МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»**

ТЕОРИЯ ПСЕВДОСЛУЧАЙНЫХ ГЕНЕРАТОРОВ

ЛАБОРАТОРНАЯ РАБОТА

студента 4 курса 431 группы
направления 10.05.01 — Компьютерная безопасность
факультета КНиИТ
Стаина Романа Игоревича

Проверил
доцент _____ И. И. Слеповичев

Саратов 2023

СОДЕРЖАНИЕ

1 Генератор псевдослучайных чисел

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

Последовательность ПСЧ, получаемая по формуле:

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

называется *линейной конгруэнтной последовательностью (ЛКП)*. Параметры:

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

При запуске программы дополнительно проверяется, что выполняются следующие условия, при выполнении которых ЛКП имеет период m:

1. числа c и m взаимно простые;
2. $a - 1$ кратно p для некоторого простого p, являющегося делителем m;
3. $a - 1$ кратно 4, если m кратно 4.

1.2 Аддитивный метод

Последовательность определяется следующим образом:

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

. Параметры:

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

1.3 Пятипараметрический метод

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

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

Параметры:
— p;
— q_1, q_2, q_3;
— w;
— начальное значение.

1.4 Регистр сдвига с обратной связью (РСЛОС)

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

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.

Параметры:
— двоичное представление вектора коэффициентов;
— начальное значение регистра.

1.5 Нелинейная комбинация РСЛОС

Последовательность получается из нелинейной комбинации трёх РСЛОС следующим образом:

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

Параметры:

— двоичное представление вектора коэффициентов для $R1, R2, R3$;

— x_1, x_2, x_3 — десятичное представление начальных состояний регистров $R1, R2, R3$;

— w, длина слова.

1.6 Вихрь Мерсенна

Следующий псевдокод представляет алгоритм генерации ПСЧ:

```
// Создание массива длины n для сохранения состояний генератора
int[0..n-1] MT
int index := n+1
const int lower_mask = (1 << r) - 1
const int upper_mask = lowest w bits of (not lower_mask)

// Initialize the generator from a seed
function seed_mt(int seed) {
    index := n
    MT[0] := seed
    for i from 1 to (n - 1) { // loop over each element
        MT[i] := lowest w bits of (f * (MT[i-1] xor (MT[i-1] >> (w-2))) + i)
    }
}

// Извлечение чисел на основе массива MT[index]
// Вызывает twist() каждые n чисел
function extract_number() {
    if index >= n {
        twist()
    }

    int y := MT[index]
    y := y xor ((y >> u) and d)
    y := y xor ((y << s) and b)
    y := y xor ((y << t) and c)
    y := y xor (y >> l)

    index := index + 1
```

```
    return lowest w bits of (y)
}

// Генерация следующих n значений
function twist() {
    for i from 0 to (n-1) {
        int x := (MT[i] and upper_mask)
                  | (MT[(i+1) mod n] and lower_mask)
        int xA := x >> 1
        if (x mod 2) != 0 { // lowest bit of x is 1
            xA := xA xor a
        }
        MT[i] := MT[(i + m) mod n] xor xA
    }
    index := 0
}
```

Константы, используемые в алгоритме:

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

Параметры:

— модуль;
— начальное значение.

1.7 RC4

Описание алгоритма:

1. Инициализация S_i.
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) $K = S_t$.

Параметры:

— 256 начальных значений S_i.

1.8 ГПСЧ на основе RSA

Описание алгоритма:

1. Инициализация чисел: $n = pq$, где p и q простые числа, числа e: $1 < e < f$, НОД(e, f) = 1, $f = (p-1)(q-1)$ и числа x_0 из интервала $[1, n-1]$.
2. `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
3. Вернуть $z_1, \ldots, z_w$.

Параметры:

— n, модуль;

— e, число;

— w, длина слова;

— x_0, начальное значение.

1.9 Алгоритм Блюм-Блюма-Шуба

Описание алгоритма:

1. Инициализация числа: $n = 127 \cdot 131 = 16637$.
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 $z_1, \ldots, z_w$.

Параметры:

— n, модуль;

2 Преобразование ПСЧ к заданному распределению

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;
6. Логнормальное распределение;
7. Логистическое распределение;
8. Биномиальное распределение.

2.1 Метод генерации случайной величины

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

2.2 Стандартное равномерное с заданным интервалом

$$Y = bU + a$$

2.3 Треугольное распределение

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

2.4 Общее экспоненциальное распределение

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

2.5 Нормальное распределение

$$Z_1 = \mu\sigma\sqrt{-2\ln(1-U_1)}\cos(2\pi U_2)$$
$$Z_2 = \mu\sigma\sqrt{-2\ln(1-U_1)}\sin(2\pi U_2)$$

2.6 Гамма распределение

Алгоритм для $c = k$ (k – целое число)

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

2.7 Логнормальное распределение

$$Y = a + \exp(b - Z)$$

2.8 Логистическое распределение

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

2.9 Биномиальное распределение

Следующий псевдокод представляет алгоритм преоб

```
y = binominal(x, a, b, m):
  u = U(x)
  s = 0
  k = 0
  loopstart:
    s = s + C(k, b) + a^k * (1 - a)^(b - k)
    if s > u:
      y = k
      Завершить
    if k < b - 1:
      k = k + 1
      move to loopstart
    y = b
```

ПРИЛОЖЕНИЕ А
Код задания 1

```
#include <iostream>
#include <fstream>
#include <vector>
#include <algorithm>
#include <argparse.hpp>
#include <boost/dynamic_bitset.hpp>

using namespace std;
using namespace boost;


class MersenneTwister32 {
public:
    explicit MersenneTwister32(uint32_t seed) {
        mt[0] = seed;
        for (int i = 1; i < p; ++i)
        {
            mt[i] = ((mt[i - 1] ^ (mt[i - 1] >> 30)) + i);
        }
        ind = p;
    }

    uint32_t generate() {
        if (ind >= p)
        {
            twist();
        }

        uint32_t y = mt[ind++];
        y ^= (y >> u);
        y ^= (y << s) & b;
        y ^= (y << t) & c;
        y ^= (y >> l);
        return y;
    }

private:
    static constexpr int p = 624;
    const int u = 11;
    const int s = 7;
    const int t = 15;
    const int l = 18;
    const int q = 397;
    const uint32_t a = 0x9908b0dfUL;
    const uint32_t b = 0x9d2c5680UL;
    const uint32_t c = 0xefc60000UL;
```

```
    const uint32_t upper_mask = 0x80000000UL;
    const uint32_t lower_mask = 0x7fffffffUL;
    uint32_t mt[p];
    int ind;

    void twist()
    {
        for (int i = 0; i < p; ++i)
        {
            uint32_t x = (mt[i] & upper_mask) + (mt[(i + 1) % p] & lower_mask);
            uint32_t xA = x >> 1;
            if (x & 1)
            {
                xA ^= a;
            }
            mt[i] = mt[(i + q) % p] ^ xA;
        }
        ind = 0;
    }
};

bool check_lc_conditions(int m, int a, int c)
{
    if (gcd(c, m) != 1)
    {
        return true;
    }

    int a1 = a - 1;

    if (m % 4 == 0)
    {
        if (a1 % 4 != 0)
        {
            return true;
        }
    }

    vector <int> prime_factors;
    int d = 2;
    while (m > 1) {
        while (m % d == 0) {
            prime_factors.push_back(d);
            m /= d;
        }
        d++;
        if (d * d > m) {
            if (m > 1) {
```

```
                prime_factors.push_back(m);
            }
            break;
        }
    }


    for (int p : prime_factors)
    {
        if (a1 % p == 0)
        {
            return false;
        }
    }
    return true;


}

void lc(int m, int a, int c, int x0, int n, string file_name)
{
    ofstream outFile(file_name);

    if (check_lc_conditions(m, a, c))
    {
        cout << "\nУсловия теоремы не выполнены.\n\n";
    }



    int xn = x0;
    cout << "Прогресс генерации ПСЧ: \n";
    int step = n / 10;
    for (size_t i = 0; i < n; i++)
    {
        if (i % step == 0) {
            cout << '\r' << flush;
            cout << "  * Выполнено " << (i * 100) / n << "%";
        }

        outFile << xn << ',';
        xn = (a * x0 + c) % m;
        x0 = xn;
    }

    cout << '\r' << flush;
    cout << "  * Выполнено 100% \n" << "Результат генерации ПСЧ записан в " << file_name <<  "\n";

    outFile.close();
```

```
}


void add(int m, int k, int j, vector<int> lag, int n, string file_name)
{
    ofstream outFile(file_name);

    cout << "Прогресс генерации ПСЧ: \n";
    int step = n / 10;

    int xn;
    for (size_t i = 0; i < j; i++)
    {
        outFile << lag[i] << ',';
    }

    for (size_t i = 55; i < n; i++)
    {
        if (i % step == 0) {
            cout << '\r' << flush;
            cout << "  * Выполнено " << (i * 100) / n << "%";
        }

        xn = (lag[j - k] + lag[0]) % m;
        lag.erase(lag.begin());
        lag.push_back(xn);

        outFile << xn << ',';
    }

    cout << '\r' << flush;
    cout << "  * Выполнено 100% \n" << "Результат генерации ПСЧ записан в " << file_name << "\n";

    outFile.close();
}


void lfsr(string coef, int seed, int n, string file_name)
{
    ofstream outFile(file_name);
    const unsigned int bit_size = coef.length();

    dynamic_bitset<> reg(bit_size, seed);
    dynamic_bitset<> coefs(coef);

    cout << "Прогресс генерации ПСЧ: \n";
    const int step = n / 10;
```

```
    for (size_t i = 0; i < n; i++)
    {
        if (i % step == 0) {
            cout << '\r' << flush;
            cout << "  * Выполнено " << (i * 100) / n << "%";
        }

        bool new_bit = false;
        for (size_t j = 0; j < bit_size; j++)
        {
            if (coefs[j])
            {
                new_bit ^= reg[j];
            }
        }

        reg >>= 1;
        reg.set(bit_size - 1, new_bit);

        unsigned int xn = static_cast<unsigned int>(reg.to_ulong());
        outFile << xn << ',';
    }

    cout << '\r' << flush;
    cout << "  * Выполнено 100% \n" << "Результат генерации ПСЧ записан в " << file_name << "\n";

    outFile.close();
}


void p5(int p, int q1, int q2, int q3, int w, unsigned long long seed, int n, string file_name)
{
    ofstream outFile(file_name);

    dynamic_bitset<> reg(p, seed);
    dynamic_bitset<> res_reg(w, 0);

    cout << "Прогресс генерации ПСЧ: \n";
    int step = n / 10;

    for (size_t i = 0; i < n; i++)
    {
        //if (i % step == 0) {
        //    cout << '\r' << flush;
        //    cout << "  * Выполнено " << (i * 100) / n << "%";
        //}
        bool new_bit = false;
        for (size_t b = 0; b < w; b++)
```

```
        {
            new_bit = reg[q1] ^ reg[q2] ^ reg[q3] ^ reg[0];
            reg >>= 1;
            reg.set(p - 1, new_bit);
            res_reg.set(w - 1 - b, new_bit);
        }

        unsigned int xn = static_cast<unsigned int>(res_reg.to_ulong());
        outFile << xn << ',';
    }

    cout << '\r' << flush;
    cout << "  * Выполнено 100% \n" << "Результат генерации ПСЧ записан в " << file_name << "\n";

    outFile.close();
}


void nfsr(string scoefs1, string scoefs2, string scoefs3, int seed1, int seed2, int seed3, int w, int
{
    ofstream outFile(file_name);
    const unsigned int bit_size1 = scoefs1.length();
    const unsigned int bit_size2 = scoefs2.length();
    const unsigned int bit_size3 = scoefs3.length();

    dynamic_bitset<> reg1(bit_size1, seed1);
    dynamic_bitset<> coefs1(scoefs1);

    dynamic_bitset<> reg2(bit_size2, seed2);
    dynamic_bitset<> coefs2(scoefs2);

    dynamic_bitset<> reg3(bit_size3, seed3);
    dynamic_bitset<> coefs3(scoefs3);

    dynamic_bitset<> res_reg(w, 0);

    cout << "Прогресс генерации ПСЧ: \n";
    const int step = n / 10;

    for (size_t i = 0; i < n; i++)
    {
        if (i % step == 0) {
            cout << '\r' << flush;
            cout << "  * Выполнено " << (i * 100) / n << "%";
        }

        for (size_t b = 0; b < w; b++)
        {
```

```
        // R1
        bool bit1 = false;
        for (size_t j = 0; j < bit_size1; j++)
        {
            if (coefs1[j])
            {
                bit1 ^= reg1[j];
            }
        }
        reg1 >>= 1;
        reg1.set(bit_size1 - 1, bit1);

        // R2
        bool bit2 = false;
        for (size_t j = 0; j < bit_size2; j++)
        {
            if (coefs2[j])
            {
                bit2 ^= reg2[j];
            }
        }
        reg2 >>= 1;
        reg2.set(bit_size2 - 1, bit2);

        // R3
        bool bit3 = false;
        for (size_t j = 0; j < bit_size3; j++)
        {
            if (coefs3[j])
            {
                bit3 ^= reg3[j];
            }
        }
        reg3 >>= 1;
        reg3.set(bit_size3 - 1, bit3);

        res_reg.set(w - 1 - b, (bit1 & bit2) ^ (bit2 & bit3) ^ bit3);
    }

    unsigned int xn = static_cast<unsigned int>(res_reg.to_ulong());
    outFile << xn << ',';
}

cout << '\r' << flush;
cout << "  * Выполнено 100% \n" << "Результат генерации ПСЧ записан в " << file_name << "\n";

outFile.close();
```

```
}


void mt(int m, int seed, int n, string file_name)
{
    ofstream outFile(file_name);

    cout << "Прогресс генерации ПСЧ: \n";
    int step = n / 10;

    MersenneTwister32 mt32(seed);
    for (size_t i = 0; i < n; i++)
    {
        if (i % step == 0) {
            cout << '\r' << flush;
            cout << "  * Выполнено " << (i * 100) / n << "%";
        }

        outFile << mt32.generate() % m << ',';
    }

    cout << '\r' << flush;
    cout << "  * Выполнено 100% \n" << "Результат генерации ПСЧ записан в " << file_name << "\n";

    outFile.close();
}


void rc4(vector <int> k, int n, string file_name)
{
    ofstream outFile(file_name);

    cout << "Прогресс генерации ПСЧ: \n";
    int step = n / 10;

    int l = k.size();
    vector <int> s;
    for (size_t i = 0; i < l; i++)
    {
        s.push_back(i);
    }
    int j = 0;
    for (size_t i = 0; i < l; i++)
    {
        j = (j + s[i] + k[i]) % l;
        swap(s[i], s[j]);
    }
```

```
    int i = 0;
    int t;
    j = 0;
    for (size_t k = 0; k < n; k++)
    {
        if (k % step == 0) {
            cout << '\r' << flush;
            cout << "  * Выполнено " << (k * 100) / n << "%";
        }

        i = (i + 1) % l;
        j = (j + s[i]) % l;
        swap(s[i], s[j]);
        t = (s[i] + s[j]) % l;

        outFile << s[t] << ',';
    }

    cout << '\r' << flush;
    cout << "  * Выполнено 100% \n" << "Результат генерации ПСЧ записан в " << file_name << "\n";

    outFile.close();
}


void rsa(int pq, int e, int x0, int w, int n, string file_name)
{
    ofstream outFile(file_name);

    dynamic_bitset<> z(w, 0);

    unsigned int xi = x0;

    cout << "Прогресс генерации ПСЧ: \n";
    const int step = n / 10;

    for (size_t i = 0; i < n; i++)
    {
        if (i % step == 0) {
            cout << '\r' << flush;
            cout << "  * Выполнено " << (i * 100) / n << "%";
        }

        bool new_bit = false;
        for (size_t j = 0; j < w; j++)
        {
            unsigned int x = xi;
            for (size_t d = 1; d < e; d++)
```

```
            {
                xi = xi * x % pq;
            }
            z.set(w - 1 - j, xi & 1);
        }

        unsigned int xn = static_cast<unsigned int>(z.to_ulong());
        outFile << xn << ',';
    }

    cout << '\r' << flush;
    cout << "  * Выполнено 100% \n" << "Результат генерации ПСЧ записан в " << file_name << "\n";

    outFile.close();
}


void bbs(int x0, int w, int n, string file_name)
{
    ofstream outFile(file_name);
    dynamic_bitset<> z(w, 0);

    const unsigned int pq = 16637;
    unsigned int xi = x0 * x0 % pq;

    cout << "Прогресс генерации ПСЧ: \n";
    const int step = n / 10;

    for (size_t i = 0; i < n; i++)
    {
        if (i % step == 0) {
            cout << '\r' << flush;
            cout << "  * Выполнено " << (i * 100) / n << "%";
        }

        bool new_bit = false;
        for (size_t j = 0; j < w; j++)
        {
            xi = xi * xi % pq;
            z.set(w - 1 - j, xi & 1);
            cout << (xi & 1) << " ";
        }

        unsigned int xn = static_cast<unsigned int>(z.to_ulong());
        outFile << xn << ',';
    }

    cout << '\r' << flush;
```

```
    cout << "  * Выполнено 100% \n" << "Результат генерации ПСЧ записан в " << file_name << "\n";

    outFile.close();
}

int main(int argc, char* argv[])
{
    setlocale(LC_ALL, "Russian");

    argparse::ArgumentParser parser("Генератор псевдослучайных чисел", "1.4");

    parser.set_prefix_chars("-/");
    parser.set_assign_chars("=:");

    parser.add_argument("/g")
        .help("Методы генерации ПСЧ: lc, add, 5p, lfsr, nfsr, mt, rc4, rsa");

    parser.add_argument("/i")
        .help(R"(Инициализационный вектор генератора (параметры записываются через запятую).
                * lc: m, a, c, x0
                * add: m, k, j, j начальных значений
                * 5p: p, q1, q2, q3, w, начальное значение
                * lfsr: двоичный вектор коэффициентов (до 32 бит), начальное значение регистра
                * nfsr: три двоичных вектора коэффициентов (до 32 бит), начальные значения трёх реги
                * mt: модуль, начальное значение
                * rc4: 256 начальных значений
                * rsa: модуль n, число e, начальное значение x, битность w
                * bbs: начальное значение x (взаимно простое с n = 16637), битность w)");

    parser.add_argument("/n")
        .help("Количество генерируемых чисел")
        .default_value(int(10000))
        .scan<'i', int>();

    parser.add_argument("/f")
        .help("Имя файла для вывода")
        .default_value(string("rnd.dat"));

    try {
        parser.parse_args(argc, argv);
    }
    catch (const runtime_error& err) {
        cout << err.what() << endl;
        cout << parser;
        return 1;
    }

    vector <int> i_vec;
```

```
vector <string> is_vec; // TODO: Возможно стоит придумать что-то получше для lfsr
string method_code;
int n = 10000;
string file_name = "rnd.dat";

if (parser.is_used("/g")) {
    method_code = parser.get("/g");
    //cout << "/g: " << method_code << "\n";
}

if (parser.is_used("/i")) {
    string i_str = parser.get<string>("/i");

    stringstream ss(i_str);
    string item;

    if (method_code == "lfsr" or method_code == "nfsr" or method_code == "5p")
    {
        while (getline(ss, item, ','))
        {
            is_vec.push_back(item);
        }
    }
    else
    {
        while (getline(ss, item, ','))
        {
            i_vec.push_back(stoi(item));
        }
    }
    //cout << "/i: ";
    //for (auto x : i_vec)
    //{
    //    cout << x << " ";
    //}
    //cout << "\n";
}

if (parser.is_used("/n")) {
    n = parser.get<int>("/n");
    //cout << "/n: " << n << "\n";
}

if (parser.is_used("/f")) {
    file_name = parser.get("/f");
    //cout << "/f: " << file_name << "\n";
}
```

```
    if (method_code == "lc") {
        lc(i_vec[0], i_vec[1], i_vec[2], i_vec[3], n, file_name);
    }
    else if (method_code == "add")
    {
        int k = i_vec[0];
        int j = i_vec[1];
        int m = i_vec[2];
        i_vec.erase(i_vec.begin(), i_vec.begin() + 3);
        add(k, j, m, i_vec, n, file_name);
    }
    else if (method_code == "lfsr")
    {
        lfsr(is_vec[0], stoi(is_vec[1]), n, file_name);
    }
    else if (method_code == "5p")
    {
        p5(stoi(is_vec[0]), stoi(is_vec[1]), stoi(is_vec[2]), stoi(is_vec[3]), stoi(is_vec[4]), stoul
    }
    else if (method_code == "nfsr")
    {
        nfsr(is_vec[0], is_vec[1], is_vec[2], stoi(is_vec[3]), stoi(is_vec[4]), stoi(is_vec[5]), stoi
    }
    else if (method_code == "mt")
    {
        mt(i_vec[0], i_vec[1], n, file_name);
    }
    else if (method_code == "rc4")
    {
        rc4(i_vec, n, file_name);
    }
    else if (method_code == "rsa")
    {
        rsa(i_vec[0], i_vec[1], i_vec[2], i_vec[3], n, file_name);
    }
    else if (method_code == "bbs")
    {
        bbs(i_vec[0], i_vec[1], n, file_name);
    }


    return 0;

}
```

ПРИЛОЖЕНИЕ Б
Код задания 2

```
#include <iostream>
#include <fstream>
#include <vector>
#include <algorithm>
#include <cmath>
#include <argparse.hpp>

using namespace std;

void st(vector <double> &u, double a, double b)
{
    for (size_t i = 0; i < u.size(); i++)
    {
        u[i] = b * u[i] + a;
    }
}

void tr(vector <double>& u, double a, double b)
{
    double u0 = u[0]; // надо в цикле сохранять
    for (size_t i = 0; i < u.size() - 1; i++)
    {
        u[i] = b * (u[i] + u[i + 1]) + a;
    }
    // Здесь U1 - последнее число, U2 - самое первое
    u[u.size() - 1] = b * (u[u.size() - 1] + u0) + a;
}

void ex(vector <double>& u, double a, double b)
{
    for (size_t i = 0; i < u.size(); i++)
    {
        u[i] = -b * log(u[i]) + a;
    }
}

void nr(vector <double>& u, double mu, double sigma)
{
    const double PI = acos(-1.0);
    for (size_t i = 0; i < u.size() - 1; i += 2)
    {
        double u1 = u[i];
        u[i] = mu + sigma * sqrt(-2 * log(1 - u[i])) * cos(2 * PI * u[i + 1]);
        u[i + 1] = mu + sigma * sqrt(-2 * log(1 - u1)) * sin(2 * PI * u[i + 1]);
    }
}
```

```
void gm(vector <double>& u, double a, double b, double c)
{
    vector <double> u_copy(u);
    if (floor(c) == c)
    {
        int len = u.size();
        double lg_uk;
        for (size_t i = 0; i < len; i++)
        {
            lg_uk = 1;
            for (size_t j = 0; j < c; ++j)
            {
                // Модуль len нужен для циклического сдвига "окна"
                lg_uk *= (1 - u_copy[(i + j) % len]);
            }
            u[i] = a - b * log(lg_uk);
        }
    }
    else
    {
        cout << "Число c должно быть целым!\n";
    }

}


void ln(vector <double>& u, double a, double b)
{
    nr(u, a, b);
    for (size_t i = 0; i < u.size(); i++)
    {
        u[i] = a + exp(b - u[i]);
    }
}


void ls(vector <double>& u, double a, double b)
{
    for (size_t i = 0; i < u.size(); i++)
    {
        u[i] = a + b * log(u[i] / (1 - u[i]));
    }
}


int binomialCoeff(int n, int k)
{
```

```
    if (k > n)
        return 0;
    if (k == 0 || k == n)
        return 1;

    return binomialCoeff(n - 1, k - 1)
        + binomialCoeff(n - 1, k);
}

void bi(vector <double>& u, double a, double b)
{

    for (size_t i = 0; i < u.size(); i++)
    {
        double s = 0;
        int k = 0;
        while (true)
        {
            s += binomialCoeff(b, k) * pow(a, k) * pow((1 - a), b - k);
            if (s > u[i])
            {
                u[i] = k;
                break;
            }
            if (k < b - 1)
            {
                k += 1;
                continue;
            }
            u[i] = b;
        }
    }
}

int main(int argc, char* argv[])
{
    setlocale(LC_ALL, "Russian");

    argparse::ArgumentParser parser("Преобразование ПСЧ к заданному распределению", "1.1");

    parser.set_prefix_chars("-/");
    parser.set_assign_chars("=:");

    parser.add_argument("/d")
        .help("Распределения: st, tr, ex, nr, gm, ln, ls, bi");

    parser.add_argument("/f")
        .help("Имя входного файла")
```

```
    .default_value(string("rnd.dat"));

parser.add_argument("/p1")
    .help("Первый параметр")
    .scan<'g', double>();

parser.add_argument("/p2")
    .help("Второй параметр")
    .scan<'g', double>();

parser.add_argument("/p3")
    .help("Третий параметр (для гамма распределения)")
    .scan<'g', double>();

try {
    parser.parse_args(argc, argv);
}
catch (const runtime_error& err) {
    cout << err.what() << endl;
    cout << parser;
    return 1;
}

string method_code;
string infile = "D:/CSIT/TPRG/TPRG_Task_1/rnd.dat";
string outfile = "";
double p1, p2, p3;

if (parser.is_used("/d")) {
    method_code = parser.get("/d");
    outfile = "distr_" + method_code + ".dat";
}
else
{
    return 1;
}

if (parser.is_used("/f")) {
    infile = parser.get("/f");
}

if (parser.is_used("/p1")) {
    p1 = parser.get<double>("/p1");
}

if (parser.is_used("/p2")) {
    p2 = parser.get<double>("/p2");
}
```

```
if (parser.is_used("/p2")) {
    p2 = parser.get<double>("/p2");
}

if (parser.is_used("/p3")) {
    p3 = parser.get<double>("/p3");
}

const int max = 1000; // ?Нужно ли считать максимальное число в файле?

// Считывание чисел из файла и приведение их к случайной величине
vector <double> numbers;
ifstream inputFile(infile);
if (inputFile)
{
    string line;
    while (getline(inputFile, line))
    {
        stringstream ss(line);
        string numberString;
        while (getline(ss, numberString, ','))
        {
            double number = stod(numberString);
            numbers.push_back(number / max);
        }
    }
    inputFile.close();
}
else
{
    cout << "Не удалось открыть файл. \n";
    return 1;
}

if (method_code == "st")
{
    st(numbers, p1, p2);
}
else if (method_code == "tr")
{
    tr(numbers, p1, p2);
}
else if (method_code == "ex")
{
    ex(numbers, p1, p2);
}
else if (method_code == "nr")
```

```
    {
        nr(numbers, p1, p2);
    }
    else if (method_code == "gm")
    {
        gm(numbers, p1, p2, p3);
    }
    else if (method_code == "ln")
    {
        ln(numbers, p1, p2);
    }
    else if (method_code == "ls")
    {
        ls(numbers, p1, p2);
    }
    else if(method_code == "bi")
    {
        bi(numbers, p1, p2);
    }

    ofstream outFile(outfile);

    for (double x : numbers)
    {
        outFile << x << ',';
    }

    cout << "Результат записан в " << outfile << "\n";

    outFile.close();

    return 0;
}
```