

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

ТЕОРИЯ ПСЕВДОСЛУЧАЙНЫХ ГЕНЕРАТОРОВ

студента 4 курса 431 группы
направления 10.05.01 — Компьютерная безопасность
факультета КНиИТ
Токарева Никиты Сергеевича

Проверено:
доцент

И. И. Слеповичев

СОДЕРЖАНИЕ

1	Генератор псевдослучайных чисел	3
1.1	Линейный конгруэнтный метод	3
1.2	Аддитивный метод	3
1.3	Пятипараметрический метод	4
1.4	Регистр сдвига с обратной связью (РСЛОС)	4
1.5	Нелинейная комбинация РСЛОС	5
1.6	Вихрь Мерсенна	5
1.7	RC4	6
1.8	ГПСЧ на основе RSA	7
1.9	Алгоритм Блум-Блюма-Шуба	7
2	Преобразование ПСЧ к заданному распределению	8
2.1	Метод генерации случайной величины	8
2.2	Стандартное равномерное с заданным интервалом	8
2.3	Треугольное распределение	8
2.4	Общее экспоненциальное распределение	8
2.5	Нормальное распределение	8
2.6	Гамма распределение	9
2.7	Логнормальное распределение	9
2.8	Логистическое распределение	9
2.9	Биномиальное распределение	9
Приложение А	Код задания 1	10
Приложение Б	Код задания 2	23

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 Блума-Блума-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:

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

Параметры:

- p, q_1, q_2, q_3 , коэффициенты пятипараметрического метода;
- w , размерность чисел в битах (целое положительное число);
- начальное значение регистра (целое положительное число).

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, \dots, 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 $R1, R2, R3$;
- w , длина слова;
- x_1, x_2, x_3 — десятичное представление начальных состояний регистров $R1, R2, R3$.

1.6 Вихрь Мерсенна

Псевдокод ниже (источник: википедия) представляет собой алгоритм генерации ПСЧ с помощью вихря Мерсенна:

```
// 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 >> 1)
```

```

    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

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

1. Инициализация S_i .

2. $i = 0, j = 0$.
3. Итерация алгоритма:
 - а) $i = (i + 1) \bmod 256$;
 - б) $j = (j + S_i) \bmod 256$;
 - в) $Swap(S_i, S_j)$;
 - г) $t = (S_i + S_j) \bmod 256$;
 - д) $K = S_t$.

Параметры:

- 256 начальных значений S_i .

1.8 ГПСЧ на основе RSA

Очевидным недостатком этого ГПСЧ на основе алгоритма шифрования является низкая скорость и громоздкость реализации. Описание алгоритма:

1. Инициализация чисел: $n = pq$, где p и q простые числа, числа e : $1 < e < f$, $\text{НОД}(e, f) = 1$, $f = (p - 1)(q - 1)$ и числа x_0 из интервала $[1, n - 1]$.
2. For $i = 1$ to w do
 - а) $x_i \leftarrow x_{i-1}^e \bmod n$.
 - б) $z_i \leftarrow$ последний значащий бит x_i
3. Вернуть z_1, \dots, 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 $x_0 = x^2 \bmod n$, которое будет начальным вектором.
3. For $i = 1$ to w do
 - а) $x_i \leftarrow x_{i-1}^2 \bmod n$.
 - б) $z_i \leftarrow$ последний значащий бит x_i
4. Вернуть z_1, \dots, 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: $U = X/m$. Далее будут перечислены формулы преобразования случайных последовательностей к тому или иному распределению с учетом значения U .

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

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

2.9 Биномиальное распределение

```
y = binominal(x, a, b, m):  
u = U(x)  
s = 0  
k = 0  
Начало цикла:  
  s = s + C(k, b) * a^k * (1 - a)^(b - k)  
  if s > u:  
    y = k  
    Завершить  
  if k < b - 1:  
    k = k + 1  
    Перейти к новой итерации цикла  
y = b
```

ПРИЛОЖЕНИЕ А

Код задания 1

```
1  #include<iostream>
2  #include<vector>
3  #include<numeric>
4  #include<cstring>
5  #include<string>
6  #include<fstream>
7  #include<cmath>
8  using namespace std;
9
10
11 struct {
12     string filename = "rnd.dat", method_code = "", i = "";
13     int n = 10000;
14 } Flags_inf;
15
16
17 // For mersenne twister method
18 const int mt_p = 624;
19 const int mt_u = 11;
20 const int mt_s = 7;
21 const int mt_t = 15;
22 const int mt_l = 18;
23 const int mt_q = 397;
24 const long mt_a = 2567483615;
25 const long mt_b = 2636928640;
26 const long mt_c = 4022730752;
27 const long mt_d = 4294967295;
28 const long upp_mask = 0x80000000;
29 const long low_mask = 0x7fffffff;
30
31
32 void linear_congruent_method(vector<int>& seq, vector<int>& p, int n=10000) {
33     seq.push_back(p[3]);
34     for (int i = 1; i < n; i++) {
35         seq.push_back((p[1] * seq[i - 1] + p[2]) % p[0]);
36     }
37 }
38
39
40 void additive_method(vector<int>& seq, vector<int>& p, int m, int k, int j, int n=10000) {
41     int t = p.size(), num;
42     for (int i = 0; i < n; i++) {
43         num = (p[t - k] + p[t - j]) % m;
44         seq.push_back(num);
45         p.push_back(num);
46         t++;
```

```

47     }
48 }
49
50
51 void five_parameter_method( vector<int>&seq, vector<bool>& xs, int p, int q1
52                             , int q2, int q3, int w, int n=10000 ) {
53     int num, k = 0;
54     string s;
55     for (int i = 1; i < n; i++) {
56         s = "";
57         for (int j = 0; j < w; j++) {
58             num = (xs[k + q1] ^ xs[k + q2] ^ xs[k + q3] ^ xs[k]) & 1;
59             k++;
60             s += to_string(num);
61             xs.push_back(num);
62         }
63         seq.push_back(stoi(s, nullptr, 2));
64     }
65 }
66
67
68 void get_binaryarr(vector<bool>& x, int num) {
69     int d = 0;
70     for (int i = 0; i < 33; i++) {
71         if (pow(2, i) > num) {
72             d = i;
73             break;
74         }
75     }
76     for (int i = d - 1; i >= 0; i--) {
77         int k = num >> i;
78         if (k & 1)
79             x.push_back(1);
80         else
81             x.push_back(0);
82     }
83 }
84
85
86 void shift_arr(vector<bool>& a, int new_el, char fl='l') {
87     if (fl == 'l') {
88         a.erase(a.begin());
89         a.push_back(new_el);
90     }
91     else {
92         vector<bool> tmp;
93         tmp.push_back(new_el);
94         for (int i = 1; i < a.size(); i++)

```

```

95         tmp.push_back(a[i - 1]);
96     a = tmp;
97 }
98 }
99
100
101 void lfsr(vector<int>& seq, vector<bool>& coefs, vector<bool>& x, int n=10000) {
102     int num, len = x.size(), w = x.size();
103     string s;
104     for (int k = 0; k < n; k++) {
105         s = "";
106         for (int j = 0; j < w; j++) {
107             num = coefs[0] * x[0];
108             for (int i = 1; i < len; i++) {
109                 if (x[i])
110                     num ^= coefs[i];
111             }
112             s += to_string(num);
113             shift_arr(coefs, num, 'r');
114         }
115         seq.push_back(stoi(s, nullptr, 2));
116     }
117 }
118
119
120 int get_bit_nfsr(vector<bool>& r, int n) {
121     int b = r[0];
122     for (int i = 1; i < n; i++)
123         b ^= r[i];
124     shift_arr(r, b, 'l');
125     return b;
126 }
127
128
129 void nfsr(vector<int>& seq, vector<bool>& r1, vector<bool>& r2, vector<bool>& r3, int n=10000) {
130     int b1, b2, b3, w;
131     int n1 = r1.size(), n2 = r2.size(), n3 = r3.size();
132     w = max(n1, n2);
133     w = max(w, n3);
134     string s;
135     for (int k = 0; k < n; k++) {
136         s = "";
137         for (int j = 0; j < w; j++) {
138             b1 = get_bit_nfsr(r1, n1);
139             b2 = get_bit_nfsr(r2, n2);
140             b3 = get_bit_nfsr(r3, n3);
141             s += to_string((b1 & b2) ^ (b2 & b3) ^ b3);
142         }

```

```

143     seq.push_back(stoi(s, nullptr, 2));
144 }
145 }
146
147
148 long get_kth_lowest_bits(long long num, int k) {
149     return (((1L << k) - 1) & num);
150 }
151
152
153 template<typename T>
154 int count_bits(T num) {
155     int c = 1;
156     for (num; num >>= 1;)
157         c += 1;
158     return c;
159 }
160
161
162 void initialize_mt(vector<long>& x, int p, int x0, int w) {
163     long long f = 1812433253, num;
164     x.push_back(x0);
165     for (int i = 1; i < p; i++) {
166         num = (f * (x[i - 1] ^ (x[i - 1] >> (w - 2)))) + i;
167         x.push_back(get_kth_lowest_bits(num, w));
168     }
169 }
170
171
172 void mt(vector<int>& seq, vector<long> xs, int m, int n=10000) {
173     int ind = mt_p;
174     for (int cnt = 0; cnt < n; cnt++) {
175         if (ind >= mt_p) {
176             for (int i = 0; i < mt_p; i++) {
177                 long num = (xs[i] & upp_mask) + (xs[(i + 1) % mt_p] & low_mask);
178                 long xA = num >> 1;
179                 if (num & 1) xA ^= mt_a;
180                 xs[i] = xs[(i + mt_q) % mt_p] ^ xA;
181             }
182             ind = 0;
183         }
184         long y = xs[ind];
185         y ^= ((y >> mt_u) & mt_d);
186         y ^= ((y << mt_s) & mt_b);
187         y ^= ((y << mt_t) & mt_c);
188         y ^= (y >> 1);
189         seq.push_back(y % m);
190         ind++;

```

```

191     }
192 }
193
194
195 void rc4(vector<int>& seq, vector<int>& k, int b_len, int n=10000) {
196     vector<int> s(b_len);
197     if (k.empty()) {
198         return;
199     }
200     for (int i = 0; i < b_len; i++) s[i] = i;
201     int i, j = 0, t;
202     for (i = 0; i < b_len; i++) {
203         j = (j + s[i] + k[i]) % b_len;
204         swap(s[i], s[j]);
205     }
206     i = 0, j = 0;
207     for (int p = 0; p < n; p++) {
208         i = (i + 1) % b_len;
209         j = (j + s[i]) % b_len;
210         swap(s[i], s[j]);
211         t = (s[i] + s[j]) % b_len;
212         seq.push_back(s[t]);
213     }
214 }
215
216
217 int get_bitrait (int i) {
218     int n = 1;
219     while (i > 1) {
220         n *= 10;
221         i--;
222     }
223     return n;
224 }
225
226
227 bool check_prime(int n) {
228     if (n == 0 || n == 1) return false;
229     for (int i = 2; i < n / 2; i++)
230         if (n % i == 0) return false;
231     return true;
232 }
233
234
235 // is not used
236 int gen_rand_num(int upp, int low) {
237     srand((unsigned) time(NULL));
238     int n = (rand() % (upp - low + 1)) + low;

```

```

239     while (!check_prime(n))
240         n = (rand() % (upp - low + 1)) + low;
241     return n;
242 }
243
244
245 long long power(long long base, long long exp, int mod) {
246     long long res = 1;
247     while (exp > 0) {
248         if (exp % 2 == 1)
249             res = (res * base) % mod;
250         exp = exp >> 1;
251         base = (base * base) % mod;
252     }
253     return res;
254 }
255
256
257 void rsa(vector<int>& seq, int n, int e, int x0, int l=10000) {
258     int num, w = count_bits(x0);
259     string s;
260     cout << "Input parameters:\n";
261     cout << "n=" << n << " e=" << e << " x=" << x0 << endl;
262     num = x0;
263     for (int i = 1; i < l; i++) {
264         s = "";
265         for (int j = 0; j < w; j++) {
266             num = power(num, e, n);
267             s += to_string(num & 1);
268         }
269         seq.push_back(stoi(s, nullptr, 2));
270     }
271 }
272
273
274 void blum_blum_shub_algo(vector<int>& seq, int x0, int l=10000) {
275     int num, n = 16637, w = count_bits(x0);
276     string s;
277     num = x0;
278     for (int i = 1; i < l; i++) {
279         s = "";
280         for (int j = 0; j < w; j++) {
281             num = power(num, 2, n);
282             s += to_string(num & (1 << 0));
283         }
284         seq.push_back(stoi(s, nullptr, 2));
285     }
286 }

```

```

287
288
289 void get_coeffs_from_string(vector<bool>& key, string& s) {
290     string num = s.substr(0, s.find(","));
291     for (int i = 0; i < num.length(); i++) {
292         if (num[i] != '0' && num[i] != '1') {
293             key.clear();
294             break;
295         }
296         key.push_back((int) num[i] - '0');
297     }
298     s = s.substr(s.find(",") + 1);
299 }
300
301
302 int convert_parameters(string& s) {
303     int num = stoi(s.substr(0, s.find(",")));
304     s = s.substr(s.find(",") + 1);
305     return num;
306 }
307
308
309 void find_multiply_of_number(int aM1, int m) {
310     int k = 2;
311     while (k <= m / 2) {
312         if (m % k == 0 && check_prime(k) && aM1 % k == 0)
313             break;
314         k++;
315     }
316     if (k <= m / 2)
317         cout << "a - 1 is a multiple of p = " << k << " that "
318             << "is a divisor of m\n";
319     else
320         cout << "There is no prime number p such that a is a multiple of p\n";
321 }
322
323
324 vector<int> define_method(string code) {
325     vector<int> seq;
326     if (code == "lc") {
327         vector<int> p;
328         string params = Flags_inf.i;
329         for (int i = 0; i < 4; i++) {
330             if (params.find(",") == -1) {
331                 p.push_back(convert_parameters(params));
332                 break;
333             }
334             p.push_back(convert_parameters(params));

```



```

335     }
336     if (p.size() != 4) return seq;
337     if ( p[0] <= 0 || p[3] < 0 || p[3] > p[0] ||
338         p[1] < 0 || p[1] > p[0] || p[2] < 0 || p[2] > p[0] )
339         return seq;
340     cout << "m = " << p[0] << " a = " << p[1] << " c = "
341         " " << p[2] << " x0 = " << p[3] << endl;
342     if (gcd(p[2], p[0]) != 1)
343         cout << "m and c are not mutually prime\n";
344     else
345         cout << "m and c are mutually prime\n";
346     find_multiply_of_number(p[1] - 1, p[0]);
347     if ((p[1] - 1) % 4 == 0 && p[0] % 4 == 0)
348         cout << "a - 1 is a multiply of 4 and m is a multiply of 4\n";
349     else {
350         if (p[0] % 4 == 0)
351             cout << "a - 1 is not a multiply of 4\n";
352         else if ((p[1] - 1) % 4 == 0)
353             cout << "m is not a multiply of 4\n";
354         else
355             cout << "a - 1 is not a multiply of 4 and m is not a multiply of 4\n";
356     }
357     linear_congruent_method(seq, p, Flags_inf.n);
358 }
359 if (code == "add") {
360     vector<int> p;
361     string params = Flags_inf.i;
362     int m, k, j;
363     m = convert_parameters(params);
364     k = convert_parameters(params);
365     j = convert_parameters(params);
366     for (int i = 0; i < 56; i++) {
367         if (params.find(",") == -1) {
368             p.push_back(convert_parameters(params));
369             break;
370         }
371         p.push_back(convert_parameters(params));
372     }
373     if (k >= j || k < 1 || j < 1 || k > p.size() || j > p.size())
374         return seq;
375     additive_method(seq, p, m, k, j, Flags_inf.n);
376 }
377 if (code == "5p") {
378     vector<bool> xs;
379     string params = Flags_inf.i;
380     int p, q1, q2, q3, w;
381     p = convert_parameters(params);
382     if (params.find(",") == -1) return seq;

```

```

383     q1 = convert_parameters(params);
384     if (params.find(",") == -1) return seq;
385     q2 = convert_parameters(params);
386     if (params.find(",") == -1) return seq;
387     q3 = convert_parameters(params);
388     w = convert_parameters(params);
389     for (int i = 0; i < p; i++) {
390         if (params.find(",") == -1) {
391             xs.push_back(convert_parameters(params));
392             break;
393         }
394         xs.push_back(convert_parameters(params));
395     }
396     five_parameter_method(seq, xs, p, q1, q2, q3, w, Flags_inf.n);
397 }
398 if (code == "lfsr") {
399     int x;
400     vector<bool> c, xs;
401     string params = Flags_inf.i;
402     get_coeffs_from_string(c, params);
403     x = convert_parameters(params);
404     get_binaryarr(xs, x);
405     if (xs.size() > c.size())
406         return seq;
407     lfsr(seq, c, xs, Flags_inf.n);
408 }
409 if (code == "nfsr") {
410     vector<bool> r1, r2, r3;
411     string params = Flags_inf.i;
412     get_coeffs_from_string(r1, params);
413     if (params.find(",") == -1) return seq;
414     get_coeffs_from_string(r2, params);
415     get_coeffs_from_string(r3, params);
416     if (r1.size() < 1 || r2.size() < 1 || r3.size() < 1) return seq;
417     nfsr(seq, r1, r2, r3, Flags_inf.n);
418 }
419 }
420 if (code == "mt") {
421     string params = Flags_inf.i;
422     int m = convert_parameters(params);
423     int x = convert_parameters(params);
424     vector<long> xs;
425     initialize_mt(xs, mt_p, x, count_bits(x));
426     mt(seq, xs, m, Flags_inf.n);
427 }
428 if (code == "rc4") {
429     vector<int> p;
430     string params = Flags_inf.i;

```

```

431     for (int i = 0; i < 257; i++) {
432         if (params.find(",") == -1) {
433             p.push_back(convert_parameters(params) % 256);
434             break;
435         }
436         p.push_back(convert_parameters(params) % 256);
437     }
438     rc4(seq, p, p.size(), Flags_inf.n);
439 }
440 if (code == "rsa") {
441     int n, e, x;
442     string params = Flags_inf.i;
443     n = convert_parameters(params);
444     if (params.find(",") == -1) return seq;
445     e = convert_parameters(params);
446     x = convert_parameters(params);
447     if (1 > x || x > n)
448         return seq;
449     rsa(seq, n, e, x, Flags_inf.n);
450 }
451 if (code == "bbs") {
452     int x0 = stoi(Flags_inf.i);
453     if (gcd(x0, 16637) != 1) {
454         cout << "x0=" << x0 << " and n=16637 must be reciprocal!\n";
455         return seq;
456     }
457     blum_blum_shub_algo(seq, x0, Flags_inf.n);
458 }
459 return seq;
460 }
461
462
463 void advert(string par="") {
464     if (par == "") {
465         cout << "The program has the following commands:\n";
466         cout << "/g:<code_method> --- parameter specifies the"
467             << " method of IF generations.\n" << "code_method has the following values:\n";
468         cout << "\tlc - linear congruent generator\n\tadd - additive method\n"
469             << "\t5p - five-parametric method\n\tlfsr - linear-feedback shift register\n"
470             << "\tnfsr - nonlinear feedback shift register\n\tmt - Mersenne twister\n"
471             << "\trc4 - RC4\n\ttrsa - RSA\n\tbbs - Blum-Blum-Shub\'s algorithm\n\n";
472         cout << "/i:<parameters> --- generator initialization vector.\n";
473         cout << "parameters for methods:\n"
474             << "\t<m,a,c,x0> - parameters for \\'lc'\n"
475             << "\t<m,k,j,x_1,...x_55> - parameters for \\'add'\n"
476             << "\t<p,q1,q2,q3,w,x_1...x_p> - parameters for \\'5p'\n"
477             << "\t<coeffs,num> - parameters for \\'lfsr'\n"
478             << "\t<r1,r2,r3> - parameters for \\'nfsr'\n"

```

```

479         "\t<m, x> - parameters for \"mt\"\\n"
480         "\t<seq> - parameter for \"rc4\". seq - sequence of numbers (1 < sequence length< 257)\\n"
481         "\t<n,e,x> - parameters for \"rsa\"\\n\t<x> - parameter for \"bbs\"\\n";
482     cout << "/n:<length> --- amount of generated numbers.\\n\\n";
483     cout << "/f:<complete_file_name> --- to output generated numbers to file.\\n\\n";
484     cout << "/h --- complete information about commands.\\n\\n";
485 }
486 else if (par == "lc") {
487     cout << "The following parameters must be entered for correct operation:\\n";
488     cout << "/i:<m,a,c,x0> - parameters for \"lc\",\\n where m > 0 - module\\n"
489         "\t0 <= a <= m - multiplier\\n\t0 <= c <= m - increment\\n\t0 <= X0 <= m - initial element\\n";
490     cout<< "Also you can use /n:<length> to set length of the sequence of numbers 0 < n < 10001\\n";
491 }
492 else if (par == "add") {
493     cout << "The following parameters must be entered for correct operation:\\n";
494     cout << "/i:<m,k,j,x_1,x_2,...,x_55> - parameters for \"add\",\\n "
495         "where m - the module, k and j (0 < k < j) - indices of numbers x_1,x_2,...,x_55\\n";
496     cout<< "Also you can use /n:<length> to set length of the sequence of numbers 0 < n < 10001\\n";
497 }
498 else if (par == "5p") {
499     cout << "The following parameters must be entered for correct operation:\\n";
500     cout << "/i:<p,q_1,q_2,q_3,w,x_1,x_2,...,x_p> - parameters for \"5p\",\\n "
501         "where p - length of numbers x_1,x_2,...,x_p, q_1, q_2 and q_3\\n"
502         " - indices of those numbers x_1,x_2,...,x_55, w - bits number";
503     cout<< "Also you can use /n:<length> to set length of the sequence of numbers 0 < n < 10001\\n";
504 }
505 else if (par == "lfsr") {
506     cout << "The following parameters must be entered for correct operation:\\n\\n";
507     cout << "/i:<coeffs,num> - coeffs - sequence of 0 and 1, num - decimal number\\n";
508     cout<< "Also you can use /n:<length> to set length of the sequence of numbers 0 < n < 10001\\n";
509 }
510 else if (par == "nfsr") {
511     cout << "The following parameters must be entered for correct operation:\\n\\n";
512     cout << "/i:<r1,r2,r3> - r1,r2 and r3 - sequences of 0 and 1 numbers\\n";
513     cout<< "Also you can use /n:<length> to set length of the sequence of numbers 0 < n < 10001\\n";
514 }
515 else if (par == "mt") {
516     cout << "The following parameters must be entered for correct operation:\\n\\n";
517     cout << "/i:<m, x> - m - module, x - initial number\\n";
518     cout<< "Also you can use /n:<length> to set length of the sequence of numbers 0 < n < 10001\\n";
519 }
520 else if (par == "rc4") {
521     cout << "The following parameters must be entered for correct operation:\\n\\n";
522     cout << "/i:<sequence of numbers> - sequence of numbers have length more then 0 and less then 255\\n";
523     cout<< "Also you can use /n:<length> to set length of the sequence of numbers 0 < n < 10001\\n";
524 }
525 else if (par == "rsa") {
526     cout << "The following parameters must be entered for correct operation:\\n\\n";

```

```

527     cout << "/i:<m,e,x> - where m - module, e - number for to exponentiate,"
528         " x - initial number (0 < x < m + 1).\n";
529     cout<< "Also you can use /n:<length> to set length of the sequence of numbers 0 < n < 10001\n";
530 }
531 else if (par == "bbs") {
532     cout << "The following parameters must be entered for correct operation:\n\n";
533     cout << "/i:<x> - where x - initial number (0 < x < 16638) and x must be GCD(x, 16637) = 1.\n";
534     cout<< "Also you can use /n:<length> to set length of the sequence of numbers 0 < n < 10001\n";
535 }
536 }
537
538
539 bool check_parameters()
540 {
541     if (Flags_inf.method_code == "") {
542         cout << "Incorrect data! Please enter \'/h\' to see information\n";
543         return false;
544     }
545     else if (Flags_inf.method_code != "" && Flags_inf.i == "") {
546         cout << "Incorrect data! Please enter \'/h\' to see information\n";
547         advert(Flags_inf.method_code);
548         return false;
549     }
550     return true;
551 }
552
553
554 int main(int argc, char* argv[]) {
555     string s;
556     for (int i = 0; i < argc; i++) {
557         if (argv[i][1] == '/') continue;
558         switch (argv[i][1]) {
559             case 'g': {
560                 s = argv[i];
561                 Flags_inf.method_code = s.erase(0, 3);
562                 break;
563             }
564             case 'i': {
565                 s = argv[i];
566                 Flags_inf.i = s.erase(0, 3);
567                 break;
568             }
569             case 'n': {
570                 s = argv[i];
571                 Flags_inf.n = stoi(s.erase(0, 3));
572                 if (Flags_inf.n > 10000) {
573                     cout << "Size of sequence cannot be more 10000!\n";
574                     return 0;

```

```

575         }
576         break;
577     }
578     case 'f': {
579         s = argv[i];
580         Flags_inf.filename = s.erase(0, 3);
581     }
582     break;
583     case 'h': {
584         advert();
585         return 0;
586     }
587     break;
588 }
589 }
590 if (!check_parameters()) {
591     return 0;
592 }
593 vector<int> seq;
594 seq = define_method(Flags_inf.method_code);
595 if (seq.empty()) {
596     cout << "Incorrect parameters input!\n";
597     advert(Flags_inf.method_code);
598     return 0;
599 }
600 ofstream out;
601 out.open(Flags_inf.filename);
602 if (out.is_open()) {
603     for (int i = 0; i < Flags_inf.n - 1; i++)
604         out << seq[i] << ',';
605     out << seq[Flags_inf.n - 1];
606 }
607 out.close();
608 cout << "\nThe sequence of random numbers was generated successfully "
609        "and written to the file called " << Flags_inf.filename << "\n";
610 cout << endl;
611 return 0;
612 }

```

ПРИЛОЖЕНИЕ Б

Код задания 2

```
1  #include<iostream>
2  #include<vector>
3  #include<numeric>
4  #include<cstring>
5  #include<string>
6  #include<fstream>
7  #include<cmath>
8  using namespace std;
9
10 struct {
11     string fileNameRead = "", method_code = "";
12     double p1 = -100000, p2 = -100000, p3 = -100000;
13     int n;
14     double nmax = -1.0;
15 } Flags_inf;
16
17 void get_U(vector<double>& u, int n, double mx) {
18     for (int i = 0; i < n; i++) {
19         u[i] /= mx;
20     }
21 }
22
23 void standard_dist(vector<double>& u, int n, double a, double b) {
24     for (int i = 0; i < n; i++) {
25         u[i] = b * u[i] + a;
26     }
27 }
28
29 void triangle_dist(vector<double>& u, int n, double a, double b) {
30     double u0 = u[0];
31     for (int i = 0; i < n; i++) {
32         u[i] = a + b * (u[i] + u[i + 1] - 1);
33     }
34     u[n - 1] = a + b * (u[n - 1] + u0 - 1);
35 }
36
37 void exponential_dist(vector<double>& u, int n, double a, double b) {
38     for (int i = 0; i < n; i++) {
39         u[i] = -b * log(u[i]) + a;
40     }
41 }
42
43
44 void normal_dist(vector<double>& u, int n, double mu, double sigma) {
45     for (int i = 0; i < n - 1; i += 2) {
46         double u1 = u[i];
```

```

47         u[i] = mu + sigma * sqrt(-2.0 * log(1 - u[i])) * cos(2.0 * M_PI * u[i + 1]);
48         u[i + 1] = mu + sigma * sqrt(-2.0 * log(1 - u[i])) * sin(2.0 * M_PI * u[i + 1]);
49     }
50 }
51
52
53 void gamma_dist(vector<double>& u, int n, double a, double b, double c) {
54     vector<double> t(u);
55     if (floor(c) != c) {
56         c = floor(c);
57         cout << "The third parameter (p3) must be an integer! Therefore, the number c = " << c << endl;
58     }
59     else {
60         double tmp_mult;
61         for (int i = 0; i < n; i++) {
62             tmp_mult = 1;
63             for (int j = 0; j < c; j++) tmp_mult *= 1 - t[(i + j) % n];
64             u[i] = a - b * log(tmp_mult);
65         }
66     }
67 }
68
69
70 void lognormal_dist(vector<double>& u, int n, double a, double b) {
71     normal_dist(u, n, a, b);
72     for (int i = 0; i < n; i++)
73         u[i] = a + exp(b - u[i]);
74 }
75
76
77 void logistic_dist(vector<double>& u, int n, double a, double b) {
78     for (int i = 0; i < n; i++)
79         u[i] = a + b * log(u[i] / (1 - u[i]));
80 }
81
82
83 int binomialCoeff(int n, int k) {
84     if (k > n)
85         return 0;
86     if (k == 0 || k == n)
87         return 1;
88     return binomialCoeff(n - 1, k - 1) + binomialCoeff(n - 1, k);
89 }
90
91
92 void binomial_dist(vector<double>& u, int n, double a, double b) {
93     double s;
94     int k;

```



```

95     for (int i = 0; i < n; i++) {
96         s = 0.0;
97         k = 0;
98         while (true) {
99             s += binomialCoeff(b, k) * pow(a, k) * pow((1 - a), b - k);
100             if (s > u[i]) {
101                 u[i] = k;
102                 break;
103             }
104             if (k < b - 1) {
105                 k++;
106                 continue;
107             }
108             u[i] = b;
109         }
110     }
111 }
112
113
114 bool define_method(vector<double>& dist, string code) {
115     if (code == "st") {
116         standard_dist(dist, Flags_inf.n, Flags_inf.p1, Flags_inf.p2);
117     }
118     else if (code == "tr") {
119         triangle_dist(dist, Flags_inf.n, Flags_inf.p1, Flags_inf.p2);
120     }
121     else if (code == "ex") {
122         exponential_dist(dist, Flags_inf.n, Flags_inf.p1, Flags_inf.p2);
123     }
124     else if (code == "nr") {
125         normal_dist(dist, Flags_inf.n, Flags_inf.p1, Flags_inf.p2);
126     }
127     else if (code == "gm") {
128         if (Flags_inf.p3 == -100000)
129             return false;
130         gamma_dist(dist, Flags_inf.n, Flags_inf.p1, Flags_inf.p2, Flags_inf.p3);
131     }
132     else if (code == "ln") {
133         lognormal_dist(dist, Flags_inf.n, Flags_inf.p1, Flags_inf.p2);
134     }
135     else if (code == "ls") {
136         logistic_dist(dist, Flags_inf.n, Flags_inf.p1, Flags_inf.p2);
137     }
138     else if (code == "bi") {
139         binomial_dist(dist, Flags_inf.n, Flags_inf.p1, Flags_inf.p2);
140     }
141     if (dist.empty())
142         return false;

```

```

143     return true;
144 }
145
146
147 void advert(string par="") {
148     if (par == "") {
149         cout << "The program has the following commands:\n";
150         cout << "/d:<code_distribution> --- allocation code for sequence conversion.\n"
151             "code_method has the following values:\n";
152         cout << "\tst - standard even distribution\n"
153             "\ttr - triangle distribution\n\tex - exponential distribution\n"
154             "\tnr - normal distribution\n\tgm - gamma distribution\n"
155             "\tln - lognormal distribution\n"
156             "\tls - logistic distribution\n"
157             "\tbi - binomial distribution\n";
158         cout << "\nParameters for methods:\n";
159         cout << "/p1:<a> --- first parameter\n/p2:<b> --- second parameter\n"
160             "/p3:<c> --- third parameter (needs only for gamma distribution)\n\n";
161         cout << "/f:<complete_file_name> --- to get generated numbers from file.\n\n";
162         cout << "/h --- complete information about commands.\n\n";
163     }
164     else if ( par == "st" || par == "tr" || par == "nr" ||
165         par == "ln" || par == "ls" || par == "bi") {
166         cout << "The following parameters must be entered for correct operation:\n";
167         cout << "/p1:<a> --- first parameter\n/p2:<b> --- second parameter\n";
168         cout << "/f:<complete_file_name> --- to get generated numbers from file.\n\n";
169     }
170     else if (par == "gm") {
171         cout << "/p1:<a> --- first parameter\n/p2:<b> --- second parameter\n"
172             "/p3<c> --- third parameter";
173         cout << "/f:<complete_file_name> --- to get generated numbers from file.\n\n";
174         cout << "/h --- complete information about commands.\n\n";
175     }
176 }
177
178 double convert_parameters(string& s) {
179     double num = stod(s.substr(0, s.find(", ")));
180     s = s.substr(s.find(", ") + 1);
181     return num;
182 }
183
184
185 bool check_parameters()
186 {
187     if (Flags_inf.method_code == "" || Flags_inf.fileNameRead == "") {
188         cout << "Incorrect data! Please enter \'/h\' to see information\n";
189         return false;
190     }

```

```

191     if (Flags_inf.method_code != "" && (Flags_inf.p1 == -100000 || Flags_inf.p2 == -100000)) {
192         cout << "Incorrect data! Please enter \'/h\' to see information\n";
193         advert(Flags_inf.method_code);
194         return false;
195     }
196     return true;
197 }
198
199
200 int main(int argc, char* argv[]) {
201     string s;
202     for (int i = 0; i < argc; i++) {
203         if (argv[i][1] == '/') continue;
204         switch (argv[i][1]) {
205             case 'd': {
206                 s = argv[i];
207                 Flags_inf.method_code = s.erase(0, 3);
208                 break;
209             }
210             case 'p': {
211                 switch (argv[i][2]){
212                     case '1':
213                         s = argv[i];
214                         Flags_inf.p1 = stod(s.erase(0, 4));
215                         break;
216                     case '2':
217                         s = argv[i];
218                         Flags_inf.p2 = stod(s.erase(0, 4));
219                         break;
220                     case '3':
221                         s = argv[i];
222                         Flags_inf.p3 = stod(s.erase(0, 4));
223                         break;
224                 }
225                 break;
226             }
227             case 'f': {
228                 s = argv[i];
229
230                 Flags_inf.fileNameRead = s.erase(0, 3);
231             }
232             break;
233             case 'h': {
234                 advert();
235                 return 0;
236             }
237             break;
238         }

```

```

239     }
240     if (!check_parameters()) {
241         return 0;
242     }
243     ifstream fin;
244     fin.exceptions(std::ios::badbit | std::ios::failbit) ;
245     string seq;
246     try {
247         fin.open(Flags_inf.fileNameRead);
248         while (!fin.eof()) {
249             fin >> seq;
250         }
251     }
252     catch (const std::exception & e) {
253         cerr << "Error: " << e.what() << endl;
254         cout << "Some problem with file: " << Flags_inf.fileNameRead << "!\n";
255         return 0;
256     }
257     fin.close();
258     vector<double> dist;
259     double tmp, nmax;
260     while (true) {
261         if (seq.find(",") == -1) {
262             tmp = convert_parameters(seq);
263             if (nmax < tmp) nmax = tmp;
264             dist.push_back(tmp);
265             break;
266         }
267         tmp = convert_parameters(seq);
268         if (nmax < tmp) nmax = tmp;
269         dist.push_back(tmp);
270     }
271     Flags_inf.nmax = nmax + 1;
272     Flags_inf.n = dist.size();
273     cout << Flags_inf.n << " numbers was read.\n";
274     cout << "Maximal number of secquence: " << Flags_inf.nmax << endl << endl;
275     get_U(dist, Flags_inf.n, Flags_inf.nmax);
276     bool f1 = define_method(dist, Flags_inf.method_code);
277     if (dist.empty() || !f1) {
278         cout << "Incorrect parameters input!\n";
279         advert(Flags_inf.method_code);
280         return 0;
281     }
282     string fout = "distr-" + Flags_inf.method_code + ".dat";
283     ofstream out;
284     out.open(fout);
285     if (out.is_open()) {
286         for (int i = 0; i < Flags_inf.n - 1; i++)

```

```
287         out << dist[i] << ',';
288     out << dist[Flags_inf.n - 1];
289 }
290 out.close();
291 cout << "\nThe sequence of random numbers was generated successfully "
292         "and written to the file called " << fout << "\n";
293 cout << endl;
294 return 0;
295 }
```