

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

ТЕОРИЯ ПСЕВДОСЛУЧАЙНЫХ ГЕНЕРАТОРОВ
ОТЧЕТ ПО ПРАКТИЧЕСКОМУ КУРСУ

студента 4 курса 431 группы
специальности 10.05.01 — Компьютерная безопасность
факультета компьютерных наук и информационных технологий
Гущина Андрея Юрьевича

Проверил
доцент

И. И. Слеповичев

Саратов 2023

СОДЕРЖАНИЕ

1	Генератор псевдослучайных чисел	3
1.1	Линейный конгруэнтный метод	3
1.2	Аддитивный метод	4
1.3	Пятипараметрический метод	5
1.4	Регистр сдвига с обратной связью (РСЛОС)	6
1.5	Нелинейная комбинация РСЛОС	8
1.6	Вихрь Мерсенна	9
1.7	RC4	12
1.8	ГПСЧ на основе RSA	13
1.9	Алгоритм Блюма-Блюма-Шуба	14
2	Преобразование ПСЧ к заданному распределению	16
2.1	Стандартное равномерное с заданным интервалом	16
2.2	Треугольное распределение	16
2.3	Общее экспоненциальное распределение	17
2.4	Нормальное распределение	18
2.5	Гамма распределение (для параметра $p_3 = k$)	19
2.6	Логнормальное распределение	20
2.7	Логистическое распределение	21
2.8	Биномиальное распределение	22
Приложение А	Главный файл main.rs проекта prng	24
Приложение Б	Файл prgenerator.rs проекта prng	31
Приложение В	Файл парсера аргументов командной строки clp.rs проекта prng	32
Приложение Г	Файл Cargo.toml проекта prng	39
Приложение Д	Главный файл main.rs проекта rnc	40
Приложение Е	Файл prgenerator.rs проекта rnc	43
Приложение Ж	Файл парсера аргументов командной строки clp.rs проекта rnc	43
Приложение З	Файл Cargo.toml проекта rnc	48

1 Генератор псевдослучайных чисел

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

1.1 Линейный конгруэнтный метод

Последовательность ПСЧ, получаемая по формуле:

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

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

Параметры:

- m — модуль
- a — множитель
- c — приращение
- x_0 — начальное значение (десятичное число)

Исходный код генератора:

```
1 use crate::prgenerator::{PRGenerator, MOD};
2
3 pub struct LinearPRG {
4     m: u32,
5     a: u32,
6     c: u32,
7     x: u32,
8 }
9
10 impl LinearPRG {
11     pub fn new(m: u32, a: u32, c: u32, x_0: u32) -> Self {
12         Self { m, a, c, x: x_0 }
13     }
14 }
15
16 impl PRGenerator for LinearPRG {
17     fn next(&mut self) -> u32 {
18         let x = self.x;
19         self.x = (self.a * self.x + self.c) % self.m;
```

```

20         x % MOD
21     }
22 }

```

1.2 Аддитивный метод

Последовательность Фибоначчи с задержкой определяется следующим образом:

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

Параметры:

- m — модуль
- k — младший индекс
- j — старший индекс
- x_0, \dots, x_n — n начальных значений (десятичные числа), $n \geq j$

Исходный код генератора:

```

1  use crate::prgenerator::{PRGenerator, MOD};
2  use std::collections::VecDeque;
3
4  pub struct AdditivePRG {
5      m: u32,
6      j: usize,
7      k: usize,
8      n: usize,
9      vec: VecDeque<u32>,
10 }
11
12 impl AdditivePRG {
13     pub fn new(m: u32, j: usize, k: usize, init: Vec<u32>) -> Self {
14         Self {
15             m,
16             j,
17             k,
18             n: init.len(),
19             vec: init.into(),
20         }
21     }
22 }
23
24 impl PRGenerator for AdditivePRG {
25     fn next(&mut self) -> u32 {

```

```

26         let x =
27             (self.vec[self.n - self.j] + self.vec[self.n - self.k]) % self.m;
28         self.vec.push_back(x);
29         self.vec.pop_front();
30         x % MOD
31     }
32 }

```

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:

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

Параметры:

- p — количество элементов
- q_1 — первый индекс
- q_2 — второй индекс
- q_3 — третий индекс
- w — длина слова в битах
- x_1, \dots, x_p — начальное значение регистра (последовательность 0 и 1)

Исходный код генератора:

```

1 use crate::prgenerator::{PRGenerator, MOD};
2 use bitvec::prelude::*;
3
4 pub struct FiveParamPRG {
5     q1: usize,
6     q2: usize,
7     q3: usize,
8     w: u32,
9     vec: BitVec,
10 }
11
12 impl FiveParamPRG {
13     pub fn new(
14         p: usize,

```

```

15         q1: usize,
16         q2: usize,
17         q3: usize,
18         w: u32,
19         xs: Vec<u32>,
20     ) -> FiveParamPRG {
21         let mut bv = bitvec![0; p];
22         for (i, x) in xs.iter().enumerate() {
23             *bv.get_mut(i).unwrap() = *x != 0;
24         }
25         FiveParamPRG {
26             q1,
27             q2,
28             q3,
29             w,
30             vec: bv,
31         }
32     }
33 }
34
35 impl PRGenerator for FiveParamPRG {
36     fn next(&mut self) -> u32 {
37         let mut word = BitVec::<u32, Msb0>::new();
38         for _ in 0..self.w {
39             let bit = self.vec[0]
40                 ^ self.vec[self.q1 - 1]
41                 ^ self.vec[self.q2 - 1]
42                 ^ self.vec[self.q3 - 1];
43             self.vec.shift_left(1);
44             *self.vec.last_mut().unwrap() = bit;
45             word.push(bit);
46         }
47         word.load_be::<u32>() % MOD
48     }
49 }

```

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.

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

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

Как видно из формулы, для РСЛОС функция обратной связи является линейной булевой функцией от состояний всех или некоторых битов регистра.

Одна итерация алгоритма, генерирующего последовательность, состоит из следующих шагов:

1. Содержимое ячейки $p - 1$ формирует очередной бит ПСП битов.
2. Содержимое ячейки 0 определяется значением функции обратной связи, являющейся линейной булевой функцией с коэффициентами a_1, \dots, a_{p-1} . Его вычисляют по формуле 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:

- a_0, \dots, a_p — вектор коэффициентов (последовательность 0 и 1)
- x_0, \dots, x_p — начальное значение регистра (последовательность 0 и 1)

Исходный код генератора:

```

1 use crate::prgenerator::{PRGenerator, MOD};
2 use std::collections::VecDeque;
3
4 pub struct LfsrPRG {
5     coeff: Vec<u32>,
6     vec: VecDeque<u32>,
7 }
8
9 impl LfsrPRG {
10     pub fn new(coeff: Vec<u32>, init: Vec<u32>) -> LfsrPRG {
11         LfsrPRG {
12             coeff,
13             vec: init.into(),
14         }
15     }
16
17     pub fn next_bit(&mut self) -> u32 {
18         let mut new_x0 = 0;
19         for i in 0..self.coeff.len() {

```

```

20         new_x0 += self.coeff[i] * self.vec[i];
21     }
22     new_x0 %= 2;
23     let new_bit = *self.vec.front().unwrap();
24     self.vec.pop_front();
25     self.vec.push_back(new_x0);
26     new_bit
27 }
28 }
29
30 impl PRGenerator for LfsrPRG {
31     fn next(&mut self) -> u32 {
32         let mut result = 0;
33         for i in 0..std::mem::size_of::<u32>() * 8 {
34             result |= self.next_bit() << i;
35         }
36         result % MOD
37     }
38 }

```

1.5 Нелинейная комбинация РСЛОС

Последовательность получается с помощью нелинейной комбинации трёх РСЛОС:

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

Параметры:

- a_0, \dots, a_n — коэффициенты первого РСЛОС (последовательность 0 и 1)
- b_0, \dots, b_n — коэффициенты второго РСЛОС (последовательность 0 и 1)
- c_0, \dots, c_n — коэффициенты третьего РСЛОС (последовательность 0 и 1)
- w — длина слова в битах
- x_1 — начальное состояние первого РСЛОС (десятичное число)
- x_2 — начальное состояние второго РСЛОС (десятичное число)
- x_3 — начальное состояние третьего РСЛОС (десятичное число)

Исходный код генератора:

```

1 use crate::lfsr::LfsrPRG;
2 use crate::prgenerator::{PRGenerator, MOD};
3 use bitvec::prelude::*;
4
5 pub struct NfsrPRG {

```



```

6      r1: LfsrPRG,
7      r2: LfsrPRG,
8      r3: LfsrPRG,
9      w: u32,
10 }
11
12 impl NfsrPRG {
13     pub fn new(
14         coeffs1: Vec<u32>,
15         init1: Vec<u32>,
16         coeffs2: Vec<u32>,
17         init2: Vec<u32>,
18         coeffs3: Vec<u32>,
19         init3: Vec<u32>,
20         w: u32,
21     ) -> NfsrPRG {
22         NfsrPRG {
23             r1: LfsrPRG::new(coeffs1, init1),
24             r2: LfsrPRG::new(coeffs2, init2),
25             r3: LfsrPRG::new(coeffs3, init3),
26             w,
27         }
28     }
29 }
30
31 impl PRGenerator for NfsrPRG {
32     fn next(&mut self) -> u32 {
33         let mut word = BitVec::<u32, Msb0>::new();
34         for _ in 0..self.w {
35             let bit1 = self.r1.next_bit() > 0;
36             let bit2 = self.r2.next_bit() > 0;
37             let bit3 = self.r3.next_bit() > 0;
38             let bit = bit1 & bit2 ^ bit2 & bit3 ^ bit3;
39             word.push(bit);
40         }
41         word.load_be::<u32>() % MOD
42     }
43 }

```

1.6 Вихрь Мерсенна

Параметры:

— m — модуль

— x — начальное значение (десятичное число)

Исходный код генератора:

```
1 use crate::prgenerator::{PRGenerator, MOD};
2 use std::cmp::Ordering;
3
4 const W: u32 = 32;
5 const N: usize = 624;
6 const M: usize = 397;
7 const R: u32 = 31;
8 const A: u32 = 0x9908B0DF;
9 const U: u32 = 11;
10 const D: u32 = 0xFFFFFFFF;
11 const S: u32 = 7;
12 const B: u32 = 0x9D2C5680;
13 const T: u32 = 15;
14 const C: u32 = 0xEFC60000;
15 const L: u32 = 18;
16 const F: u32 = 1812433253;
17
18 const LOWER_MASK: u32 = (1 << R) - 1;
19 const UPPER_MASK: u32 = !LOWER_MASK;
20
21 pub struct MersennePRG {
22     mt: [u32; N],
23     index: usize,
24     modulo: u32,
25 }
26
27 impl MersennePRG {
28     pub fn new(modulo: u32, x: u32) -> MersennePRG {
29         let mut prg = MersennePRG {
30             mt: [0; N],
31             index: N + 1,
32             modulo,
33         };
34         prg.seed_mt(x);
35         prg
36     }
37
38     fn seed_mt(&mut self, seed: u32) {
```

```

39         self.index = N;
40         self.mt[0] = seed;
41         for i in 1..=N - 1 {
42             let a = (F % self.modulo) as u64;
43             let b = ((self.mt[i - 1] ^ (self.mt[i - 1] >> (W - 2)))
44                 % self.modulo) as u64;
45             let c = (a * b + i as u64) % self.modulo as u64;
46             self.mt[i] = c as u32;
47         }
48     }
49
50     fn twist(&mut self) {
51         for i in 0..N {
52             let x =
53                 (self.mt[i] & UPPER_MASK) | (self.mt[(i + 1) % N] &
54 ↪ LOWER_MASK);
55             let x_a = if x % 2 == 0 { x >> 1 } else { (x >> 1) ^ A };
56             self.mt[i] = self.mt[(i + M) % N] ^ x_a;
57         }
58         self.index = 0;
59     }
60
61     impl PRGenerator for MersennePRG {
62         fn next(&mut self) -> u32 {
63             match self.index.cmp(&N) {
64                 Ordering::Equal => self.twist(),
65                 Ordering::Greater => panic!("Генератор не был инициализирован"),
66                 _ => (),
67             }
68
69             let mut y = self.mt[self.index];
70             y = y ^ ((y >> U) & D);
71             y = y ^ ((y << S) & B);
72             y = y ^ ((y << T) & C);
73             y = y ^ (y >> L);
74
75             self.index += 1;
76             (y % self.modulo) % MOD
77         }
78     }

```

1.7 RC4

Описание алгоритма:

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

Параметры:

— x_0, \dots, x_{256} — начальные значения (256 десятичных чисел)

Исходный код генератора:

```
1 use crate::prgenerator::{PRGenerator, MOD};
2
3 pub struct Rc4PRG {
4     k: Vec<u32>,
5     s: Vec<usize>,
6     i: usize,
7     j: usize,
8 }
9
10 impl Rc4PRG {
11     pub fn new(coeff: Vec<u32>) -> Rc4PRG {
12         let mut prg = Rc4PRG {
13             k: coeff,
14             s: vec![0; 256],
15             i: 0,
16             j: 0,
17         };
18         for i in 0..=255 {
19             prg.s[i] = i;
20         }
21         let mut j = 0;
22         for i in 0..=255 {
```

```

23         j = (j + prg.s[i] + prg.k[i] as usize) % 256;
24         prg.s.swap(i, j);
25     }
26     prg
27 }
28 }
29
30 impl PRGenerator for Rc4PRG {
31     fn next(&mut self) -> u32 {
32         self.i = (self.i + 1) % 256;
33         self.j = (self.j + self.s[self.i]) % 256;
34         self.s.swap(self.i, self.j);
35         let t = (self.s[self.i] + self.s[self.j]) % 256;
36         let k: u32 = self.s[t].try_into().unwrap();
37         k % MOD
38     }
39 }

```

1.8 ГПСЧ на основе RSA

Описание алгоритма:

1. Сгенерировать два секретных простых числа p и q , а также $n = p \cdot q$ и $f = (p - 1)(q - 1)$. Выбрать случайное целое число $e : 1 < e < f$, такое что $\gcd(e, f) = 1$.
2. Выбрать случайное целое x_0 — начальный вектор из интервала $[1, n - 1]$.
3. for $i = 1$ to l do
 - a) $x_i \leftarrow x_{i-1}^e \pmod{n}$
 - б) $z_i \leftarrow$ последний значащий бит x_i
4. Вернуть z_1, \dots, z_l .

Параметры:

- n — модуль $n = p \cdot q$, где p, q — простые числа
- e — такое число, что $1 < e < (p - 1)(q - 1)$, $\gcd(e, (p - 1)(q - 1)) = 1$
- w — длина слова в битах
- x — число из интервала $[1, n]$

Исходный код генератора:

```

1 use crate::prgenerator::{PRGenerator, MOD};
2 use bitvec::prelude::*;
3
4 pub struct RsaPRG {

```

```

5      n: u32,
6      e: u32,
7      w: u32,
8      x: u32,
9  }
10
11 impl RsaPRG {
12     pub fn new(n: u32, e: u32, w: u32, x: u32) -> RsaPRG {
13         RsaPRG { n, e, w, x }
14     }
15 }
16
17 impl PRGenerator for RsaPRG {
18     fn next(&mut self) -> u32 {
19         fn pow_mod(x: u32, e: u32, m: u32) -> u32 {
20             let mut res = 1;
21             for _ in 0..e {
22                 res = (res * x) % m;
23             }
24             res
25         }
26         let mut word = BitVec::<u32, Msb0>::new();
27         for _ in 0..self.w {
28             self.x = pow_mod(self.x, self.e, self.n);
29             word.push((self.x & 1) != 0);
30         }
31         word.load_be::<u32>() % MOD
32     }
33 }

```

1.9 Алгоритм Блюма-Блюма-Шуба

Описание алгоритма:

1. Сгенерировать два секретных простых числа p и q , сравнимых с 3 по модулю 4. Произведение этих чисел — $n = p \cdot q$ является числом Блюма. Выберем другое случайное число x , взаимно простое с n .
2. Вычислим $x_0 = x^2 \pmod{n}$, которое будет начальным вектором;
3. for $i = 1$ to l do
 - а) $x_i \leftarrow x_{i-1}^2 \pmod{n}$
 - б) $z_i \leftarrow$ последний значащий бит x_i
4. Вернуть z_1, \dots, z_l .

Параметры:

— x — начальное значение (десятичное число), $\gcd(x, 16637) = 1$

Исходный код генератора:

```
1 use crate::prgenerator::{PRGenerator, MOD};
2 use bitvec::prelude::*;
3
4 const N: u32 = 16637;
5
6 pub struct BbsPRG {
7     x: u32,
8 }
9
10 impl BbsPRG {
11     pub fn new(x: u32) -> BbsPRG {
12         BbsPRG {
13             x: ((x % N) * (x % N)) % N,
14         }
15     }
16 }
17
18 impl PRGenerator for BbsPRG {
19     fn next(&mut self) -> u32 {
20         let mut word = BitVec::<u32, Msb0>::new();
21         for _ in 0..std::mem::size_of::<u32>() {
22             self.x = ((self.x % N) * (self.x % N)) % N;
23             word.push((self.x & 1) != 0);
24         }
25         word.load_be::<u32>() % MOD
26     }
27 }
```

2 Преобразование ПСЧ к заданному распределению

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

- Стандартное равномерное с заданным интервалом;
- Треугольное распределение;
- Общее экспоненциальное распределение;
- Нормальное распределение;
- Гамма распределение (для параметра $c=k$);
- Логнормальное распределение;
- Логистическое распределение;
- Биномиальное распределение.

2.1 Стандартное равномерное с заданным интервалом

$$y = p_2 U(x, m) + p_1$$

Исходный код преобразования:

```
1 use crate::prdistribution::PRDistribution;
2
3 pub struct StandardDistribution {}
4
5 impl StandardDistribution {
6     pub fn new() -> Self {
7         StandardDistribution {}
8     }
9 }
10
11 impl PRDistribution for StandardDistribution {
12     fn distribute_numbers(&self, m: u32, xs: &[u32]) -> Vec<f32> {
13         xs.iter().map(|x| *x as f32 / m as f32).collect()
14     }
15 }
```

2.2 Треугольное распределение

$$y = p_1 + p_2(U(x_1, m) + U(x_2, m) - 1)$$

Исходный код преобразования:


```

1 use crate::prdistribution::PRDistribution;
2
3 pub struct TriangleDistribution {
4     p1: f32,
5     p2: f32,
6 }
7
8 impl TriangleDistribution {
9     pub fn new(p1: f32, p2: f32) -> Self {
10         TriangleDistribution { p1, p2 }
11     }
12 }
13
14 impl PRDistribution for TriangleDistribution {
15     fn distribute_numbers(&self, m: u32, xs: &[u32]) -> Vec<f32> {
16         let m = m as f32;
17         let mut res = Vec::new();
18         for i in 0..xs.len() / 2 {
19             let x1 = xs[i * 2] as f32;
20             let x2 = xs[i * 2 + 1] as f32;
21             let val = self.p1 + self.p2 * (x1 / m + x2 / m - 1.0);
22             res.push(val);
23         }
24         res
25     }
26 }

```

2.3 Общее экспоненциальное распределение

$$y = -p_2 \ln(U(x, m)) + a$$

Исходный код преобразования:

```

1 use crate::prdistribution::PRDistribution;
2
3 pub struct ExponentialDistribution {
4     p1: f32,
5     p2: f32,
6 }
7
8 impl ExponentialDistribution {
9     pub fn new(p1: f32, p2: f32) -> Self {

```

```

10     ExponentialDistribution { p1, p2 }
11 }
12 }
13
14 impl PRDistribution for ExponentialDistribution {
15     fn distribute_numbers(&self, m: u32, xs: &[u32]) -> Vec<f32> {
16         xs.iter()
17             .map(|x| self.p1 - self.p2 * (*x as f32 / m as f32).ln())
18             .collect()
19     }
20 }

```

2.4 Нормальное распределение

$$y_1 = p_1 + p_2 \cdot \sqrt{-2 \ln(1 - U(x_1, m))} \cos(2\pi U(x_2, m))$$

$$y_2 = p_1 + p_2 \cdot \sqrt{-2 \ln(1 - U(x_1, m))} \sin(2\pi U(x_2, m))$$

Исходный код преобразования:

```

1 use crate::prdistribution::PRDistribution;
2 use std::f32::consts::PI;
3
4 pub struct NormalDistribution {
5     p1: f32,
6     p2: f32,
7 }
8
9 impl NormalDistribution {
10     pub fn new(p1: f32, p2: f32) -> Self {
11         NormalDistribution { p1, p2 }
12     }
13
14     pub fn norm(&self, m: u32, x1: u32, x2: u32) -> (f32, f32) {
15         let t1 = (-2.0 * (1.0 - x1 as f32 / m as f32).ln()).sqrt();
16         let t2 = 2.0 * PI * (x2 as f32 / m as f32);
17
18         let y1 = self.p1 + self.p2 * t1 * t2.cos();
19         let y2 = self.p1 + self.p2 * t1 * t2.sin();
20
21         (y1, y2)

```

```

22     }
23 }
24
25 impl PRDistribution for NormalDistribution {
26     fn distribute_numbers(&self, m: u32, xs: &[u32]) -> Vec<f32> {
27         let mut res = Vec::new();
28         for i in 0..xs.len() / 2 {
29             let (y1, y2) = self.norm(m, xs[i * 2], xs[i * 2 + 1]);
30             res.push(y1);
31             res.push(y2);
32         }
33         res
34     }
35 }

```

2.5 Гамма распределение (для параметра $p_3 = k$)

$$\begin{aligned}
 y &= \text{gamma}_k(x_1, \dots, x_{p_3}, p_1, p_2, p_3, m) = \\
 &= p_1 - p_2 \cdot \ln\{[1 - U(x_1, m)] \cdot \dots \cdot [1 - U(x_m, m)]\}
 \end{aligned}$$

Исходный код преобразования:

```

1  use crate::prdistribution::PRDistribution;
2
3  pub struct GammaDistribution {
4      p1: f32,
5      p2: f32,
6      p3: f32,
7  }
8
9  impl GammaDistribution {
10     pub fn new(p1: f32, p2: f32, p3: f32) -> Self {
11         GammaDistribution { p1, p2, p3 }
12     }
13 }
14
15 impl PRDistribution for GammaDistribution {
16     fn distribute_numbers(&self, m: u32, xs: &[u32]) -> Vec<f32> {
17         let mut res = Vec::new();
18         // если p3 не целый

```

```

19         if self.p3 - (self.p3 as u32) as f32 > f32::EPSILON {
20             todo!("non-integer p3 parameter");
21         } else {
22             let c = self.p3 as usize;
23             let mut i = 0;
24             while i < xs.len() {
25                 let mut val = 1.0;
26                 for j in 0..c {
27                     val *= 1.0 - (xs[i + j] as f32 / m as f32);
28                 }
29                 res.push(self.p1 - self.p2 * val.ln());
30                 i += c;
31             }
32         }
33         res
34     }
35 }

```

2.6 Логнормальное распределение

$$z_1, z_2 = \text{norm}(x_1, x_2, 0, 1, m)$$

$$y_1 = p_1 \cdot \exp(p_2 - z_1)$$

$$y_2 = p_1 \cdot \exp(p_2 - z_2)$$

Исходный код преобразования:

```

1 use crate::normal::NormalDistribution;
2 use crate::prdistribution::PRDistribution;
3
4 pub struct LognormalDistribution {
5     p1: f32,
6     p2: f32,
7 }
8
9 impl LognormalDistribution {
10     pub fn new(p1: f32, p2: f32) -> Self {
11         LognormalDistribution { p1, p2 }
12     }
13 }
14

```

```

15 impl PRDistribution for LognormalDistribution {
16     fn distribute_numbers(&self, m: u32, xs: &[u32]) -> Vec<f32> {
17         let normal_distr = NormalDistribution::new(0.0, 1.0);
18         let mut res = Vec::new();
19         for i in 0..xs.len() / 2 {
20             let x1 = xs[i * 2];
21             let x2 = xs[i * 2 + 1];
22             let (z1, z2) = normal_distr.norm(m, x1, x2);
23
24             let y1 = self.p1 + (self.p2 - z1).exp();
25             let y2 = self.p1 + (self.p2 - z2).exp();
26
27             res.push(y1);
28             res.push(y2);
29         }
30         res
31     }
32 }

```

2.7 Логистическое распределение

$$y = p_1 + p_2 \cdot \ln \left(\frac{U(x, m)}{1 - U(x, m)} \right)$$

Исходный код преобразования:

```

1 use crate::prdistribution::PRDistribution;
2
3 pub struct LogisticDistribution {
4     p1: f32,
5     p2: f32,
6 }
7
8 impl LogisticDistribution {
9     pub fn new(p1: f32, p2: f32) -> Self {
10         LogisticDistribution { p1, p2 }
11     }
12 }
13
14 impl PRDistribution for LogisticDistribution {
15     fn distribute_numbers(&self, m: u32, xs: &[u32]) -> Vec<f32> {
16         xs.iter()
17             .map(|x| {

```

```

18         let u = *x as f32 / m as f32;
19         self.p1 + self.p2 * (u / (1.0 - u)).ln()
20     })
21     .collect()
22 }
23 }

```

2.8 Биномиальное распределение

$y = \text{binominal}(x, p_1, p_2, m) :$

1. $u = U(x, m);$
2. $s = 0;$
3. $k = 0;$
4. loopstart:
 - a) $s = s + C_{p_2}^k p_1^k (1 - p_1)^{p_2 - k};$
 - б) if $s > u \implies y = k$, завершить;
 - в) if $k < p_2 - 1 \implies k = k + 1$, перейти к loopstart;
 - г) $y = p_2;$

Исходный код преобразования:

```

1 use crate::prdistribution::PRDistribution;
2
3 pub struct BinomialDistribution {
4     p1: f32,
5     p2: f32,
6 }
7
8 impl BinomialDistribution {
9     pub fn new(p1: f32, p2: f32) -> Self {
10         BinomialDistribution { p1, p2 }
11     }
12 }
13
14 impl PRDistribution for BinomialDistribution {
15     fn distribute_numbers(&self, m: u32, xs: &[u32]) -> Vec<f32> {
16         xs.iter()
17             .map(|x| {
18                 let u = *x as f32 / m as f32;
19                 let mut s = 0.0;
20                 let mut k = 0.0;
21

```

```

22         loop {
23             s += num_integer::binomial(self.p2 as u64, k as u64) as
↪ f32
24             * self.p1.powf(k)
25             * (1.0 - self.p1).powf(self.p2 - k);
26         if s > u {
27             return k;
28         }
29
30         if k < self.p2 - 1.0 {
31             k += 1.0;
32             continue;
33         }
34     }
35 })
36 .collect()
37 }
38 }

```

ПРИЛОЖЕНИЕ А

Главный файл main.rs проекта prng

```
1 use std::fs::File;
2 use std::io::Write;
3 use std::rc::Rc;
4
5 use gcd::Gcd;
6 use pbr::ProgressBar;
7
8 mod clp;
9 use clp::{parse_args, Config, GeneratorType};
10
11 mod prgenerator;
12 use prgenerator::{PRGenerator, MOD};
13
14 mod additive;
15 mod bbs;
16 mod fp;
17 mod lfsr;
18 mod linear;
19 mod mersenne;
20 mod nfsr;
21 mod rc4;
22 mod rsa;
23 use additive::AdditivePRG;
24 use bbs::BbsPRG;
25 use fp::FiveParamPRG;
26 use lfsr::LfsrPRG;
27 use linear::LinearPRG;
28 use mersenne::MersennePRG;
29 use nfsr::NfsrPRG;
30 use rc4::Rc4PRG;
31 use rsa::RsaPRG;
32
33 // Генератор запускается только если (1) корректно введены все аргументы,
34 // (2) инициализационный вектор соответствует указанному генератору.
35 // Иначе, будет выведено сообщение с ошибкой.
36 fn main() {
37     match parse_args(std::env::args()) {
38         Ok(conf) => match construct_generator(&conf) {
39             Ok(gen) => {
```



```

40         let numbers = generate_numbers(&conf, gen);
41         write_numbers(&conf, &numbers);
42     }
43     Err(msg) => println!("{}", msg),
44 },
45     Err(msg) => println!("{}", msg),
46 };
47 }
48
49 // Здесь конструируются генераторы на основе введенных в командной строке
50 // параметров. Каждый из генераторов реализует признак (интерфейс)
51 ↳ PRGenerator,
52 // поэтому необходимо хранить ссылку на объект с общим интерфейсом для
53 // использования в других функциях. Если на основе указанных параметров
54 // невозможно сконструировать генератор, то возвращается ошибка.
54 fn construct_generator(conf: &Config) -> Result<Rc<dyn PRGenerator>, String> {
55     match conf.generator {
56         GeneratorType::Lcg => {
57             if conf.init.len() != 4 {
58                 Err("Для линейного конгруэнтного генератора \
59                     инициализационный вектор должен содержать 4 элемента"
60                     .to_string())
61             } else {
62                 Ok(Rc::new(LinearPRG::new(
63                     conf.init[0],
64                     conf.init[1],
65                     conf.init[2],
66                     conf.init[3],
67                 )))
68             }
69         }
70         GeneratorType::Additive => {
71             if conf.init.len() < 3 {
72                 return Err("Для аддитивного генератора инициализационный \
73                     вектор должен содержать m, j и k"
74                     .to_string());
75             }
76             let m = conf.init[0];
77             let k = conf.init[1] as usize;
78             let j = conf.init[2] as usize;
79             let xs = Vec::from(&conf.init[3..]);

```

```

80         if j <= k || k < 1 {
81             return Err("Для аддитивного генератора должно выполняться \
82                 j > k >= 1"
83                 .to_string());
84         }
85         if xs.len() < j {
86             return Err(
87                 "Для аддитивного генератора длина инициализационного \
88                 вектора len(xs) должна быть >= j"
89                 .to_string(),
90             );
91         }
92         Ok(Rc::new(AdditivePRG::new(m, j, k, xs)))
93     }
94     GeneratorType::FiveParam => {
95         let len = conf.init.len();
96         if len < 5 {
97             return Err("Для 5-параметрического генератора необходимо \
98                 указать параметры p,q1,q2,q3,w"
99                 .to_string());
100         }
101         let p = conf.init[0] as usize;
102         let q1 = conf.init[1] as usize;
103         let q2 = conf.init[2] as usize;
104         let q3 = conf.init[3] as usize;
105         let w = conf.init[4];
106         let xs = Vec::from(&conf.init[5..]);
107         if xs.len() < p as _ {
108             return Err(
109                 "Для 5-параметрического генератора необходимо указать \
110                 p коэффициентов X"
111                 .to_string(),
112             );
113         }
114         Ok(Rc::new(FiveParamPRG::new(p, q1, q2, q3, w, xs)))
115     }
116     GeneratorType::Lfsr => {
117         let len = conf.init.len();
118         if len % 2 != 0 {
119             return Err(
120                 "Инициализационный вектор должен содержать чётное \

```

```

121             количество элементов"
122             .to_string(),
123         );
124     }
125     let coeff = Vec::from(&conf.init[..len / 2]);
126     let init = Vec::from(&conf.init[len / 2..]);
127     Ok(Rc::new(LfsrPRG::new(coeff, init)))
128 }
129 GeneratorType::Nfsr => {
130     let len = conf.init.len();
131     if len < 7 {
132         return Err("Инициализационный вектор должен содержать \
133             коэффициенты для трёх РСЛОС, w, x1, x2, x3"
134             .to_string());
135     }
136     let coeffs = &conf.init[..len - 4];
137     if coeffs.len() % 3 != 0 {
138         return Err("Список коэффициентов должен содержать \
139             количество элементов кратное трём"
140             .to_string());
141     }
142     let w = conf.init[len - 4];
143     let x1 = conf.init[len - 3];
144     let x2 = conf.init[len - 2];
145     let x3 = conf.init[len - 1];
146
147     let coeffs1 = Vec::from(&coeffs[..coeffs.len() / 3]);
148     let coeffs2 =
149         Vec::from(&coeffs[coeffs.len() / 3..coeffs.len() * 2 / 3]);
150     let coeffs3 = Vec::from(&coeffs[coeffs.len() * 2 / 3..]);
151
152     fn to_bin(x: u32) -> Vec<u32> {
153         let mut n = x;
154         let mut vec = Vec::new();
155         for _ in 0..std::mem::size_of_val(&x) * 8 {
156             vec.push(n % 2);
157             n /= 2;
158         }
159         vec
160     }
161

```

```

162         let init1 = Vec::from(&to_bin(x1)[..coeffs.len() / 3]);
163         let init2 = Vec::from(&to_bin(x2)[..coeffs.len() / 3]);
164         let init3 = Vec::from(&to_bin(x3)[..coeffs.len() / 3]);
165
166         Ok(Rc::new(NfsrPRG::new(
167             coeffs1, init1, coeffs2, init2, coeffs3, init3, w,
168         )))
169     }
170     GeneratorType::MersenneTwister => {
171         if conf.init.len() != 2 {
172             return Err(
173                 "Инициализационный вектор должен содержать только \
174                     значения модуля и начальный x"
175                 .to_string(),
176             );
177         }
178         let m = conf.init[0];
179         let x = conf.init[1];
180         Ok(Rc::new(MersennePRG::new(m, x)))
181     }
182     GeneratorType::Rc4 => {
183         if conf.init.len() != 256 {
184             return Err("Инициализационный вектор должен содержать 256 \
185                 значений"
186                 .to_string());
187         }
188         Ok(Rc::new(Rc4PRG::new(conf.init.clone())))
189     }
190     GeneratorType::Rsa => {
191         if conf.init.len() != 4 {
192             return Err("Инициализационный вектор должен содержать \
193                 параметры n, e, w, x"
194                 .to_string());
195         }
196         let n = conf.init[0];
197         let e = conf.init[1];
198         let w = conf.init[2];
199         let x = conf.init[3];
200         Ok(Rc::new(RsaPRG::new(n, e, w, x)))
201     }
202     GeneratorType::Bbs => {

```

```

203         if conf.init.len() != 1 {
204             return Err("Инициализационный вектор должен содержать \
205                 только параметр x"
206                 .to_string());
207         }
208         let x = conf.init[0];
209         if x.gcd(16637) != 1 {
210             return Err("x должен быть взаимно простым с
211 ↪ 16637".to_string());
212         }
213         Ok(Rc::new(BbsPRG::new(x)))
214     }
215 }
216
217 // Так как все генераторы имеют общий интерфейс, то просто генерируем заданное
218 // количество чисел по одному с помощью метода .next()
219 fn generate_numbers<T: PRGenerator + ?Sized>(
220     conf: &Config,
221     mut gen: Rc<T>,
222 ) -> Vec<u32> {
223     let mut numbers = Vec::new();
224     let mut pb = ProgressBar::new(conf.n);
225     for _ in 0..conf.n {
226         numbers.push(Rc::get_mut(&mut gen).unwrap().next());
227         pb.message("Генерация чисел: ");
228         pb.inc();
229     }
230     pb.finish_println("Генерация чисел завершена.");
231     println!();
232     numbers
233 }
234
235 fn write_numbers(conf: &Config, numbers: &Vec<u32>) {
236     let mut file = File::create(&conf.file).unwrap();
237     let mut pb = ProgressBar::new(numbers.len().try_into().unwrap());
238     file.write_all(format!("{}", MOD).as_bytes()).unwrap();
239     for i in 0..numbers.len() {
240         let num = numbers[i];
241         if i == numbers.len() - 1 {
242             file.write_all(format!("{}", num).as_bytes()).unwrap();

```

```
243         } else {
244             file.write_all(format!("{}", num).as_bytes()).unwrap();
245         }
246         pb.message("Запись чисел в файл: ");
247         pb.inc();
248     }
249     pb.finish_println("Запись чисел завершена.");
250     println!();
251 }
```

ПРИЛОЖЕНИЕ Б

Файл prgenerator.rs проекта prng

```
1 pub const MOD: u32 = 1024;
2
3 pub trait PRGenerator {
4     fn next(&mut self) -> u32;
5 }
```

ПРИЛОЖЕНИЕ В

Файл парсера аргументов командной строки `clp.rs` проекта `prng`

```
1  #[derive(Debug, Clone, Copy)]
2  pub enum GeneratorType {
3      Lcg,                // линейный конгруэнтный метод;
4      Additive,          // аддитивный метод;
5      FiveParam,         // пятипараметрический метод;
6      Lfsr,              // регистр сдвига с обратной связью (РСЛОС);
7      Nfsr,              // нелинейная комбинация РСЛОС;
8      MersenneTwister,   // вихрь Мерсенна;
9      Rc4,               // RC4;
10     Rsa,               // ГПСЧ на основе RSA;
11     Bbs,              // алгоритм Блюма-Блюма-Шуба;
12 }
13
14 impl GeneratorType {
15     fn parse(name: &str) -> Option<Self> {
16         match name {
17             "lc" => Some(GeneratorType::Lcg),
18             "add" => Some(GeneratorType::Additive),
19             "5p" => Some(GeneratorType::FiveParam),
20             "lfsr" => Some(GeneratorType::Lfsr),
21             "nfsr" => Some(GeneratorType::Nfsr),
22             "mt" => Some(GeneratorType::MersenneTwister),
23             "rc4" => Some(GeneratorType::Rc4),
24             "rsa" => Some(GeneratorType::Rsa),
25             "bbs" => Some(GeneratorType::Bbs),
26             &_ => None,
27         }
28     }
29 }
30
31 #[derive(Debug)]
32 pub struct Config {
33     pub generator: GeneratorType,
34     pub init: Vec<u32>,
35     pub n: u64,
36     pub file: String,
37 }
38
39 fn parse_name(arg: &String) -> (String, String) {
```



```

40     let mut name = Vec::new();
41     let mut i = 0;
42
43     while !arg[i..].starts_with(':') {
44         name.push(arg.as_bytes()[i]);
45         i += 1;
46         if i >= arg.len() {
47             break;
48         }
49     }
50     let name = String::from_utf8(name).unwrap();
51     (name, arg[i..].to_string())
52 }
53
54
55 const HELP_USAGE: &str =
56     "Использование: ./generators /g:<код> /i:<число> [/n:<длина>]
57     ↪  [/f:<имя_файла>] [/h]";
58
59 const HELP_PARAMS: &str =
60     "/i:<число>           - инициализационный вектор генератора.
61     /n:<длина>           - количество генерируемых чисел
62     /f:<полное_имя_файла> - полное имя файла, в который будут выводиться данные.
63     /h                   - Вывод доступных параметров";
64
65 const HELP_GEN: &str =
66     "/g:<код_метода>      - параметр указывает на метод генерации ПСЧ
67     Допустимые значения:
68     * lc   - линейный конгруэнтный метод; (/i:t,a,c,x0)
69     * add  - аддитивный метод; (/i:t,k,j,x0,...,xn), n >= j
70     * 5p   - пятипараметрический метод (/i:p,q1,q2,q3,w,x1,...,xp);
71     * lfsr - регистр сдвига с обратной связью (РСЛОС); (/i:a0,...,ap,x0,...,xp)
72     ↪  (/i:a0,...,ap,b0,...,bp,c0,...,cp,w,x1,x2,x3);
73     * mt   - вихрь Мерсенна (/i:t,x);
74     * rc4   - RC4 (/i:x0,...,x256);
75     * rsa   - ГПСЧ на основе RSA (/i:n,e,w,x);
76     * bbs   - алгоритм Блума-Блума-Шуба (/i:x);";
77
78 const HELP_LC: &str =
79     "/g:lc - линейный конгруэнтный метод; (/i:t,a,c,x0)

```

```

79  Параметры:
80  * m - модуль
81  * a - множитель
82  * c - приращение
83  * x0 - начальное значение (десятичное число);
84
85  const HELP_ADD: &str =
86  "/g:add - аддитивный метод; (/i:m,k,j,x0,...,xn), n >= j
87  Параметры:
88  * m - модуль
89  * k - младший индекс
90  * j - старший индекс
91  * x0,...,xn - n начальных значений (десятичные числа), n >= j";
92
93  const HELP_5P: &str =
94  "/g:5p - пятипараметрический метод (/i:p,q1,q2,q3,w,x1,...,xp);
95  Параметры:
96  * p - количество элементов
97  * q1 - первый индекс
98  * q2 - второй индекс
99  * q3 - третий индекс
100 * w - длина слова в битах
101 * x1,...,xp - начальное значение регистра (последовательность 0 и 1)";
102
103 const HELP_LFSR: &str =
104 "/g:lfsr - регистр сдвига с обратной связью (РСЛОС); (/i:a0,...,ap,x0,...,xp)
105 Параметры:
106 * a0,...,ap - вектор коэффициентов (последовательность 0 и 1)
107 * x0,...,xp - начальное значение регистра (последовательность 0 и 1)";
108
109 const HELP_NFSR: &str =
110 "/g:nfsr - нелинейная комбинация РСЛОС
111 ↪ (/i:a0,...,ap,b0,...,bp,c0,...,cp,w,x1,x2,x3);
112 Параметры:
113 * a0,...,ap - коэффициенты первого РСЛОС (последовательность 0 и 1)
114 * b0,...,bp - коэффициенты второго РСЛОС (последовательность 0 и 1)
115 * c0,...,cp - коэффициенты третьего РСЛОС (последовательность 0 и 1)
116 * w - длина слова в битах
117 * x1 - начальное состояние первого РСЛОС (десятичное число)
118 * x2 - начальное состояние второго РСЛОС (десятичное число)
119 * x3 - начальное состояние третьего РСЛОС (десятичное число)";

```

```

119
120 const HELP_MT: &str = "/g:mt - вихрь Мерсенна (/i:m,x);
121   Параметры:
122   * m - модуль
123   * x - начальное значение (десятичное число);
124
125 const HELP_RC4: &str = "/g:rc4 - RC4 (/i:x0,...,x256);
126   Параметры:
127   * x0,...,x256 - начальные значения (256 десятичных чисел);
128
129 const HELP_RSA: &str = "/g:rsa - ГПСЧ на основе RSA (/i:n,e,w,x);
130   Параметры:
131   * n - модуль  $n=pq$ , где  $p, q$  - простые числа
132   * e - такое число, что  $1 < e < (p-1)(q-1)$ ,  $\gcd(e, (p-1)(q-1)) = 1$ 
133   * x - число из интервала  $[1, n]$ 
134   * w - длина слова в битах";
135
136 const HELP_BBS: &str = "/g:bbs - алгоритм Блума-Блума-Шуба (/i:x);
137   Параметры:
138   * x - начальное значение (десятичное число),  $\gcd(x, 16637) = 1$ ";
139
140 fn help_string(generator: Option<GeneratorType>) -> String {
141   match generator {
142     None => format!("{}", \n {} \n {}", HELP_USAGE, HELP_GEN, HELP_PARAMS),
143     Some(GeneratorType::Lcg) =>
144       format!("{}", \n {} \n {}", HELP_USAGE, HELP_LC, HELP_PARAMS),
145     Some(GeneratorType::Additive) =>
146       format!("{}", \n {} \n {}", HELP_USAGE, HELP_ADD, HELP_PARAMS),
147     Some(GeneratorType::FiveParam) =>
148       format!("{}", \n {} \n {}", HELP_USAGE, HELP_5P, HELP_PARAMS),
149     Some(GeneratorType::Lfsr) =>
150       format!("{}", \n {} \n {}", HELP_USAGE, HELP_LFSR, HELP_PARAMS),
151     Some(GeneratorType::Nfsr) =>
152       format!("{}", \n {} \n {}", HELP_USAGE, HELP_NFSR, HELP_PARAMS),
153     Some(GeneratorType::MersenneTwister) =>
154       format!("{}", \n {} \n {}", HELP_USAGE, HELP_MT, HELP_PARAMS),
155     Some(GeneratorType::Rc4) =>
156       format!("{}", \n {} \n {}", HELP_USAGE, HELP_RC4, HELP_PARAMS),
157     Some(GeneratorType::Rsa) =>
158       format!("{}", \n {} \n {}", HELP_USAGE, HELP_RSA, HELP_PARAMS),
159     Some(GeneratorType::Bbs) =>

```

```

160         format!("{}", n {}, n {}", HELP_USAGE, HELP_BBS, HELP_PARAMS),
161     }
162 }
163
164 pub fn parse_args(args_iter: std::env::Args) -> Result<Config, String> {
165     let mut generator = None;
166     let mut init = None;
167     let mut n: u64 = 10000;
168     let mut file = String::from("rnd.dat");
169
170     // Первый аргумент содержит путь до исполняемого файла, поэтому пропускаем
171     for arg in args_iter.skip(1) {
172         if !arg.starts_with('/') {
173             continue;
174         }
175
176         let rest = arg[1..].to_string();
177         let (name, rest) = parse_name(&rest);
178
179         if name == "h" {
180             return Err(help_string(generator));
181         }
182
183         if !rest.starts_with(':') {
184             return Err(format!(
185                 "Не указано значение параметра '{}'\n\n{}",
186                 name, help_string(generator)
187             ));
188         }
189
190         let value = rest[1..].to_string();
191         match name.as_str() {
192             "g" => match GeneratorType::parse(&value) {
193                 None => {
194                     return Err(format!(
195                         "Недопустимое значение кода генератора\n\n{}",
196                         help_string(generator)
197                     ));
198                 }
199                 gentype => generator = gentype,
200             },

```

```

201     "n" => match value.parse::() {
202         Ok(num) => n = num,
203         Err(_) => {
204             return Err(format!(
205                 "Значение аргумента n должно быть \
206                 неотрицательным числом \n\n{ }",
207                 help_string(generator)
208             ))
209         }
210     },
211     "f" => file = value,
212     "i" => {
213         let mut vec = Vec::new();
214         for num in value.split(',') {
215             match num.parse::() {
216                 Ok(n) => vec.push(n),
217                 Err(_) => {
218                     return Err(format!(
219                         "Значения вектора инициализации должны \
220                         быть неотрицательными числами \n\n{ }",
221                         help_string(generator)
222                     ))
223                 }
224             }
225         }
226         init = Some(vec);
227     }
228     argname => {
229         return Err(format!(
230             "Неизвестный параметр '{ }' \n\n{ }",
231             argname, help_string(generator)
232         ))
233     }
234 }
235 }
236
237 if generator.is_none() {
238     return Err(format!(
239         "Параметр 'g' является обязательным \n\n{ }",
240         help_string(generator)
241     ));

```

```

242     }
243
244     if init.is_none() {
245         return Err(format!(
246             "Параметр 'i' является обязательным \n\n{}",
247             help_string(generator)
248         ));
249     }
250
251     if let (Some(generator), Some(init)) = (generator, init) {
252         Ok(Config {
253             generator,
254             init,
255             n,
256             file,
257         })
258     } else {
259         Err(help_string(generator))
260     }
261 }

```

ПРИЛОЖЕНИЕ Г

Файл Cargo.toml проекта prng

```
1 [package]
2 name = "prng"
3 version = "0.1.0"
4 edition = "2021"
5
6 # See more keys and their definitions at
   ↪ https://doc.rust-lang.org/cargo/reference/manifest.html
7
8 [dependencies]
9 bitvec = "1.0.1"
10 gcd = "2.3.0"
11 pbr = "1.1.1"
```

ПРИЛОЖЕНИЕ Д

Главный файл main.rs проекта rnc

```
1 use std::fs::File;
2 use std::io::Write;
3
4 mod clp;
5 use crate::clp::{parse_args, Config, DistributionType};
6
7 mod prdistribution;
8 use crate::prdistribution::PRDistribution;
9
10 mod binomial;
11 mod exponential;
12 mod gamma;
13 mod logistic;
14 mod lognormal;
15 mod normal;
16 mod standard;
17 mod triangle;
18 use crate::binomial::BinomialDistribution;
19 use crate::exponential::ExponentialDistribution;
20 use crate::gamma::GammaDistribution;
21 use crate::logistic::LogisticDistribution;
22 use crate::lognormal::LognormalDistribution;
23 use crate::normal::NormalDistribution;
24 use crate::standard::StandardDistribution;
25 use crate::triangle::TriangleDistribution;
26
27 fn main() {
28     match parse_args(std::env::args()) {
29         Ok(conf) => match construct_distribution(&conf) {
30             Ok(distr) => {
31                 let numbers = read_numbers(&conf);
32                 let module = numbers[0];
33                 let numbers = Vec::from(&numbers[1..]);
34                 let distributed = distr.distribute_numbers(module, &numbers);
35                 write_numbers(&conf, &distributed);
36             }
37             Err(msg) => println!("{}", msg),
38         },
39         Err(msg) => println!("{}", msg),
40     }
```



```

40     };
41 }
42
43 fn construct_distribution(
44     conf: &Config,
45 ) -> Result<Box<dyn PRDistribution>, String> {
46     match conf.distribution {
47         DistributionType::Standrard => {
48             Ok(Box::new(StandardDistribution::new()))
49         }
50         DistributionType::Triangle => {
51             Ok(Box::new(TriangleDistribution::new(conf.p1, conf.p2)))
52         }
53         DistributionType::Exponential => {
54             Ok(Box::new(ExponentialDistribution::new(conf.p1, conf.p2)))
55         }
56         DistributionType::Normal => {
57             Ok(Box::new(NormalDistribution::new(conf.p1, conf.p2)))
58         }
59         DistributionType::Gamma => {
60             match conf.p3 {
61                 Some(p3) => {
62                     Ok(Box::new(GammaDistribution::new(conf.p1, conf.p2, p3)))
63                 }
64                 None => Err("Параметр p3 обязателен для гамма-распределения"
65                     .to_string()),
66             }
67         }
68         DistributionType::Lognormal => {
69             Ok(Box::new(LognormalDistribution::new(conf.p1, conf.p2)))
70         }
71         DistributionType::Logistic => {
72             Ok(Box::new(LogisticDistribution::new(conf.p1, conf.p2)))
73         }
74         DistributionType::Binomial => {
75             Ok(Box::new(BinomialDistribution::new(conf.p1, conf.p2)))
76         }
77     }
78 }
79
80 fn read_numbers(conf: &Config) -> Vec<u32> {

```

```

81     std::fs::read_to_string(&conf.file)
82         .unwrap()
83         .trim()
84         .split(',')
85         .map(|x| x.parse:::<u32>().unwrap())
86         .collect()
87 }
88
89 fn write_numbers(conf: &Config, numbers: &Vec<f32>) {
90     let file_name = format!("distr-{}.dat", conf.distribution.to_string());
91     let mut file = File::create(file_name).unwrap();
92
93     for i in 0..numbers.len() {
94         let num = numbers[i];
95         if i == numbers.len() - 1 {
96             let _ = file.write_all(format!("{}", num).as_bytes());
97         } else {
98             let _ = file.write_all(format!("{}", num).as_bytes());
99         }
100     }
101 }

```

ПРИЛОЖЕНИЕ Е

Файл `prdistribution.rs` проекта `rnc`

```
1 pub trait PRDistribution {  
2     fn distribute_numbers(&self, m: u32, xs: &[u32]) -> Vec<f32>;  
3 }
```

ПРИЛОЖЕНИЕ Ж

Файл парсера аргументов командной строки `clp.rs` проекта `gnc`

```
1  #[derive(Debug, Clone, Copy)]
2  pub enum DistributionType {
3      Standrard,    // стандартное равномерное с заданным интервалом;
4      Triangle,    // треугольное распределение;
5      Exponential, // общее экспоненциальное распределение;
6      Normal,      // нормальное распределение;
7      Gamma,       // гамма распределение;
8      Lognormal,   // логнормальное распределение;
9      Logistic,    // логистическое распределение;
10     Binomial,    // биномиальное распределение.
11 }
12
13 impl DistributionType {
14     fn parse(name: &str) -> Option<Self> {
15         match name {
16             "st" => Some(DistributionType::Standrard),
17             "tr" => Some(DistributionType::Triangle),
18             "ex" => Some(DistributionType::Exponential),
19             "nr" => Some(DistributionType::Normal),
20             "gm" => Some(DistributionType::Gamma),
21             "ln" => Some(DistributionType::Lognormal),
22             "ls" => Some(DistributionType::Logistic),
23             "bi" => Some(DistributionType::Binomial),
24             &_ => None,
25         }
26     }
27 }
28
29 impl ToString for DistributionType {
30     fn to_string(&self) -> String {
31         match self {
32             DistributionType::Standrard => "st",
33             DistributionType::Triangle => "tr",
34             DistributionType::Exponential => "ex",
35             DistributionType::Normal => "nr",
36             DistributionType::Gamma => "gm",
37             DistributionType::Lognormal => "ln",
38             DistributionType::Logistic => "ls",
39             DistributionType::Binomial => "bi",
```

```

40     }
41     .to_string()
42 }
43 }
44
45 #[derive(Debug)]
46 pub struct Config {
47     pub distribution: DistributionType,
48     pub file: String,
49     pub p1: f32,
50     pub p2: f32,
51     pub p3: Option<f32>,
52 }
53
54 fn parse_name(arg: &String) -> (String, String) {
55     let mut name = Vec::new();
56     let mut i = 0;
57
58     while !arg[i..].starts_with(':') {
59         name.push(arg.as_bytes()[i]);
60         i += 1;
61         if i >= arg.len() {
62             break;
63         }
64     }
65     let name = String::from_utf8(name).unwrap();
66     (name, arg[i..].to_string())
67 }
68
69 const HELP_STR: &str =
70     "Использование: ./rnc /d:<код> /f:<имя_файла> /p1:<число> /p2:<число>
71     ↪  [/p3:<число>] [/h]
72     /d:<код_метода>          - параметр указывает на метод распределения ППСЧ
73     Допустимые значения:
74     * st - стандартное равномерное с заданным интервалом;
75     * tr - треугольное распределение;
76     * ex - общее экспоненциальное распределение;
77     * nr - нормальное распределение;
78     * gm - гамма распределение;
79     * ln - логнормальное распределение;
80     * ls - логистическое распределение;

```

```

80     * bi - биномиальное распределение.
81 /f:<полное_имя_файла> - полное имя файла, из которого будет считываться
    ↪ последовательность.
82 /p1:<число>           - первый параметр.
83 /p2:<число>           - второй параметр.
84 /p3:<число>           - третий параметр (для гамма-распределения).
85 /h                   - вывод доступных параметров";
86
87 pub fn parse_args(args_iter: std::env::Args) -> Result<Config, String> {
88     let mut distribution = None;
89     let mut file = String::from("rnd.dat");
90     let mut p1 = None;
91     let mut p2 = None;
92     let mut p3 = None;
93
94     for arg in args_iter {
95         if !arg.starts_with('/') {
96             continue;
97         }
98
99         let rest = arg[1..].to_string();
100         let (name, rest) = parse_name(&rest);
101
102         if name == "h" {
103             return Err(HELP_STR.to_string());
104         }
105
106         if !rest.starts_with(':') {
107             return Err(format!(
108                 "Не указано значение параметра '{}'\n\n{}",
109                 name, HELP_STR
110             ));
111         }
112
113         let value = rest[1..].to_string();
114         match name.as_str() {
115             "d" => match DistributionType::parse(&value) {
116                 None => {
117                     return Err(format!(
118                         "Недопустимое значение кода распределения\n\n{}",
119                         HELP_STR

```

```

120         ))
121     }
122     distrtype => distribution = distrtype,
123 },
124 "f" => file = value,
125 "p1" => match value.parse::<f32>() {
126     Ok(num) => p1 = Some(num),
127     Err(_) => {
128         return Err("Значение аргумента p1 должно быть \
129             неотрицательным числом"
130             .to_string())
131     }
132 },
133 "p2" => match value.parse::<f32>() {
134     Ok(num) => p2 = Some(num),
135     Err(_) => {
136         return Err("Значение аргумента p2 должно быть \
137             неотрицательным числом"
138             .to_string())
139     }
140 },
141 "p3" => match value.parse::<f32>() {
142     Ok(num) => p3 = Some(num),
143     Err(_) => {
144         return Err("Значение аргумента p3 должно быть \
145             неотрицательным числом"
146             .to_string())
147     }
148 },
149 argname => {
150     return Err(format!(
151         "Неизвестный параметр '{} '\n\n{}",
152         argname, HELP_STR
153     ))
154 }
155 }
156 }
157
158 if distribution.is_none() {
159     return Err(format!(
160         "Параметр 'd' является обязательным\n\n{}",

```

```

161         HELP_STR
162     ));
163 }
164
165 if p1.is_none() {
166     return Err(format!(
167         "Параметр 'p1' является обязательным \n\n{",
168         HELP_STR
169     ));
170 }
171
172 if p2.is_none() {
173     return Err(format!(
174         "Параметр 'p2' является обязательным \n\n{",
175         HELP_STR
176     ));
177 }
178
179 if let (Some(distribution), Some(p1), Some(p2)) = (distribution, p1, p2) {
180     Ok(Config {
181         distribution,
182         file,
183         p1,
184         p2,
185         p3,
186     })
187 } else {
188     Err(HELP_STR.to_string())
189 }
190 }

```


ПРИЛОЖЕНИЕ 3

Файл Cargo.toml проекта rnc

```
1 [package]
2 name = "rnc"
3 version = "0.1.0"
4 edition = "2021"
5
6 # See more keys and their definitions at
  ↪ https://doc.rust-lang.org/cargo/reference/manifest.html
7
8 [dependencies]
9 num-integer = "0.1.45"
```