkj) {
        cout << "\nError. The parameters do not meet the requirements (The transferred
        ↪  initial values are insufficient).\n";
        return;
}
for (int i = maxkj + 1; i < NumCount + maxkj + 1; i++)
{
        x0 = (xNs[i - k] + xNs[i - j]) % m;
        xNs.push_back(x0);
        if (defaultVec)
        f << x0 % 1001 << ",";
        else
        f << x0 << ",";

        if (i == showsVector[check_progress_index]) {
                cout << (check_progress_index + 1) * 10 << "% completed\n";
                check_progress_index++;
        }
}
f.close();
return;
}

// Five-parameter method
void fivePFunc(string codeMethod, vector <int> genVec, int NumCount, const char* fileName,
↪  bool defaultVec) {
// Input: p, q1, q2, q3, w
int p = 4253;
long long q1 = 1093;
long long q2 = 2254;
long long q3 = 3297;
int w = 16;
if (!defaultVec) {
        p = genVec[0];
        q1 = genVec[1];
        q2 = genVec[2];
        q3 = genVec[3];
        w = genVec[4];
}

if (q1 >= p || q2 >= p || q3 >= p) {
        cout << "\nError. The parameters do not meet the requirements.\n";
        return;
}
```

```
int regLength = p;
deque <bool> reg = gen_p_lc(p);
vector<int> showsVector = progressVector(NumCount);
int check_progress_index = 0;

ofstream f;
f.open(fileName, ios::out);
for (int i = 0; i < NumCount; i++)
{
        bool xNplusP = reg[regLength - q1 - 1] ^ reg[regLength - q2 - 1] ^ reg[regLength -
        ↪  q3 - 1];
        reg.push_front(xNplusP);
        reg.pop_back();
        long long ReginDec = 0;
        for (int h = 0; h < w; h++)
        {
                ReginDec *= 2;
                ReginDec += reg[h];
        }
        if (defaultVec)
        f << ReginDec % 1001 << ",";
        else
        f << ReginDec << ",";

        if (i == showsVector[check_progress_index]) {
                cout << (check_progress_index + 1) * 10 << "% completed\n";
                check_progress_index++;
        }
}
f.close();
return;
}

// Linear-feedback shift register (LFSR)
void lfsrFunc(string codeMethod, vector <int> genVec, int NumCount, const char* fileName,
↪  bool defaultVec) {
// Input: binary representation of the coefficient vector, initial value of the register
if (defaultVec) {
        genVec = { 121, 14, 5, 3, 1, 0 }; // x0, coeffs (100000000101011)
}
int x0 = genVec[0];
int x0Length = log2(x0) + 1;
int regLength = x0Length - 1;
bool bigCoeff = false;
for (int i = 1; i < genVec.size(); i++)
{
        if (genVec[i] > regLength) {
                regLength = genVec[i] + 1;
```

```
                bigCoeff = true;
        }
}
if (x0 < 0) {
        cout << "\nError. The parameters do not meet the requirements.\n";
        return;
}

vector<int> showsVector = progressVector(NumCount);
int check_progress_index = 0;

deque<bool> reg;
if (!bigCoeff) {
        regLength++;
        reg = binary(x0, regLength);
}
else {
        reg.assign(regLength - x0Length, 0);
        deque<bool> regDop = binary(x0, x0Length);
        for (int i = 0; i < regDop.size(); i++)
        {
                reg.push_back(regDop[i]);
        }
}
ofstream f;
f.open(fileName, ios::out);
for (int i = 0; i < NumCount; i++)
{
        bool xNplusP = reg[regLength - genVec[1] - 1];
        for (int j = 2; j < genVec.size(); j++)
        {
                xNplusP ^= reg[regLength - genVec[j] - 1];
        }
        reg.push_front(xNplusP);
        reg.pop_back();
        long long ReginDec = 0;
        for (int h = 0; h < reg.size(); h++)
        {
                ReginDec *= 2;
                ReginDec += reg[h];
        }
        if (defaultVec)
        f << ReginDec % 1001 << ",";
        else
        f << ReginDec << ",";

        if (i == showsVector[check_progress_index]) {
                cout << (check_progress_index + 1) * 10 << "% completed\n";
```

```
                check_progress_index++;
        }
}
f.close();
return;
}

// Non-linear combination of LFSR
void nfsrFunc(string codeMethod, vector <int> genVec, int NumCount, const char* fileName,
↪  vector <deque <bool>> nfsrRegs, bool defaultVec) {
// Input: binary representation of the coefficient vectors for R1, R2, R3
if (defaultVec) {
        genVec = { 5, 14, 4, 3, 1, 0, 3, 15, 1, 0, 5, 16, 5, 3, 2, 0 }; //
        ↪  size1,coeffs1,size2,coeffs2,size3,coeffs3
        ↪  (100000000011011,1000000000000011,10000000000101101)
        nfsrRegs = { {1,0,0,0,0,0,0,0,0,0,1,1,0,1,1}, {1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1},
        ↪  {1,0,0,0,0,0,0,0,0,0,0,1,0,1,1,0,1} };
}
int nachR1 = 1;
int endR1 = genVec[0] + 1;
int nachR2 = endR1 + 1;
int endR2 = genVec[endR1] + nachR2 + 1;
int nachR3 = endR2 + 1;
int endR3 = genVec.size();
vector<int> startsRslos = { nachR1, nachR2, nachR3 };
vector<int> endssRslos = { endR1, endR2, endR3 };
vector<int> regLengths;
for (int i = 0; i < 3; i++)
{
        regLengths.push_back(nfsrRegs[i].size());
}
vector <deque<bool>> regs = nfsrRegs;

vector<int> showsVector = progressVector(NumCount);
int check_progress_index = 0;

ofstream f;
f.open(fileName, ios::out);
for (int i = 0; i < NumCount; i++)
{
        for (int j = 0; j < 3; j++)
        {
                int startH = startsRslos[j];
                int endH = endssRslos[j];
                bool xNext = regs[j][regLengths[j] - genVec[startH] - 1];
                for (int h = startH; h < endH; h++)
                {
                        xNext ^= regs[j][regLengths[j] - genVec[h] - 1];
```

```
                }
                regs[j].push_front(xNext);
                regs[j].pop_back();
        }
        vector <deque<bool>> regs2(2);
        regs2[0] = multRs(regs[0], regs[1]);
        regs2[1] = multRs(regs[1], regs[2]);
        deque<bool> resR = xorRs(xorRs(regs2[0], regs2[1]), regs[2]);

        long long ReginDec = 0;
        for (int h = 0; h < resR.size(); h++)
        {
                ReginDec *= 2;
                ReginDec += resR[h];
        }
        if (defaultVec)
        f << ReginDec % 1001 << ",";
        else
        f << ReginDec << ",";

        if (i == showsVector[check_progress_index]) {
                cout << (check_progress_index + 1) * 10 << "% completed\n";
                check_progress_index++;
        }
}
f.close();
return;
}

// Mersenne vortex
void mtFunc(string codeMethod, vector <int> genVec, int NumCount, const char* fileName, bool
↪  defaultVec) {
// Input: module, initial value of x
int p = 624;
int w = 32;
//string uBin = "10000000000000000000000000000000";
//string hBin = "01111111111111111111111111111111";
long long uDec = 2147483647;
long long hDec = 2147483648;
long long r = 31;
long long q = 397;
long long a = 2567483615;
long long u = 11;
long long s = 7;
long long t = 15;
long long l = 18;
long long b = 2636928640;
long long c = 4022730752;
```

```
if (defaultVec) {
        genVec = { 10001, 8191 };
}
long long zMod = genVec[0];
long long x0 = genVec[1];
vector <long long> x_zns(1, x0);
for (int i = 0; i < p; i++)
{
        x_zns.push_back(abs(1812433253 * (x_zns[i - 1] ^ (x_zns[i - 1] >> 30)) + i));
}

vector<int> showsVector = progressVector(NumCount);
int check_progress_index = 0;

ofstream f;
f.open(fileName, ios::out);
int n = 0;
for (int countN = 0; countN < NumCount; countN++)
{
        long long xn = x_zns[n];
        long long xnPlus1 = x_zns[(n + 1) % p];
        long long yNew = (xn & uDec) | (xnPlus1 & hDec);
        int yNewLength = log2(yNew) + 1;
        deque<bool> yBin = binary(yNew, yNewLength);
        long long Xnew;
        yNew >>= 1;
        long long xnqp = x_zns[(n + q) % p];
        if (yBin[yBin.size() - 1] == 1)
        Xnew = xnqp ^ yNew ^ a;
        else
        Xnew = xnqp ^ yNew ^ 0;
        yNew = Xnew;
        yNew = yNew ^ (yNew >> u);
        yNew = yNew ^ ((yNew << s) & b);
        yNew = yNew ^ ((yNew << t) & c);
        long long zNew = yNew ^ (yNew >> l);
        if (defaultVec)
        f << (zNew % zMod) % 1001 << ",";
        else
        f << zNew % zMod << ",";

        if (countN == showsVector[check_progress_index]) {
                cout << (check_progress_index + 1) * 10 << "% completed\n";
                check_progress_index++;
        }
        x_zns[n] = Xnew;
        n = (n + 1) % p;
}
```

```
f.close();
return;
}

// RC4
void rc4Func(string codeMethod, vector <int> genVec, int NumCount, const char* fileName,
↪  bool defaultVec) {
// Input: 256 initial values
if (defaultVec) {
        genVec = { 802, 720, 341, 337, 961, 882, 417, 785, 198, 727, 899, 372, 374, 425,
        ↪  556, 615, 813, 768, 840, 183, 893, 568, 73, 387, 18, 436, 182, 125, 806, 899,
        ↪  485, 607, 619, 825, 944, 579, 707, 360, 363, 904, 87, 262, 276, 460, 687, 831,
        ↪  75, 499, 599, 915, 681, 492, 483, 754, 878, 500, 189, 60, 624, 994, 959, 109,
        ↪  600, 577, 934, 544, 156, 640, 903, 519, 544, 990, 781, 819, 449, 468, 650, 524,
        ↪  967, 248, 438, 647, 739, 920, 400, 617, 419, 588, 676, 43, 581, 634, 151, 181,
        ↪  211, 84, 724, 367, 723, 627, 886, 267, 617, 667, 85, 65, 134, 735, 589, 100,
        ↪  983, 26, 747, 721, 945, 147, 337, 364, 734, 13, 406, 315, 647, 556, 496, 858,
        ↪  640, 220, 224, 362, 847, 110, 629, 463, 776, 713, 528, 909, 448, 116, 9, 430,
        ↪  141, 755, 151, 86, 901, 488, 449, 635, 500, 855, 950, 147, 410, 446, 4, 49, 665,
        ↪  227, 411, 511, 336, 39, 974, 112, 752, 501, 21, 200, 617, 29, 629, 757, 784,
        ↪  779, 843, 684, 266, 292, 319, 766, 146, 269, 912, 556, 714, 916, 605, 378, 142,
        ↪  15, 889, 478, 54, 862, 590, 806, 363, 610, 5, 979, 638, 634, 736, 421, 413, 578,
        ↪  105, 679, 869, 424, 444, 14, 692, 356, 569, 405, 271, 173, 783, 413, 188, 671,
        ↪  891, 242, 533, 480, 48, 895, 89, 53, 873, 727, 686, 608, 147, 98, 185, 252, 776,
        ↪  54, 675, 220, 67, 366, 576, 636, 771, 846, 808, 553, 259, 996, 224, 149 };
}
vector <int> k = genVec;
vector <int> s = genVec;
int j = 0;
for (int i = 0; i < 256; i++)
{
        j = (j + s[i] + k[i]) % 256;
        swap(s[i], s[j]);
}
int i = 0;
j = 0;

vector<int> showsVector = progressVector(NumCount);
int check_progress_index = 0;

ofstream f;
f.open(fileName, ios::out);
for (int countN = 0; countN < NumCount; countN++)
{
        i = (i + 1) % 256;
        j = (j + s[j]) % 256;
        swap(s[i], s[j]);
        int t = (s[i] + s[j]) % 256;
```

```
        if (defaultVec)
        f << s[t] % 1001 << ",";
        else
        f << s[t] << ",";

        if (countN == showsVector[check_progress_index]) {
                cout << (check_progress_index + 1) * 10 << "% completed\n";
                check_progress_index++;
        }
}
f.close();
return;
}

// RSA-based PRNG
void rsaFunc(string codeMethod, vector <int> genVec, int NumCount, const char* fileName,
↪  bool defaultVec) {
// Input: module n, number e, initial value of x; e satisfies conditions: 1 < e <
↪  (p-1)(q-1), NOD(e, (p-1)(q-1)) = 1, where p*q=n. x from the interval [1,n]
if (defaultVec) {
        genVec = { 7191817, 151, 69 };
}
int n = genVec[0];
int e = genVec[1];
int x = genVec[2];
int l = 20;

if (e <= 1 || x < 1 || x > n - 1) {
        cout << "\nError. The parameters do not meet the requirements.\n";
        return;
}

vector<int> showsVector = progressVector(NumCount);
int check_progress_index = 0;

ofstream f;
f.open(fileName, ios::out);
for (int countN = 0; countN < NumCount; countN++)
{
        string zs = "";
        for (int i = 0; i < l; i++)
        {
                x = powmod(x, e, n);
                char z = '0';
                if (x % 2 != 0)
                z = '1';
                zs.push_back(z);
        }
```

```
        if (defaultVec)
        f << binToDec(zs) % 1001 << ",";
        else
        f << binToDec(zs) << ",";

        if (countN == showsVector[check_progress_index]) {
                cout << (check_progress_index + 1) * 10 << "% completed\n";
                check_progress_index++;
        }
}
f.close();
return;
}

// Blum-Blum-Schube algorithm
void bbsFunc(string codeMethod, vector <int> genVec, int NumCount, const char* fileName,
↪  bool defaultVec) {
// Input: Initial value of x (mutually prime with n)
if (defaultVec) {
        genVec = { 8627 };
}
int x0 = genVec[0];
int n = 16637;
int l = 10;

if (nod(x0, n) != 1) {
        cout << "\nThe parameters do not meet the requirements (x and n are not mutually
        ↪  prime).\n";
}

vector<int> showsVector = progressVector(NumCount);
int check_progress_index = 0;

ofstream f;
f.open(fileName, ios::out);
for (int countN = 0; countN < NumCount; countN++)
{
        x0 = (x0 * x0) % n;
        string zs = "";
        for (int i = 0; i < l; i++)
        {
                x0 = (x0 * x0) % n;
                char z = '0';
                if (x0 % 2 != 0)
                z = '1';
                zs.push_back(z);
        }
        if (defaultVec)
```

```
        f << binToDec(zs) % 1001 << ",";
        else
        f << binToDec(zs) << ",";

        if (countN == showsVector[check_progress_index]) {
                cout << (check_progress_index + 1) * 10 << "% completed\n";
                check_progress_index++;
        }
}
f.close();
return;
}

void setMethod(string codeMethod, vector <int> genVec, int NumCount, const char* fileName,
↪  int genVecSize, vector <deque <bool>> nfsrRegs, bool defaultVec) {
if (codeMethod == "lc")
lcFunc(codeMethod, genVec, NumCount, fileName, defaultVec);
else if (codeMethod == "add")
addFunc(codeMethod, genVec, NumCount, fileName, defaultVec);
else if (codeMethod == "5p")
fivePFunc(codeMethod, genVec, NumCount, fileName, defaultVec);
else if (codeMethod == "lfsr")
lfsrFunc(codeMethod, genVec, NumCount, fileName, defaultVec);
else if (codeMethod == "nfsr")
nfsrFunc(codeMethod, genVec, NumCount, fileName, nfsrRegs, defaultVec);
else if (codeMethod == "mt")
mtFunc(codeMethod, genVec, NumCount, fileName, defaultVec);
else if (codeMethod == "rc4")
rc4Func(codeMethod, genVec, NumCount, fileName, defaultVec);
else if (codeMethod == "rsa")
rsaFunc(codeMethod, genVec, NumCount, fileName, defaultVec);
else if (codeMethod == "bbs")
bbsFunc(codeMethod, genVec, NumCount, fileName, defaultVec);
}

int main(int argc, char* argv[])
{
setlocale(LC_ALL, "Rus");
string codeMethod;
vector <int> genVec;
vector <deque <bool>> nfsrRegs;
int NumCount = 10000;
string fName = "rnd.dat";
bool defaultVec = true;
for (int i = 0; argv[i]; i++)
{
        string checkStr = string(argv[i]);
        if (findInStr(checkStr, 2) == "/h") {
```

```
                helpFunc();
                return 0;
        }
        if (checkStr.length() > 2) {
                string ifStr = findInStr(checkStr, 3);
                string subStr = checkStr.substr(3, checkStr.length());
                if (ifStr == "/g:") {
                        codeMethod = subStr;
                }
                if (ifStr == "/i:" && (codeMethod == "lfsr" || codeMethod == "nfsr")) {
                        char symbol = ',';
                        pair<vector<int>, vector <deque <bool>>> nfsrPair;
                        nfsrPair = splitStrFsr(subStr, symbol, codeMethod);
                        genVec = nfsrPair.first;
                        nfsrRegs = nfsrPair.second;
                        defaultVec = false;
                }
                else if (ifStr == "/i:") {
                        char symbol = ',';
                        genVec = splitStr(subStr, symbol);
                        defaultVec = false;
                }
                if (ifStr == "/n:") {
                        NumCount = stoi(subStr);
                }
                if (ifStr == "/f:") {
                        fName = subStr;
                }
        }
}
const char* fileName = fName.c_str();
int genVecSize = genVec.size();
setMethod(codeMethod, genVec, NumCount, fileName, genVecSize, nfsrRegs, defaultVec);
cout << "\nGeneration method: " << codeMethod << endl;
cout << "Count of generated numbers: " << NumCount << endl;
cout << "Full name of the output file: " << fileName << endl;
return 0;
}
```

ПРИЛОЖЕНИЕ Б
Исходный код задания 2

```
#define _USE_MATH_DEFINES

#include <iostream>
#include <string>
#include <vector>
#include <fstream>
#include <iomanip>
#include <bitset>
#include <sstream>
#include <deque>
#include <cmath>

using namespace std;

string findInStr(string const& str, int n) {
        if (str.length() < n) {
                return str;
        }
        return str.substr(0, n);
}

void helpFunc() {
        cout << "Введена команда с /h. Допустимые параметры:";
        cout << "\n\n/d:<распределение> - код распределения для преобразования
        ↪  последовательности. Возможные коды распределений:\n";
        cout << "\n  st - стандартное равномерное с заданным интервалом (Параметры: a, b);\n
        ↪  tr - треугольное распределение (Параметры: a, b);\n  ex - общее экспоненциальное
        ↪  распределение (Параметры: a, b);\n  nr - нормальное распределение (Параметры:
        ↪  mu, sigma);\n  gm - гамма распределение (Параметры: a, b);\n  ln - логнормальное
        ↪  распределение (Параметры: a, b);\n  ls - логистическое распределение (Параметры:
        ↪  a, b);\n  bi - биномиальное распределение. (Параметры: n, p)\n";
        cout << "\n\n/f:<имя_файла> - имя файла с входной последовательностью";
        cout << "\n\n/p1:<параметр1> - 1-й параметр, необходимый, для генерации ПСЧ
        ↪  заданного распределения.";
        cout << "\n\n/p2:<параметр2> - 2-й параметр, необходимый, для генерации ПСЧ
        ↪  заданного распределения.";
        cout << "\n\n/h - информация о допустимых параметрах командной строки программы.\n";
}

vector <double> XtoU(vector <double> allElems) {
        int allElemsSize = allElems.size();
        double max = 2.22507e-308;
        for (int i = 0; i < allElemsSize; i++)
        {
                if (allElems[i] > max)
```

```
                max = allElems[i];
        }
        max++;
        for (int i = 0; i < allElemsSize; i++)
        {
                allElems[i] = allElems[i] / max;
        }
        return allElems;
}

vector <double> stFunc(string distributionMethod, string inputFileName, string parameter1,
↪  string parameter2, bool defaultVec, bool& errorMessage) {
        ifstream inFile(inputFileName);
        string lineFile;
        vector <double> allElems;
        if (!inFile.is_open()) {
                cout << "Ошибка открытия файла " << inputFileName << endl;
                errorMessage = true;
                return allElems;
        }
        string getlineElem;
        getline(inFile, getlineElem);
        stringstream ss(getlineElem);
        while (getline(ss, getlineElem, ',')) {
                double intElem = stod(getlineElem);
                allElems.push_back(intElem);
        }
        inFile.close();

        vector <double> allU = XtoU(allElems);
        double a = stod(parameter1);
        double b = stod(parameter2);
        for (int i = 0; i < allU.size(); i++)
        {
                allU[i] = (b * allU[i]) + a;
        }

        return allU;
}

vector <double> trFunc(string distributionMethod, string inputFileName, string parameter1,
↪  string parameter2, bool defaultVec, bool& errorMessage) {
        ifstream inFile(inputFileName);
        string lineFile;
        vector <double> allElems;
        if (!inFile.is_open()) {
                cout << "Ошибка открытия файла " << inputFileName << endl;
                errorMessage = true;
```

```
                return allElems;
        }
        string getlineElem;
        getline(inFile, getlineElem);
        stringstream ss(getlineElem);
        while (getline(ss, getlineElem, ',')) {
                double intElem = stod(getlineElem);
                allElems.push_back(intElem);
        }
        inFile.close();

        vector <double> allU = XtoU(allElems);
        double a = stod(parameter1);
        double b = stod(parameter2);
        for (int i = 0; i < allU.size() - 1; i++)
        {
                allU[i] = a + (b * (allU[i] + allU[i + 1] - 1));
        }

        return allU;
}

vector <double> exFunc(string distributionMethod, string inputFileName, string parameter1,
↪  string parameter2, bool defaultVec, bool& errorMessage) {
        ifstream inFile(inputFileName);
        string lineFile;
        vector <double> allElems;
        if (!inFile.is_open()) {
                cout << "Ошибка открытия файла " << inputFileName << endl;
                errorMessage = true;
                return allElems;
        }
        string getlineElem;
        getline(inFile, getlineElem);
        stringstream ss(getlineElem);
        while (getline(ss, getlineElem, ',')) {
                double intElem = stod(getlineElem);
                allElems.push_back(intElem);
        }
        inFile.close();

        vector <double> allU = XtoU(allElems);
        double a = stod(parameter1);
        double b = stod(parameter2);
        for (int i = 0; i < allU.size() - 1; i++)
        {
                if (allU[i] == 0) {
```

```
                        cout << "Одно из случайных чисел U = 0. Ошибка в вычислении ln(U)"
                        ↪  << endl;
                        errorMessage = true;
                        return allU;
                }
                allU[i] = ((b * (-1)) * log(allU[i])) + a;
        }

        return allU;
}

vector <double> nrFunc(string distributionMethod, string inputFileName, string parameter1,
↪  string parameter2, bool defaultVec, bool& errorMessage) {
        ifstream inFile(inputFileName);
        string lineFile;
        vector <double> allElems;
        if (!inFile.is_open()) {
                cout << "Ошибка открытия файла " << inputFileName << endl;
                errorMessage = true;
                return allElems;
        }
        string getlineElem;
        getline(inFile, getlineElem);
        stringstream ss(getlineElem);
        while (getline(ss, getlineElem, ',')) {
                double intElem = stod(getlineElem);
                allElems.push_back(intElem);
        }
        inFile.close();

        if (allElems.size() % 2 != 0)
        allElems.erase(allElems.end() - 1);

        vector <double> allU = XtoU(allElems);
        double mu = stod(parameter1);
        double sigma = stod(parameter2);
        for (int i = 0; i < allU.size(); i += 2)
        {
                if (allU[i] == 1) {
                        cout << "Одно из случайных чисел U = 1. Ошибка в вычислении ln(1 -
                        ↪  U)" << endl;
                        errorMessage = true;
                        return allU;
                }
                double U1 = allU[i];
                double U2 = allU[i + 1];
                allU[i] = mu + (sigma * sqrt(-2 * log(1 - U1)) * cos(2 * M_PI * U2)); // Z1
```

```
                allU[i + 1] = mu + (sigma * sqrt(-2 * log(1 - U1)) * sin(2 * M_PI * U2)); //
                ↪ Z2
        }

        return allU;
}

vector <double> gmFunc(string distributionMethod, string inputFileName, string parameter1,
↪ string parameter2, bool defaultVec, bool& errorMessage) {
        ifstream inFile(inputFileName);
        string lineFile;
        vector <double> allElems;
        if (!inFile.is_open()) {
                cout << "Ошибка открытия файла " << inputFileName << endl;
                errorMessage = true;
                return allElems;
        }
        string getlineElem;
        getline(inFile, getlineElem);
        stringstream ss(getlineElem);
        while (getline(ss, getlineElem, ',')) {
                double intElem = stod(getlineElem);
                allElems.push_back(intElem);
        }
        inFile.close();

        vector <double> allU = XtoU(allElems);
        double a = stod(parameter1);
        double b = stod(parameter2);
        for (int i = 0; i < allU.size(); i++)
        {
                if (allU[i] == 1) {
                        cout << "Одно из случайных чисел U = 1. Ошибка в вычислении ln(1 -
                        ↪ U)" << endl;
                        errorMessage = true;
                        return allU;
                }
                allU[i] = a - (b * log(1 - allU[i]));
        }

        return allU;
}

vector <double> lnFunc(string distributionMethod, string inputFileName, string parameter1,
↪ string parameter2, bool defaultVec, bool& errorMessage) {
        ifstream inFile(inputFileName);
        string lineFile;
        vector <double> allElems;
```

```
if (!inFile.is_open()) {
        cout << "Ошибка открытия файла " << inputFileName << endl;
        errorMessage = true;
        return allElems;
}
string getlineElem;
getline(inFile, getlineElem);
stringstream ss(getlineElem);
while (getline(ss, getlineElem, ',')) {
        double intElem = stod(getlineElem);
        allElems.push_back(intElem);
}
inFile.close();

if (allElems.size() % 2 != 0)
allElems.erase(allElems.end() - 1);

vector <double> allU = XtoU(allElems);
double a = stod(parameter1);
double b = stod(parameter2);
for (int i = 0; i < allU.size(); i += 2)
{
        if (allU[i] == 1) {
                cout << "Одно из случайных чисел U = 1. Ошибка в вычислении ln(1 -
                ↪  U)" << endl;
                errorMessage = true;
                return allU;
        }
        double U1 = allU[i];
        double U2 = allU[i + 1];
        double Z1 = a + (b * sqrt(-2 * log(1 - U1)) * cos(2 * M_PI * U2));
        double Z2 = a + (b * sqrt(-2 * log(1 - U1)) * sin(2 * M_PI * U2));
        double bminZ1 = b - Z1;
        if (bminZ1 > 43) {
                cout << "Разность b - Z1 больше 43. Выход за границы double в
                ↪  вычислении exp(b - Z1)" << endl;
                errorMessage = true;
                return allU;
        }
        double bminZ2 = b - Z2;
        if (bminZ1 > 43) {
                cout << "Разность b - Z2 больше 43. Выход за границы double в
                ↪  вычислении exp(b - Z2)" << endl;
                errorMessage = true;
                return allU;
        }
        allU[i] = a + exp(bminZ1);
        allU[i + 1] = a + exp(bminZ2);
```

```
        }

        return allU;
}

vector <double> lsFunc(string distributionMethod, string inputFileName, string parameter1,
↪  string parameter2, bool defaultVec, bool& errorMessage) {
        ifstream inFile(inputFileName);
        string lineFile;
        vector <double> allElems;
        if (!inFile.is_open()) {
                cout << "Ошибка открытия файла " << inputFileName << endl;
                errorMessage = true;
                return allElems;
        }
        string getlineElem;
        getline(inFile, getlineElem);
        stringstream ss(getlineElem);
        while (getline(ss, getlineElem, ',')) {
                double intElem = stod(getlineElem);
                allElems.push_back(intElem);
        }
        inFile.close();

        vector <double> allU = XtoU(allElems);
        double a = stod(parameter1);
        double b = stod(parameter2);
        for (int i = 0; i < allU.size(); i += 2)
        {
                if (allU[i] == 1) {
                        cout << "Одно из случайных чисел U = 1. Ошибка в вычислении U/(1 -
                        ↪  U). Деление на 0." << endl;
                        errorMessage = true;
                        return allU;
                }
                else if (allU[i] == 0) {
                        cout << "Одно из случайных чисел U = 0. Ошибка в вычислении ln(U/(1
                        ↪  - U))" << endl;
                        errorMessage = true;
                        return allU;
                }
                allU[i] = a + (b * log(allU[i] / (1 - allU[i])));
        }

        return allU;
}

double Cnk(double n, double k) {
```

```
        double nfact = 1;
        for (int i = 1; i <= n; i++) {
                nfact = nfact * i;
        }
        double nminkfact = 1;
        for (int i = 1; i <= n - k; i++) {
                nminkfact = nminkfact * i;
        }
        double kfact = 1;
        for (int i = 1; i <= k; i++) {
                kfact = kfact * i;
        }
        double res = nfact / (nminkfact * kfact);
        return res;
}

double funcBiF(double y, double n, double p) {
        double resSum = 0;
        for (int k = 0; k <= y; k++)
        {
                resSum = resSum + (Cnk(n, k) * pow(p, k) * pow(1 - p, n - k));
        }
        return resSum;
}

vector <double> biFunc(string distributionMethod, string inputFileName, string parameter1,
↪  string parameter2, bool defaultVec, bool& errorMessage) {
        ifstream inFile(inputFileName);
        string lineFile;
        vector <double> allElems;
        if (!inFile.is_open()) {
                cout << "Ошибка открытия файла " << inputFileName << endl;
                errorMessage = true;
                return allElems;
        }
        string getlineElem;
        getline(inFile, getlineElem);
        stringstream ss(getlineElem);
        while (getline(ss, getlineElem, ',')) {
                double intElem = stod(getlineElem);
                allElems.push_back(intElem);
        }
        inFile.close();

        vector <double> allU = XtoU(allElems);
        double n = stod(parameter1); //n
        double p = stod(parameter2); //p
        if (p > 1 || p < 0) {
```

```
                cout << "Ошибка. Параметр p2 не может быть > 1 или < 0, т.к. это
                ↪  вероятность." << endl;
                errorMessage = true;
                return allElems;
        }
        for (int i = 0; i < allU.size(); i++)
        {
                double y = 0;
                for (int j = 0; j <= n; j++)
                {
                        if (allU[i] > funcBiF(y, n, p)) {
                                y++;
                                if (j == n) {
                                        cout << "Ошибка. Не нашлось такого y = (0,...,n),
                                        ↪  при котором U <= F(y)" << endl;
                                        errorMessage = true;
                                        return allElems;
                                }
                        }
                        else
                        break;
                }
                allU[i] = y;
        }

        return allU;
}

vector <double> setMethod(string distributionMethod, string inputFileName, string
↪  parameter1, string parameter2, bool defaultVec, bool& errorMessage) {
        vector <double> distributionResVec;
        if (distributionMethod == "st")
        distributionResVec = stFunc(distributionMethod, inputFileName, parameter1,
        ↪  parameter2, defaultVec, errorMessage);
        else if (distributionMethod == "tr")
        distributionResVec = trFunc(distributionMethod, inputFileName, parameter1,
        ↪  parameter2, defaultVec, errorMessage);
        else if (distributionMethod == "ex")
        distributionResVec = exFunc(distributionMethod, inputFileName, parameter1,
        ↪  parameter2, defaultVec, errorMessage);
        else if (distributionMethod == "nr")
        distributionResVec = nrFunc(distributionMethod, inputFileName, parameter1,
        ↪  parameter2, defaultVec, errorMessage);
        else if (distributionMethod == "gm")
        distributionResVec = gmFunc(distributionMethod, inputFileName, parameter1,
        ↪  parameter2, defaultVec, errorMessage);
        else if (distributionMethod == "ln")
```

```
        distributionResVec = lnFunc(distributionMethod, inputFileName, parameter1,
        ↪  parameter2, defaultVec, errorMessage);
        else if (distributionMethod == "ls")
        distributionResVec = lsFunc(distributionMethod, inputFileName, parameter1,
        ↪  parameter2, defaultVec, errorMessage);
        else if (distributionMethod == "bi")
        distributionResVec = biFunc(distributionMethod, inputFileName, parameter1,
        ↪  parameter2, defaultVec, errorMessage);

        return distributionResVec;
}

int main(int argc, char* argv[])
{
        /*
        Программа для преобразования последовательности ПСЧ
        в другую последовательность ПСЧ с заданным распределением.
        */
        setlocale(LC_ALL, "Rus");
        string distributionMethod;
        string inputFileName = "rncInput.dat";
        string parameter1;
        string parameter2;
        bool defaultVec = true;
        for (int i = 0; argv[i]; i++)
        {
                string checkStr = string(argv[i]);
                if (findInStr(checkStr, 2) == "/h") {
                        helpFunc();
                        return 0;
                }
                if (checkStr.length() > 2) {
                        string ifStr = findInStr(checkStr, 3);
                        string subStr = checkStr.substr(3, checkStr.length());
                        if (ifStr == "/d:") {
                                distributionMethod = subStr;
                        }
                        if (ifStr == "/f:") {
                                inputFileName = subStr;
                        }
                        if (ifStr == "/p1") {
                                parameter1 = checkStr.substr(4, checkStr.length());
                        }
                        if (ifStr == "/p2") {
                                parameter2 = checkStr.substr(4, checkStr.length());
                        }

                }
```

```
        }
        bool errorMessage = false;
        vector <double> rncRes = setMethod(distributionMethod, inputFileName, parameter1,
        ↪  parameter2, defaultVec, errorMessage);
        if (errorMessage)
        return 0;
        string outputFileName = distributionMethod + "Output.dat";
        ofstream outFile(outputFileName);
        for (int i = 0; i < rncRes.size(); i++)
        {
                if (i == rncRes.size() - 1)
                outFile << (long long)rncRes[i];
                else
                outFile << (long long)rncRes[i] << ",";
        }
        outFile.close();
        cout << "\nDistribution method: " << distributionMethod << endl;
        cout << "Parameter 1: " << parameter1 << endl;
        cout << "Parameter 2: " << parameter2 << endl;
        cout << "Input File: " << inputFileName << endl;
        cout << "Output File: " << outputFileName << endl;
        return 0;
}
```