

МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

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

Кафедра теоретических основ
компьютерной безопасности и
криптографии

Практическая работа

ОТЧЕТ ПО ДИСЦИПЛИНЕ
«ТЕОРИЯ ПСЕВДОСЛУЧАЙНЫХ ГЕНЕРАТОРОВ»

студентки 4 курса 431 группы
специальности 10.05.01 Компьютерная безопасность
факультета компьютерных наук и информационных технологий
Зиминой Ирины Олеговны

Проверил: доцент

подпись, дата

И. И. Слеповичев

Саратов 2023

СОДЕРЖАНИЕ

1	Задание 1. Генератор псевдослучайных чисел	3
1.1	Линейный конгруэнтный метод	3
1.2	Аддитивный метод.....	4
1.3	Пятипараметрический метод	4
1.4	Регистр сдвига с обратной связью (РСЛОС)	5
1.5	Нелинейная комбинация РСЛОС	6
1.6	Вихрь Мерсенна	6
1.7	RC4	7
1.8	ГСПЧ RSA	8
1.9	Алгоритм Блюма-Блюма-Шуба.....	9
2	Задание 2. Преобразование ПСЧ к заданному распределению.....	11
2.1	Стандартное равномерное с заданным интервалом	11
2.2	Треугольное распределение.....	12
2.3	Общее экспоненциальное распределение	12
2.4	Нормальное распределение	12
2.5	Гамма распределение (для параметра $c = k$)	13
2.6	Логнормальное распределение	13
2.7	Логистическое распределение	14
2.8	Биномиальное распределение	14
	Приложение А. rng.cpp	16
	Приложение Б. rnc.cpp	27

1 Задание 1. Генератор псевдослучайных чисел

Создайте программу для генерации псевдослучайных величин следующими алгоритмами:

- Линейный конгруэнтный метод;
- Аддитивный метод;
- Пятипараметрический метод;
- Регистр сдвига с обратной связью (РСЛОС);
- Нелинейная комбинация РСЛОС;
- Вихрь Мерсенна;
- RC4;
- ГПСЧ на основе RSA;
- Алгоритм Блюма-Блюма-Шуба;

Информация о допустимых параметрах:

```
PS C:\Users\Ira\Desktop\TPRG> .\prng.exe /h
Введена команда с /h. Допустимые параметры:

/g:<код_метода> - параметр указывает на метод генерации ПСЧ, при этом код_метода может быть одним из следующих:

lc - линейный конгруэнтный метод (Вход: модуль, множитель, приращение, начальное значение)
add - аддитивный метод(Вход: модуль, младший индекс, старший индекс, последовательность начальных значений)
5p - пятипараметрический метод(Вход: p, q1, q2, q3, w)
lfsr - регистр сдвига с обратной связью(РСЛОС) (Вход: двоичное представление вектора коэффициентов, начальное значение регистра)
nfsr - нелинейная комбинация РСЛОС(Вход: двоичное представление векторов коэффициентов для R1, R2, R3)
mt - вихрь Мерсенна(Вход: модуль, начальное значение x)
rc4 - RC4(Вход: 256 начальных значений)
rsa - ГПСЧ на основе RSA(Вход: модуль n, число e, начальное значение x; e удовлетворяет условиям : 1 < e < (p - 1)(q - 1), НОД(e, (p - 1)(q - 1)) = 1, где p * q = n.x из интервала[1, n])
bbs - алгоритм Блюма - Шуба(Вход: Начальное значение x(взаимно простое с n));

/i:<число> - инициализационный вектор генератора.

/n:<длина> - количество генерируемых чисел. Если параметр не указан, - генерируется 10000 чисел.

/f:<полное_имя_файла> - полное имя файла, в который будут выводиться данные. Если параметр не указан, данные должны записываться в файл с именем rnd.dat.

/h - информация о допустимых параметрах командной строки программы.
```

1.1 Линейный конгруэнтный метод

Линейная конгруэнтная последовательность – последовательность ПСЧ, получается по формуле:

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

Параметры:

- $m > 0$, модуль;
- $0 \leq a \leq m$, множитель;
- $0 \leq c \leq m$, приращение (инкремент);
- $0 \leq X_0 \leq m$, начальное значение.

Результат работы программы записан в файл lc.txt:

```
prng.cpp  lc.txt x
lc.txt
1 169,266,509,126,325,933,747,763,304,103,506,770,123,701,256,807,128,691,153,935,219,163,119,696,910,610,193,319,479,785,465,727,324,201,280,885,674,139,
466,810,377,996,609,994,619,71,916,263,270,216,856,132,896,542,658,881,176,847,171,832,772,841,508,360,889,205,612,242,924,527,975,663,178,85,423,493,502,
131,471,224,934,112,325,684,490,805,528,458,590,320,162,754,133,530,296,968,634,71,823,401,298,699,832,831,523,926,742,441,683,959,388,176,554,267,333,
455,175,153,735,177,637,393,127,857,357,98,667,700,90,213,348,103,496,375,137,369,708,119,979,419,195,251,436,219,187,905,337,860,606,403,122,686,490,121,
144,598,784,982,727,182,51,877,244,573,47,897,400,239,358,606,379,410,117,613,125,7,795,577,203,70,692,778,294,543,658,539,985,917,805,162,554,91,3,497,
399,508,819,655,749,446,4,580,452,375,147,837,694,451,527,33,345,523,467,976,971,849,196,651,251,226,783,675,847,220,119,203,964,512,150,85,999,907,586,
506,253,392,962,263,577,438,9,67,8,419,937,527,492,111,66,301,665,627,774,597,208,910,835,811,709,451,361,171,373,5,370,674,598,232,280,284,758,265,919,
947,629,574,872,298,250,460,346,948,712,700,666,627,432,405,278,470,165,593,887,874,571,682,676,212,783,426,517,262,270,558,47,62,262,211,876,630,681,710,
734,529,565,428,756,441,859,288,338,25,126,183,783,416,122,276,939,9,360,913,991,253,192,920,737,778,870,884,742,841,767,84,833,313,733,846,596,760,880,
469,92,862,78,804,938,351,894,34,287,289,79,960,63,145,222,143,232,221,602,340,884,366,469,282,509,619,271,30,790,997,858,982,385,991,121,510,502,355,225,
392,537,677,661,813,792,162,36,643,115,281,84,374,547,262,11,834,103,955,141,608,276,470,849,904,747,680,910,117,247,221,436,478,912,849,445,981,209,2,
355,591,296,108,994,253,167,417,947,605,862,230,421,327,250,543,741,934,971,248,353,777,704,363,962,253,109,452,751,519,321,269,592,569,809,401,122,369,
874,982,951,937,219,553,809,836,103,271,758,748,470,58,413,332,738,26,858,723,661,46,13,316,972,683,65,560,731,763,812,84,408,727,817,147,378,928,908,620,
344,689,598,66,418,312,167,241,617,647,12,658,432,805,362,596,618,730,138,452,834,987,307,528,140,183,959,673,80,52,584,999,883,801,865,303,396,825,460,
297,733,280,577,662,837,235,612,984,199,583,867,542,575,413,190,588,623,217,622,569,550,677,178,261,680,451,278,777,387,747,822,987,448,888,322,528,47,
321,987,155,984,824,284,382,967,434,371,415,532,96,242,724,485,448,937,610,960,97,325,850,352,777,45,555,892,620,779,670,500,23,184,832,289,362,396,576,
203,339,884,708,661,212,876,288,489,780,294,860,347,679,178,578,369,591,545,365,952,102,813,333,396,493,736,353,625,159,46,62,604,330,806,69,350,928,483,
33,428,991,453,234,446,463,419,996,209,910,493,619,779,11,692,954,551,501,507,111,901,439,693,36,677,295,909,293,919,371,215,490,570,516,155,432,122,769,
958,107,403,146,471,131,71,140,808,660,188,505,912,542,223,827,274,963,478,312,650,720,729,431,698,524,160,412,625,984,717,104,755,758,890,547,462,53,360,
830,523,267,934,371,49,628,598,926,58,130,823,152,969,741,983,258,607,403,854,567,560,755,475,380,34,404,937,620,354,83,584,398,967,927,390,513,575,330,
723,602,364,669,7,419,278,719,422,551,736,519,487,131,564,86,906,703,422,986,790,421,444,898,83,208,26,409,351,176,471,873,347,196,700,466,585,833,679,
637,344,840,425,234,94,877,503,297,992,77,521,770,958,766,284,216,31,462,854,318,303,797,626,808,118,955,976,746,304,880,752,602,447,63,994,678,754,333,
645,823,694,202,197,75,496,951,478,453,9,902,146,447,346,430,263,812,450,385,298,133,813,733,480,619,261,563,804,738,309,294,235,719,163,754,792,411,366,
601,965,927,0,824,435,136,134,37,936,678,661,471,673,232,670,901,898,459,580,511,57,565,218,246,856,874,171,598,550,687,573,247,11,1000,966,927,659,705,
```

1.2 Аддитивный метод

Последовательность получается по формуле:

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

Параметры:

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

Результат работы программы записан в файле add.txt:

```
prng.cpp  add.txt x
add.txt
1 486,155,276,894,441,226,762,234,762,98,458,399,445,765,223,879,571,243,691,679,728,770,485,284,718,942,699,751,790,616,541,103,181,624,343,217,858,568,
208,956,426,679,238,836,493,912,148,252,456,213,925,761,177,158,167,460,985,551,714,999,502,626,388,566,95,428,811,899,634,153,448,723,119,210,853,847,2,
245,829,465,864,358,782,510,677,842,118,922,954,928,304,626,401,588,339,959,150,89,366,372,483,296,368,19,306,987,796,542,463,365,984,169,75,772,269,929,
820,587,80,752,348,520,798,191,812,152,334,194,837,346,654,149,529,990,828,914,463,416,292,287,795,476,366,615,894,970,676,446,123,831,816,172,217,117,
138,136,736,299,711,644,325,604,761,103,849,282,9,379,45,149,996,163,806,705,128,16,647,967,183,476,321,753,113,973,56,205,722,9,938,119,86,133,718,742,
258,685,825,175,987,818,342,29,829,266,152,389,272,901,126,646,363,881,75,905,487,731,388,983,275,88,303,530,453,386,708,471,141,169,300,663,782,942,245,
208,594,994,910,70,772,449,13,793,653,752,422,219,164,261,913,645,559,281,659,860,860,413,69,433,315,144,822,327,119,87,724,304,52,46,537,316,329,112,137,
129,697,305,430,212,307,347,229,904,74,460,406,986,503,93,593,835,119,772,839,145,530,216,307,449,961,888,393,796,996,563,718,506,652,629,491,50,230,193,
547,136,296,562,139,136,157,448,884,976,281,226,528,744,668,274,234,629,699,76,22,130,498,154,722,90,892,356,972,385,288,826,785,453,592,117,342,283,771,
276,789,252,256,319,903,732,679,936,276,576,273,794,716,868,233,48,811,862,367,569,58,312,203,266,391,583,912,476,359,818,382,754,480,631,822,577,291,247,
968,561,626,500,327,825,172,152,151,144,852,847,564,459,592,300,314,597,418,635,393,655,554,202,505,54,632,108,115,334,136,684,819,530,90,570,742,69,627,
210,671,953,212,71,938,142,180,709,882,360,222,186,702,832,886,804,267,266,485,995,537,382,989,682,870,55,666,44,845,63,614,773,273,442,203,812,824,3,701,
358,870,527,368,983,380,8,335,111,204,207,341,207,760,814,197,853,760,733,430,836,959,982,280,88,622,90,269,185,359,406,909,363,671,256,63,7,162,55,277,
955,980,39,262,400,671,583,736,130,200,828,508,655,70,8,98,611,303,563,620,256,570,430,69,267,860,922,795,707,797,938,20,542,495,299,673,11,322,559,240,
```

1.3 Пятипараметрический метод

Частный случай РСЛОС, использует характеристический многочлен из 5-ти членов и позволяет генерировать последовательности w -битовых двоичных чисел в соответствии со следующей рекуррентной формулой:

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

Параметры:

- p ;
- q_1 ;
- q_2 ;
- q_3 ;
- w .

Результат работы программы записан в файле 5p.txt:

```

5p.txt
1 969,484,978,489,980,490,245,358,179,590,530,765,382,426,448,724,597,799,134,67,769,620,45,22,
747,609,304,888,179,89,780,125,798,399,935,968,984,727,98,785,127,799,635,317,158,815,407,
203,101,551,275,638,319,159,79,775,387,929,464,232,116,293,382,191,331,401,701,851,926,463,
732,866,168,584,292,882,441,956,713,857,428,714,92,782,126,799,635,818,909,955,713,356,914,
957,213,342,171,586,793,131,801,135,803,902,451,726,598,299,885,678,574,787,629,815,908,954,
712,856,928,464,968,219,845,658,564,517,994,232,351,411,441,721,95,548,274,137,68,534,502,
751,375,923,461,966,983,226,613,542,771,621,546,8,4,2,236,118,59,29,515,993,997,233,617,308,
890,945,207,839,920,460,730,365,418,945,973,221,611,40,756,113,56,764,617,308,890,445,458,
965,482,476,974,987,994,732,866,933,967,218,344,172,321,396,433,952,211,841,420,946,473,236,
118,59,530,765,382,927,198,599,34,252,626,813,907,453,727,599,800,135,568,19,245,122,561,516,
758,879,940,205,603,802,401,200,836,653,562,781,390,195,333,166,83,277,374,422,947,974,487,
479,975,487,243,622,46,759,114,57,264,367,919,194,597,534,267,869,434,217,344,908,954,977,
223,612,541,5,2,236,118,559,515,993,997,734,602,801,636,318,659,565,17,744,107,53,262,867,
168,319,660,565,17,244,357,679,840,655,327,163,817,909,454,462,731,100,50,260,365,918,694,82,

```

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 вычислительный 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:

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

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, a_2, \dots, a_{p-1} . Он вычисляется по формуле выше.
3. Содержимое каждого i -го бита перемещается в $(i + 1)$ -й, $0 \leq i \leq p - 1$.

4. В ячейку 0 записывается новое содержимое, вычисленное на шаге 2.

Параметры:

- двоичное представление вектора коэффициентов;
- начальное значение регистра.

Результат работы программы записан в файле lfsr.txt:

```
prng.cpp  lfsr.txt
lfsr.txt
1  429,214,107,53,394,64,900,818,777,889,945,973,987,361,681,340,170,85,911,823,779,389,562,148,
442,589,162,81,40,20,378,557,146,941,470,735,868,934,334,667,701,718,727,864,800,267,501,618,
176,456,596,666,333,534,635,317,158,579,790,895,815,407,203,101,50,893,446,223,479,239,988,
862,431,83,542,771,886,811,273,637,318,527,131,433,84,910,322,529,632,316,158,447,223,111,
556,278,6,871,435,217,108,554,144,940,470,603,301,651,193,965,350,675,337,36,386,561,148,574,
154,445,590,663,832,283,141,438,219,610,172,954,344,672,336,668,201,468,734,735,367,684,209,
973,987,861,430,82,409,705,220,610,172,454,94,415,708,854,294,515,257,997,866,300,518,759,
247,123,429,582,158,79,540,770,252,126,563,649,692,846,791,395,65,32,516,125,931,966,483,241,
989,995,497,616,676,838,419,577,156,78,539,269,2,1,501,250,493,747,373,186,93,547,273,136,
436,218,109,54,27,514,257,128,64,400,200,100,50,393,697,348,174,455,728,364,182,91,914,457,
228,114,57,529,264,132,66,33,16,8,372,554,277,138,69,903,952,476,238,119,560,280,140,438,587,
```

1.5 Нелинейная комбинация РСЛОС

Последовательность получается из нелинейных комбинаций трех РСЛОС следующим образом:

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

Параметры:

- двоичное представление вектора коэффициентов R_1, R_2, R_3 .

Результат работы программы записан в файле nfsr.txt:

```
prng.cpp  nfsr.txt
nfsr.txt
1  200,548,126,78,33,986,25,255,1018,486,403,379,415,431,990,11,152,739,986,276,279,103,472,487,
128,542,614,399,960,938,232,713,527,1011,246,408,84,761,970,895,641,711,718,127,17,913,470,
488,540,865,769,802,304,508,376,903,854,240,503,591,664,69,815,637,247,864,20,490,389,705,
617,537,917,544,533,210,126,963,31,787,648,996,270,63,988,505,580,862,487,655,502,6,462,920,
474,198,69,793,742,99,268,135,941,353,993,489,574,130,915,479,269,610,447,414,993,671,960,88,
171,574,128,452,81,562,135,220,584,216,574,127,14,163,778,664,255,273,486,474,875,958,939,
988,25,859,739,394,536,278,102,972,367,948,14,550,271,965,898,229,73,519,979,574,404,766,763,
903,765,768,64,614,252,185,17,342,57,540,466,769,815,240,508,378,902,10,100,383,363,408,5,
813,637,687,1008,87,166,909,899,73,533,410,609,958,608,62,659,575,535,616,439,654,287,836,
601,596,734,871,970,1015,514,398,24,450,164,577,568,930,25,357,131,140,355,993,495,536,296,
979,501,789,612,183,702,494,703,448,510,187,565,910,388,97,574,199,120,192,456,828,254,30,
```

1.6 Вихрь Мерсенна

Метод Вихрь Мерсенна позволяет генерировать последовательность двоичных псевдослучайных целых w-битовых чисел в соответствии со следующей рекуррентной формулой:

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

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

X_n – w -битовое двоичное целое число;

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

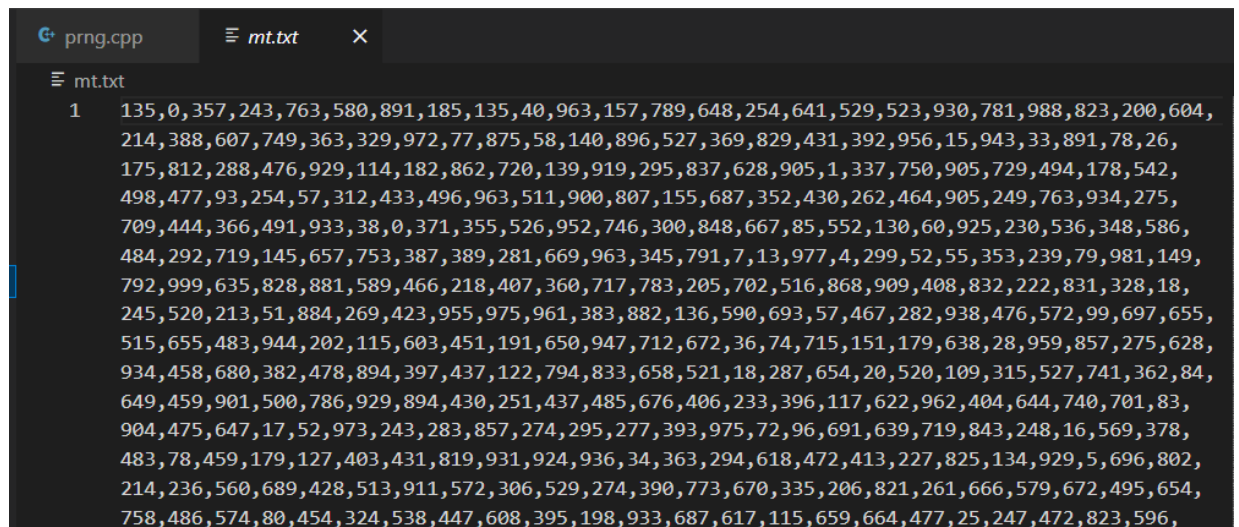
A – матрица размера $w \times w$, состоящая из нулей и единиц, определенная посредством a ;

XA – произведение, при вычислении которого сначала выполняют операцию $X \gg 1$ (сдвига битов на одну позицию вправо), если последний бит X равен 0, а затем, когда последний бит $X = 1$, вычисляют $XA = (X \gg 1) \oplus a$.

Параметры:

- модуль;
- начальное значение x .

Результат работы программы записан в файле mt.txt:



```
prng.cpp  mt.txt  X
mt.txt
1  135,0,357,243,763,580,891,185,135,40,963,157,789,648,254,641,529,523,930,781,988,823,200,604,
  214,388,607,749,363,329,972,77,875,58,140,896,527,369,829,431,392,956,15,943,33,891,78,26,
  175,812,288,476,929,114,182,862,720,139,919,295,837,628,905,1,337,750,905,729,494,178,542,
  498,477,93,254,57,312,433,496,963,511,900,807,155,687,352,430,262,464,905,249,763,934,275,
  709,444,366,491,933,38,0,371,355,526,952,746,300,848,667,85,552,130,60,925,230,536,348,586,
  484,292,719,145,657,753,387,389,281,669,963,345,791,7,13,977,4,299,52,55,353,239,79,981,149,
  792,999,635,828,881,589,466,218,407,360,717,783,205,702,516,868,909,408,832,222,831,328,18,
  245,520,213,51,884,269,423,955,975,961,383,882,136,590,693,57,467,282,938,476,572,99,697,655,
  515,655,483,944,202,115,603,451,191,650,947,712,672,36,74,715,151,179,638,28,959,857,275,628,
  934,458,680,382,478,894,397,437,122,794,833,658,521,18,287,654,20,520,109,315,527,741,362,84,
  649,459,901,500,786,929,894,430,251,437,485,676,406,233,396,117,622,962,404,644,740,701,83,
  904,475,647,17,52,973,243,283,857,274,295,277,393,975,72,96,691,639,719,843,248,16,569,378,
  483,78,459,179,127,403,431,819,931,924,936,34,363,294,618,472,413,227,825,134,929,5,696,802,
  214,236,560,689,428,513,911,572,306,529,274,390,773,670,335,206,821,261,666,579,672,495,654,
  758,486,574,80,454,324,538,447,608,395,198,933,687,617,115,659,664,477,25,247,472,823,596,
```

1.7 RC4

Последовательность ПСЧ получается так:

1. Инициализация $S_j, i = 0, 1, \dots, 255$.

a. *for* $i = 0$ to 255: $S_i = i$;

b. $j = 0$;

c. *for* $i = 0$ to 255: $j = (j + S_i + K_i) \bmod 256$; $Swap(S_i, S_j)$;

2. $i = 0, j = 0$.

3. Итерация алгоритма:

- a. $i = (i + 1) \bmod 256$;
- b. $j = (j + S_i) \bmod 256$;
- c. $Swap(S_i, S_j)$;
- d. $t = (S_i + S_j) \bmod 256$;
- e. $K = S_t$;

Параметры:

- 256 начальных значений S_i .

Результат работы программы записан в файле rc4.txt:

```

rc4.txt
1 138,66,85,167,22,245,254,26,162,140,87,219,68,83,101,120,17,133,177,46,52,211,248,104,104,
  227,60,202,56,124,86,112,112,237,217,235,120,208,235,192,8,92,163,53,10,22,146,248,146,224,
  195,7,225,128,120,87,31,253,56,60,134,213,157,41,2,216,100,73,192,204,131,88,139,246,145,87,
  249,179,224,124,228,107,233,96,142,30,227,15,241,187,121,181,207,228,158,91,132,12,160,44,32,
  190,36,164,121,31,42,181,29,129,78,205,241,238,247,187,252,201,206,104,52,228,183,53,103,251,
  173,219,81,118,16,243,217,9,27,224,111,45,167,224,227,93,150,29,208,3,97,198,174,233,229,145,
  197,225,143,33,109,66,109,181,200,230,189,184,253,156,146,208,47,133,192,146,26,93,182,251,
  144,203,11,162,26,124,195,232,158,57,170,244,235,158,122,231,19,122,237,103,10,155,207,186,
  87,86,246,245,190,90,170,133,160,238,229,68,15,150,106,32,125,152,30,64,127,237,234,94,37,
  202,227,82,95,254,134,122,205,124,4,72,68,107,20,214,182,128,120,133,113,57,244,214,137,193,
  35,166,205,20,217,167,178,185,221,186,230,136,100,83,188,221,157,125,42,5,223,77,204,118,249,
  187,161,164,79,30,184,24,222,10,230,59,21,78,244,151,93,113,13,4,207,87,148,57,148,182,23,
  212,1,57,143,241,39,154,99,120,179,125,122,68,75,75,188,19,70,131,38,20,231,233,199,21,30,48,
  96,77,230,42,103,160,238,94,56,180,78,117,247,0,13,93,58,234,168,38,99,70,23,0,111,170,100,

```

1.8 ГСПЧ RSA

Описание алгоритма:

1. Сгенерировать два секретных простых числа p и q , а также $n = pq$ и $f = (p - 1)(q - 1)$. Выбрать случайное целое число e , $1 < e < f$, такое что $\text{НОД}(e, f) = 1$.
2. Выбрать случайное целое x_0 – начальный вектор из интервала $[1, n - 1]$.
3. *For* $i = 1$ *to* l *do*
 - a. $x_i \leftarrow x_{i-1}^e \bmod n$;
 - b. $z_i \leftarrow$ последний значащий бит x_i
4. Вернуть z_1, z_2, \dots, z_l .

Параметры:

- модуль n ;
- число e ;
- начальное значение x .


```
1 532,186,978,243,346,403,786,576,946,508,918,839,576,994,821,645,590,992,635,314,341,438,965,
19,95,673,402,830,833,18,293,682,974,889,120,231,111,53,200,8,935,32,53,774,972,285,869,271,
751,545,828,986,953,464,744,935,384,13,31,449,737,212,270,175,617,160,623,831,936,257,913,97,
568,330,714,775,824,757,199,499,329,980,65,443,389,690,408,745,274,726,986,829,887,672,821,
151,948,3,11,917,834,388,879,928,919,561,648,192,227,713,783,161,334,562,561,447,606,795,756,
88,847,155,358,129,98,758,464,999,97,331,519,548,528,805,824,807,641,657,262,830,959,152,492,
830,870,861,105,482,744,426,635,371,108,249,958,791,846,896,542,105,110,59,8,619,358,390,539,
637,738,113,425,885,69,20,453,863,871,413,732,704,448,869,290,271,929,412,967,228,120,33,560,
257,264,206,996,911,590,906,337,27,428,967,13,665,606,232,603,13,291,415,486,298,79,694,888,
495,284,31,362,228,257,838,375,400,809,616,851,314,643,305,523,746,516,55,472,236,192,332,
917,285,749,449,526,743,929,752,349,515,371,673,36,979,27,325,29,923,904,828,973,592,647,397,
406,274,408,310,870,110,188,529,719,756,625,495,854,865,676,618,422,968,904,663,303,961,941,
576,624,601,924,498,221,245,713,729,206,413,443,307,84,618,905,290,199,777,198,981,230,910,
376,3,239,414,749,708,676,516,546,674,817,697,946,333,328,639,157,308,17,88,176,230,996,474,
853,815,155,23,231,601,278,225,501,179,111,528,951,152,10,210,580,10,12,112,623,202,451,231,
```

Описание алгоритма:

- Для получения i -го бита b_i при известных p и q достаточно воспользоваться формулой:

Параметры:

- Результат работы программы записан в файле bbs.txt:

```
prng.cpp  bbc.txt  X
bbc.txt
1  817,587,128,211,423,846,691,335,625,250,454,909,817,587,128,211,423,846,691,335,625,250,454,
909,817,587,128,211,423,846,691,335,625,250,454,909,817,587,128,211,423,846,691,335,625,250,
454,909,817,587,128,211,423,846,691,335,625,250,454,909,817,587,128,211,423,846,691,335,625,
250,454,909,817,587,128,211,423,846,691,335,625,250,454,909,817,587,128,211,423,846,691,335,
625,250,454,909,817,587,128,211,423,846,691,335,625,250,454,909,817,587,128,211,423,846,691,
335,625,250,454,909,817,587,128,211,423,846,691,335,625,250,454,909,817,587,128,211,423,846,
691,335,625,250,454,909,817,587,128,211,423,846,691,335,625,250,454,909,817,587,128,211,423,
846,691,335,625,250,454,909,817,587,128,211,423,846,691,335,625,250,454,909,817,587,128,211,
423,846,691,335,625,250,454,909,817,587,128,211,423,846,691,335,625,250,454,909,817,587,128,
211,423,846,691,335,625,250,454,909,817,587,128,211,423,846,691,335,625,250,454,909,817,587,
128,211,423,846,691,335,625,250,454,909,817,587,128,211,423,846,691,335,625,250,454,909,817,
587,128,211,423,846,691,335,625,250,454,909,817,587,128,211,423,846,691,335,625,250,454,909,
817,587,128,211,423,846,691,335,625,250,454,909,817,587,128,211,423,846,691,335,625,250,454,909,
```

2 Задание 2. Преобразование ПСЧ к заданному распределению

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

- a. Стандартное равномерное с заданным интервалом;
- b. Треугольное распределение;
- c. Общее экспоненциальное распределение;
- d. Нормальное распределение;
- e. Гамма распределение (для параметра $c = k$);
- f. Логнормальное распределение;
- g. Логистическое распределение;
- h. Биномиальное распределение

Для теста был выбран файл `mt.txt`, полученный в результате работы генератора на основе вихря Марсенна с параметрами $m = 10001, x = 8191$:

Нормирование:

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

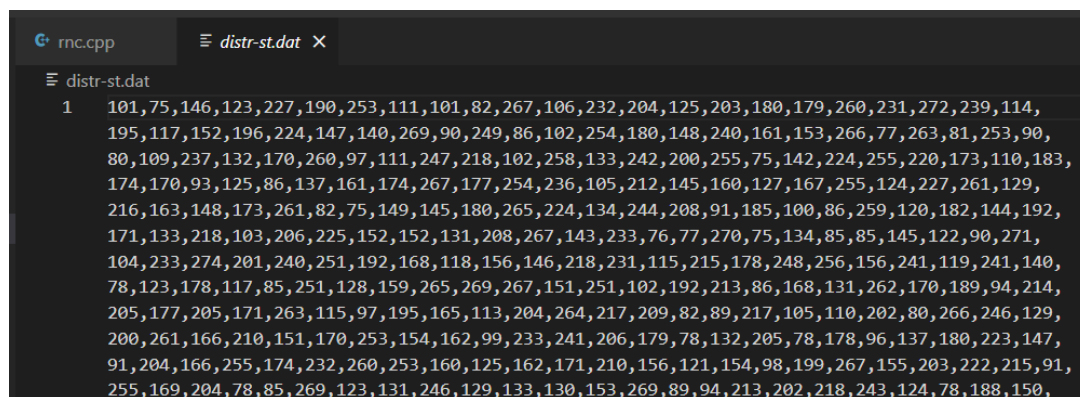
$$U = \frac{X}{m}.$$

2.1 Стандартное равномерное с заданным интервалом

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

$$Y = bU + a.$$

Результат работы программы записан в файле `distr-st.dat`:



```
1 101,75,146,123,227,190,253,111,101,82,267,106,232,204,125,203,180,179,260,231,272,239,114,
195,117,152,196,224,147,140,269,90,249,86,102,254,180,148,240,161,153,266,77,263,81,253,90,
80,109,237,132,170,260,97,111,247,218,102,258,133,242,200,255,75,142,224,255,220,173,110,183,
174,170,93,125,86,137,161,174,267,177,254,236,105,212,145,160,127,167,255,124,227,261,129,
216,163,148,173,261,82,75,149,145,180,265,224,134,244,208,91,185,100,86,259,120,182,144,192,
171,133,218,103,206,225,152,152,131,208,267,143,233,76,77,270,75,134,85,85,145,122,90,271,
104,233,274,201,240,251,192,168,118,156,146,218,231,115,215,178,248,256,156,241,119,241,140,
78,123,178,117,85,251,128,159,265,269,267,151,251,102,192,213,86,168,131,262,170,189,94,214,
205,177,205,171,263,115,97,195,165,113,204,264,217,209,82,89,217,105,110,202,80,266,246,129,
200,261,166,210,151,170,253,154,162,99,233,241,206,179,78,132,205,78,178,96,137,180,223,147,
91,204,166,255,174,232,260,253,160,125,162,171,210,156,121,154,98,199,267,155,203,222,215,91,
255,169,204,78,85,269,123,131,246,129,133,130,153,269,89,94,213,202,218,243,124,78,188,150,
```

2.2 Треугольное распределение

Если стандартные случайные числа U_1 и U_2 независимо получены методом генерации стандартного равномерного числа, то случайное число Y , подчиняющееся треугольному распределению, определяется по формуле:

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

Результат работы программы записан в файле `distr-tr.dat`:

```
nc.cpp x distr-tr.dat x
distr-tr.dat
1 135,1,357,243,763,580,891,185,135,40,963,157,789,648,254,641,529,523,930,781,988,823,200,604,
214,388,607,749,363,329,972,77,875,58,140,896,527,369,829,431,392,956,15,943,33,891,78,26,
175,812,288,476,929,114,182,862,720,139,919,295,837,628,905,1,337,750,905,729,494,178,542,
498,477,93,254,57,312,433,496,963,511,900,807,155,687,352,430,262,464,905,249,763,934,275,
709,444,366,491,933,38,1,371,355,526,952,746,300,848,667,85,552,130,60,925,230,536,348,586,
484,292,719,145,657,753,387,389,281,669,963,345,791,7,13,977,4,299,52,55,353,239,79,981,149,
792,999,635,828,881,589,466,218,407,360,717,783,205,702,516,868,909,408,832,222,831,328,18,
245,520,213,51,884,269,423,955,975,961,383,882,136,590,693,57,467,282,938,476,572,99,697,655,
515,655,483,944,202,115,603,451,191,650,947,712,672,36,74,715,151,179,638,28,959,857,275,628,
934,458,680,382,478,894,397,437,122,794,833,658,521,18,287,654,20,520,109,315,527,741,362,84,
649,459,901,500,786,929,894,430,251,437,485,676,406,233,396,117,622,962,404,644,740,701,83,
904,475,647,17,52,973,243,283,857,274,295,277,393,975,72,96,691,639,719,843,248,16,569,378,
```

2.3 Общее экспоненциальное распределение

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

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

Результат работы программы записан в файле `distr-ex.dat`:

```
nc.cpp x distr-ex.dat x
distr-ex.dat
1 8,7,11,9,16,13,17,9,8,7,18,8,16,14,10,14,13,13,18,16,18,16,9,14,9,11,14,15,11,10,18,7,17,7,8,
17,13,11,16,12,11,18,7,18,7,17,7,7,9,16,10,12,18,8,9,17,15,8,18,10,17,14,17,7,11,15,17,15,12,
9,13,12,12,8,10,7,10,12,12,18,13,17,16,8,15,11,12,10,12,17,9,16,18,10,15,12,11,12,18,7,7,11,
11,13,18,15,10,17,14,8,13,8,7,18,9,13,11,14,12,10,15,8,14,16,11,11,10,15,18,11,16,7,7,18,7,
10,7,7,11,9,7,18,8,16,18,14,16,17,14,12,9,11,11,15,16,9,15,13,17,17,11,16,9,16,10,7,9,13,9,7,
17,10,12,18,18,11,17,8,14,15,7,12,10,18,12,13,8,15,14,13,14,12,18,9,8,14,12,9,14,18,15,15,
7,7,15,8,9,14,7,18,17,10,14,18,12,15,11,12,17,11,12,8,16,16,14,13,7,10,14,7,13,8,10,13,15,11,
8,14,12,17,12,16,18,17,12,10,12,12,15,11,9,11,8,14,18,11,14,15,15,7,17,12,14,7,7,18,9,10,17,
10,10,10,11,18,7,8,15,14,15,17,9,7,13,11,12,7,12,9,8,11,12,16,18,18,18,7,11,10,14,12,11,9,16,
8,18,7,15,16,9,9,13,15,12,13,17,13,10,13,10,11,16,15,11,9,16,10,14,13,15,12,14,16,12,13,7,12,
10,13,12,14,11,9,18,15,14,8,14,14,12,7,9,12,16,14,17,11,18,10,17,16,9,17,7,8,15,17,10,17,17,
13,9,17,7,17,11,11,9,13,9,15,12,16,18,9,17,15,7,13,10,15,10,13,13,11,11,12,10,18,9,9,8,13,14,
16,10,8,8,18,8,11,9,17,18,14,17,14,12,10,10,10,7,13,18,18,14,12,18,15,14,13,10,7,11,17,18,14,
```

2.4 Нормальное распределение

Если стандартные равномерные случайные числа U_1 и U_2 независимо сгенерированы методом, установленным в разделе 2, то два независимых нормаль-

ных случайных числа Z_1, Z_2 получаются в соответствии со следующей процедурой:

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

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

Результат работы программы записан в файле distr-nr.dat:

```

rnc.cpp x distr-nr.dat x
distr-nr.dat
1 135,1,357,243,763,580,891,185,135,40,963,157,789,648,254,641,529,523,930,781,988,823,200,604,
214,388,607,749,363,329,972,77,875,58,140,896,527,369,829,431,392,956,15,943,33,891,78,26,
175,812,288,476,929,114,182,862,720,139,919,295,837,628,905,1,337,750,905,729,494,178,542,
498,477,93,254,57,312,433,496,963,511,900,807,155,687,352,430,262,464,905,249,763,934,275,
709,444,366,491,933,38,1,371,355,526,952,746,300,848,667,85,552,130,60,925,230,536,348,586,
484,292,719,145,657,753,387,389,281,669,963,345,791,7,13,977,4,299,52,55,353,239,79,981,149,
792,999,635,828,881,589,466,218,407,360,717,783,205,702,516,868,909,408,832,222,831,328,18,
245,520,213,51,884,269,423,955,975,961,383,882,136,590,693,57,467,282,938,476,572,99,697,655,
515,655,483,944,202,115,603,451,191,650,947,712,672,36,74,715,151,179,638,28,959,857,275,628,
934,458,680,382,478,894,397,437,122,794,833,658,521,18,287,654,20,520,109,315,527,741,362,84,
649,459,901,500,786,929,894,430,251,437,485,676,406,233,396,117,622,962,404,644,740,701,83,
904,475,647,17,52,972,342,382,857,274,295,277,292,975,72,96,691,629,719,842,248,16,569,278

```

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

Используя независимые равномерные случайные числа U_1, U_2, \dots, U_k , применяется формула:

$$Y = a - b * \ln\{(1 - U_1)(1 - U_2) \dots (1 - U_k)\}.$$

Результат работы программы записан в файле distr-gm.dat:

```

rnc.cpp x distr-gm.dat x
distr-gm.dat
1 8,7,11,9,16,13,17,9,8,7,18,8,16,14,10,14,13,13,18,16,18,16,9,14,9,11,14,15,11,10,18,7,17,7,8,
17,13,11,16,12,11,18,7,18,7,17,7,7,9,16,10,12,18,8,9,17,15,8,18,10,17,14,17,7,11,15,17,15,12,
9,13,12,12,8,10,7,10,12,12,18,13,17,16,8,15,11,12,10,12,17,9,16,18,10,15,12,11,12,18,7,7,11,
11,13,18,15,10,17,14,8,13,8,7,18,9,13,11,14,12,10,15,8,14,16,11,11,10,15,18,11,16,7,7,18,7,
10,7,7,11,9,7,18,8,16,18,14,16,17,14,12,9,11,11,15,16,9,15,13,17,17,11,16,9,16,10,7,9,13,9,7,
17,10,12,18,18,18,11,17,8,14,15,7,12,10,18,12,13,8,15,14,13,14,12,18,9,8,14,12,9,14,18,15,15,
7,7,15,8,9,14,7,18,17,10,14,18,12,15,11,12,17,11,12,8,16,16,14,13,7,10,14,7,13,8,10,13,15,11,
8,14,12,17,12,16,18,17,12,10,12,12,15,11,9,11,8,14,18,11,14,15,15,7,17,12,14,7,7,18,9,10,17,
10,10,10,11,18,7,8,15,14,15,17,9,7,13,11,12,7,12,9,8,11,12,16,18,18,18,7,11,10,14,12,11,9,16,
8,18,7,15,16,9,9,13,15,12,13,17,13,10,13,10,11,16,15,11,9,16,10,14,13,15,12,14,16,12,13,7,12,
10,13,12,14,11,9,18,15,14,8,14,14,12,7,9,12,16,14,17,11,18,10,17,16,9,17,7,8,15,17,10,17,17,
13,9,17,7,17,11,11,9,13,9,15,12,16,18,9,17,15,7,13,10,15,10,13,13,11,11,12,10,18,9,9,8,13,14,

```

2.6 Логнормальное распределение

Используя стандартные нормальные случайные числа Z , применяется формула:

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

Результат работы программы записан в файле distr-ln.dat:

```

rnc.cpp x distr-ln.dat x
distr-ln.dat
1 12,11,15,14,20,18,22,13,12,11,23,13,21,19,14,19,17,17,23,21,23,21,13,18,13,16,18,20,15,15,23,
12,22,11,12,22,17,15,21,16,16,23,11,23,11,22,12,11,13,21,14,17,23,12,13,22,20,12,22,14,21,19,
22,11,15,20,22,20,17,13,18,17,17,12,14,11,15,16,17,23,17,22,21,13,19,15,16,14,17,22,14,20,23,
14,20,16,15,17,23,11,11,15,15,17,23,20,14,22,19,12,18,12,11,23,13,17,15,18,17,14,20,12,19,20,
16,16,14,19,23,15,21,11,11,23,11,14,11,11,15,14,12,23,12,21,23,19,21,22,18,17,13,16,15,20,21,
13,20,17,22,22,16,21,13,21,15,11,14,17,13,11,22,14,16,23,23,23,15,22,12,18,20,11,17,14,23,17,
18,12,20,19,17,19,17,23,13,12,18,16,13,19,23,20,19,11,11,20,12,13,19,11,23,22,14,19,23,16,19,
15,17,22,16,16,12,21,21,19,17,11,14,19,11,17,12,15,17,20,15,12,19,16,22,17,21,23,22,16,14,16,
17,19,16,14,16,12,19,23,16,19,20,20,12,22,17,19,11,11,23,14,14,22,14,14,14,16,23,11,12,19,19,
20,21,14,11,18,15,17,12,16,13,12,16,16,21,23,23,23,11,15,14,19,17,16,13,21,12,23,11,20,21,13,
14,18,19,16,17,22,18,14,17,14,16,21,19,15,13,21,14,19,18,19,17,19,20,17,18,12,16,15,17,16,18,

```

2.7 Логистическое распределение

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

Результат работы программы записан в файле distr-ls.dat:

```

rnc.cpp x distr-ls.dat x
distr-ls.dat
1 9,7,12,10,19,16,21,9,9,7,22,9,19,17,11,17,15,15,21,19,22,20,10,16,10,13,16,18,12,12,22,8,20,
7,9,21,15,12,20,13,13,22,7,22,7,21,8,7,9,19,11,14,21,8,9,20,18,9,21,11,20,17,21,7,12,18,21,
18,14,9,15,14,14,8,11,7,11,13,14,22,15,21,19,9,17,12,13,11,14,21,10,19,21,11,18,14,12,14,21,
7,7,12,12,15,22,18,11,20,17,8,15,9,7,21,10,15,12,16,14,11,18,9,17,19,13,13,11,17,22,12,19,7,
7,22,7,11,7,7,12,10,8,22,9,19,22,17,20,21,16,14,10,13,12,18,19,10,18,15,20,21,13,20,10,20,12,
7,10,15,10,7,21,11,13,22,22,22,13,21,9,16,18,7,14,11,21,14,16,8,18,17,15,17,14,22,10,8,16,14,
10,17,22,18,17,7,8,18,9,9,17,7,22,20,11,17,21,14,17,13,14,21,13,13,8,19,20,17,15,7,11,17,7,
15,8,12,15,18,12,8,17,14,21,14,19,21,21,13,11,13,14,17,13,10,13,8,16,22,13,17,18,18,8,21,14,
17,7,7,22,10,11,20,11,11,11,13,22,8,8,18,17,18,20,10,7,16,13,14,8,14,9,9,13,13,20,21,21,21,7,
12,11,16,14,13,10,20,9,21,7,18,19,10,10,15,18,13,15,21,16,11,15,11,13,19,17,12,10,20,11,17,
16,17,14,17,19,14,16,8,14,12,15,14,16,13,10,21,17,16,8,17,17,14,7,10,14,20,16,21,13,22,12,20,
19,10,21,7,8,17,20,11,21,21,16,10,20,7,20,12,13,10,15,10,18,14,19,22,10,20,18,7,15,11,17,12,

```

2.8 Биноминальное распределение

Вычисляется функция распределения:

$$F(y) = \sum_{k=0}^y \binom{n}{k} p^k (1-p)^{n-k}, y = 0, 1, \dots, n.$$

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

Результат работы программы записан в файле distr-bi.dat:


```
src.cpp  distr-bi.dat X
distr-bi.dat
1  5,5,6,6,8,7,9,5,5,9,5,8,8,6,8,7,7,9,8,9,9,5,8,6,6,8,8,6,6,9,5,9,5,9,7,6,9,7,6,9,5,9,5,9,
5,5,5,9,6,7,9,5,5,9,8,5,9,6,9,8,9,5,6,8,9,8,7,5,7,7,5,6,5,6,7,7,9,7,9,9,5,8,6,7,6,7,9,6,8,
9,6,8,7,6,7,9,5,5,6,7,9,8,6,9,8,5,7,5,5,9,6,7,6,7,6,8,5,8,8,6,6,6,8,9,6,8,5,5,9,5,6,5,5,
6,6,5,9,5,8,9,8,9,9,7,7,6,7,6,8,8,6,8,7,9,9,7,9,6,9,6,5,6,7,6,5,9,6,7,9,9,9,6,9,5,7,8,5,7,6,
9,7,7,5,8,8,7,8,7,9,6,5,8,7,5,8,9,8,8,5,5,8,5,5,8,5,9,9,6,8,9,7,8,6,7,9,6,7,5,8,9,8,7,5,6,8,
5,7,5,6,7,8,6,5,8,7,9,7,8,9,9,7,6,7,7,8,7,6,6,5,8,9,7,8,8,8,5,9,7,8,5,5,9,6,6,9,6,6,6,6,9,5,
5,8,8,8,9,6,5,7,6,7,5,7,5,5,7,7,9,9,9,9,5,6,6,8,7,7,6,9,5,9,5,8,9,6,6,7,8,7,7,9,7,6,7,6,6,8,
8,6,6,9,6,8,7,8,7,8,8,7,7,5,7,6,7,7,8,6,5,9,8,8,5,8,8,7,5,6,7,9,7,9,6,9,6,9,8,6,9,5,5,8,9,6,
9,9,7,6,9,5,9,6,7,6,7,5,8,7,8,9,5,9,8,5,7,6,8,6,7,7,7,6,7,6,9,6,6,5,7,7,9,6,5,5,9,5,7,5,9,9,
8,9,8,7,6,6,6,5,7,9,9,7,7,9,8,7,7,6,5,6,9,9,8,6,5,6,6,5,6,8,9,6,6,8,7,5,7,9,7,9,6,5,5,9,8,9,
8,6,9,6,9,8,5,5,8,7,5,7,9,6,8,7,6,8,9,5,5,8,6,7,9,5,8,8,7,5,6,9,7,6,5,8,6,9,6,6,6,5,5,9,9,
9,7,6,6,7,6,9,9,7,9,8,7,8,5,6,9,7,6,7,7,5,9,8,6,8,7,6,8,8,5,6,6,6,5,6,7,8,5,7,9,9,5,5,6,7,9,
```


Приложение А

Код задание 1. prng.cpp

```
#include <iostream>
#include <vector>
#include <string>
#include <random>
#include <sstream>
#include <fstream>
#include <cmath>
#include <cstdint>
#include <windows.h>

using namespace std;

struct Parameter {
    int numbers_count;
    string output_file;

    Parameter() :
        numbers_count(10000),
        output_file("rnd.dat") {
    }
};

int pow2(int d) {
    return 1 << d;
}

void lc(const Parameter& parameters, const vector<int>& args) {
    int m = args[0];
    int a = args[1];
    int c = args[2];
    int x = args[3];

    if (m <= 0 || a > m || a < 0 || c > m || c < 0 || x > m || x < 0) {
        cout << endl << "Ошибка!" << endl;
        return;
    }

    int xi;
    vector<int> res;
    double per = 0.1;
    for (int i = 0; i < parameters.numbers_count; i++) {
        xi = (a * x + c) % m;
        res.push_back(xi % 1001);
        x = xi;

        if (static_cast<double>(i) / parameters.numbers_count >= per) {
            cout << endl << per * 100 << "%\n";
            per += 0.1;
        }
    }

    cout << endl << "100%" << endl;

    ofstream wr(parameters.output_file, ios::out);
    for (int i = 0; i < size(res); i++) {
        wr << res[i] << ", ";
    }
    wr.close();
}
```

```

}

void add(const Parameter& parameters, const vector<int>& args) {
    double per = 0.1;

    int m = args[0];
    int x0 = args[1];
    int x1 = args[2];

    if (m <= 0 || x0 >= x1 || x0 < 1 || x1 + 3 > size(args)) {
        cout << endl << "Ошибка!" << endl;
        return;
    }

    vector<int> res(begin(args) + 3, end(args));

    int xi;
    int num = size(res);

    for (int i = num; i < parameters.numbers_count + num; i++) {
        res.push_back((res[i - x0] + res[i - x1]) % m);
        if (static_cast<double>(i) / parameters.numbers_count >= per) {
            cout << endl << per * 100 << "%\n";
            per += 0.1;
        }
    }
    cout << endl << "100%" << endl;

    ofstream wr(parameters.output_file, ios::out);
    for (int i = x1; i < size(res); i++) {
        wr << res[i] % 1001 << ",";
    }
    wr.close();
}

void p5(const Parameter& parameters, const vector<int>& args) {
    double per = 0.1;
    int p = args[0];
    int q1 = args[1];
    int q2 = args[2];
    int q3 = args[3];
    int w = args[4];

    if (q1 >= p || q2 >= p || q3 >= p) {
        cout << endl << "Ошибка!" << endl;
        return;
    }

    vector<int> reg(p);
    for (int i = 0; i < p; i++) {
        reg[i] = rand() % 2;
    }

    ofstream wr(parameters.output_file, ios::out);

    for (int i = 0; i < parameters.numbers_count; i++) {
        int x_np = (reg[q1 - 1] ^ reg[q2 - 1] ^ reg[q3 - 1]);
        reg.push_back(x_np);
        reg.erase(begin(reg));

        int new_num = 0;
    }
}

```

```

    int d = 1;
    for (int i = 0; i < w; i++) {
        new_num = (new_num + reg[i] * d) % 1001;
        d *= 2;
    }

    wr << new_num % 1001 << ", ";

    if (static_cast<double>(i) / parameters.numbers_count >= per) {
        cout << endl << per * 100 << "%\n";
        per += 0.1;
    }

}
cout << endl << "100%" << endl;

wr.close();
}

int lfsr_func(const vector<int>& a_i, const vector<int>& reg_st) {
    int res = 0;

    for (int i = 0; i < size(a_i); i++) {
        res += a_i[i] * reg_st[i] % 2;
    }
    res += reg_st.back() % 2;

    return res;
}

int lfsr_vec_to_num(const vector<int>& reg_st) {
    int res = 0;

    for (int i = size(reg_st) - 1; i >= 0; i--) {
        res += reg_st[i] * pow2(i);
    }

    return res;
}

int lfsr_str_to_num(const string& reg_st) {
    int res = 0;

    for (int i = size(reg_st) - 1; i >= 0; i--) {
        res += reg_st[i] * pow2(i);
    }

    return res;
}

string num_to_str(int num, int n) {
    string res;

    while (num > 0) {
        res = to_string(num % 2) + res;
        num /= 2;
    }

    while (size(res) < n) {
        res = "0" + res;
    }
}

```

```

    return res;
}

string sdvig(const string& str, const string& new_x) {
    return str.substr(1) + new_x;
}

void lfsr(const Parameter& parameters, const vector<string>& args) {
    double per = 0.1;

    int x0 = stoi(args[0]);
    if (x0 < 0) {
        cout << endl << "Ошибка!" << endl;
        return;
    }

    ofstream wr(parameters.output_file, ios::out);

    string reg = args[1];
    for (int i = 0; i < parameters.numbers_count; i++) {
        string bin_x0 = num_to_str(x0, size(reg));
        int new_x = 0;

        for (int j = 0; j < size(reg); j++) {
            if (reg[size(reg) - j - 1] == '1' && bin_x0[size(bin_x0) - j - 1] == '1')
            {
                new_x++;
            }
        }

        x0 = x0 >> 1;

        if (new_x % 2 == 1) {
            int bits = size(reg);
            x0 = ((int)x0 + pow2(bits - 1)) % pow2(bits);
        }

        wr << x0 % 1001 << ", ";

        if (static_cast<double>(i) / parameters.numbers_count >= per) {
            cout << endl << per * 100 << "%\n";
            per += 0.1;
        }

    }

    cout << endl << "100%" << endl;
    wr.close();
}

vector<int> lfsr2(int n, const string& vec, int x0) {
    string reg(vec);
    vector<int> res;

    for (int i = 0; i < n; i++) {
        string bin_x0 = num_to_str(x0, size(reg));
        int new_x = 0;

        for (int j = 0; j < size(reg); j++) {
            if (reg[size(reg) - j - 1] == '1' && bin_x0[size(bin_x0) - j - 1] == '1')
            {
                new_x++;
            }
        }
    }

```

```

    }
}
x0 = x0 >> 1;

if (new_x % 2 == 1) {
    int bits = size(reg);
    x0 = ((int)x0 + pow2(bits - 1)) % pow2(bits);
}

res.push_back(x0 % 1001);

}

return res;
}

void nfsr(const Parameter& parameters, const vector<string>& args) {
    double per = 0.1;

    vector<int> R1 = lfsr2(parameters.numbers_count, args[0],
lfsr_str_to_num(args[0]));
    vector<int> R2 = lfsr2(parameters.numbers_count, args[1],
lfsr_str_to_num(args[1]));
    vector<int> R3 = lfsr2(parameters.numbers_count, args[2],
lfsr_str_to_num(args[2]));

    vector<int> res;

    ofstream wr(parameters.output_file, ios::out);

    for (int i = 0; i < parameters.numbers_count; i++) {
        wr << ((R1[i] & R2[i]) ^ (R2[i] & R3[i]) ^ R3[i]) << ",";

        if (static_cast<double>(i) / parameters.numbers_count >= per) {
            cout << endl << per * 100 << "%\n";
            per += 0.1;
        }
    }

    cout << endl << "100%" << endl;

    wr.close();
}

int mt_str_to_num(const vector<int>& reg_st) {
    int res = 0;

    for (int i = size(reg_st) - 1; i >= 0; i--) {
        res += reg_st[i] * pow2(i);
    }

    return res;
}

void mt(const Parameter& parameters, const vector<int>& args) {
    double per = 0.1;

    int p = 624, w = 32, r = 31, q = 397, a = 2567483615, u = 11, s = 7, t = 15, l =
18, b = 2636928640, c = 4022730752;
    int64_t u_v = 2147483648;

```

```

int u_n = 11;
int64_t h_v = 2147483647;
int mod = args[0];
int x0 = args[1];

if (x0 < 0 || mod <= 0) {
    cout << endl << "Ошибка!" << endl;
    return;
}

vector<int64_t> X;
X.push_back(x0);

for (int i = 1; i < p; i++) {
    X.push_back(abs(1812433253 * (X[i - 1] ^ (X[i - 1] >> 30)) + i));
}

ofstream wr(parameters.output_file, ios::out);

int it = 0;
for (int i = 0; i < parameters.numbers_count; i++) {
    int y_n = ((X[it] & u_n) | (X[it] & h_v));

    if (y_n % 2 == 1)
        X[it] = X[(it + q) % p] ^ (y_n >> 1) ^ a;
    else
        X[it] = X[(it + q) % p] ^ (y_n >> 1) ^ 0;

    y_n = X[it];
    y_n = y_n ^ (y_n >> u);
    y_n = y_n ^ ((y_n << s) & b);
    y_n = y_n ^ ((y_n << t) & c);
    int Z = y_n ^ (y_n >> l);

    wr << Z % mod % 1001 << ", ";

    it = (it + 1) % p;

    if (static_cast<double>(i) / parameters.numbers_count >= per) {
        cout << endl << per * 100 << "%" << endl;
        per += 0.1;
    }
}
cout << endl << "100%" << endl;

wr.close();
}

void rc4(const Parameter& parameters, const vector<int>& args) {
    double per = 0.1;

    vector<int> K(256), S(256);

    for (int i = 0; i < 256; i++) {
        K[i] = args[i];
        S[i] = i;

        if (K[i] < 0) {
            cout << endl << "Ошибка!" << endl;
            return;
        }
    }
}

```

```

int j = 0;

for (int i = 0; i < 256; i++) {
    j = (j + S[i] + K[i]) % 256;
    swap(S[i], S[j]);
}

ofstream wr(parameters.output_file, ios::out);

int i = 0;
j = 0;

for (int k = 0; k < parameters.numbers_count; k++) {
    i = (i + 1) % 256;
    j = (j + S[i]) % 256;

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

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

    wr << R % 1001 << ", ";

    if (static_cast<double>(i) / parameters.numbers_count >= per) {
        cout << endl << per * 100 << "%" << endl;
        per += 0.1;
    }
}

cout << endl << "100%" << endl;

wr.close();
}

int rsa_2_to_10(const vector<int>& reg_st) {
    int res = 0;

    for (int i = size(reg_st) - 1; i >= 0; i--) {
        res += reg_st[i] * pow2(i);
    }

    return res;
}

int rsa_pow(int x, int y, int m) {
    int res = 1;

    for (int i = 0; i < y; i++) {
        res = res * x % m;
    }

    return res;
}

void rsa(const Parameter& parameters, const vector<int>& args) {
    double per = 0.1;

    int n = args[0];
    int e = args[1];
    int x1 = args[2];
    int l = 10;

```



```

if (e <= 1 || x1 < 1 || x1 > n - 1) {
    cout << endl << "Ошибка!" << endl;
    return;
}

ofstream wr(parameters.output_file, ios::out);

for (int k = 0; k < parameters.numbers_count; k++) {
    int x2;
    vector<int> z(1);

    for (int i = 0; i < l; i++) {
        x2 = rsa_pow(x1, e, n);
        z[i] = x2 % 2;
        x1 = x2;
    }

    int res_z = rsa_2_to_10(z);

    wr << (res_z + 1000) % 1001 << ",";

    if (static_cast<double>(k) / parameters.numbers_count >= per) {
        cout << endl << per * 100 << "%\n";
        per += 0.1;
    }
}

cout << endl << "100%" << endl;

wr.close();
}

```

```

void bbs(const Parameter& parameters, const vector<int>& args) {
    double per = 0.1;
    int p = 127, q = 131, n = 16637, x0 = args[0], l = 11;

    if (x0 <= 1 || n % x0 == 0 || x0 % n == 0) {
        cout << endl << "Ошибка!" << endl;
        return;
    }

    ofstream wr(parameters.output_file, ios::out);

    for (int k = 0; k < parameters.numbers_count; k++) {
        int x2 = (x0 * x0) % n;

        vector<int> z;
        for (int i = 0; i < l; i++) {
            x2 = (x0 * x0) % n;
            z.push_back(x2 % 2);
            x0 = x2;
        }

        int res_z = rsa_2_to_10(z);

        wr << res_z % 1001 << ",";

        if (static_cast<double>(k) / parameters.numbers_count >= per) {
            cout << endl << per * 100 << "%\n";
            per += 0.1;
        }
    }
}

```

```

    }

    cout << endl << "100%" << endl;

    wr.close();
}

template <class T>
vector<T> splitStr(const string& str) {
    vector<T> elems;
    T elem;
    stringstream ss(str);

    while (ss >> elem) {
        elems.push_back(elem);

        if (ss.peek() == ',') {
            ss.ignore();
        }
    }

    return elems;
}

string to_str(char* s) {
    string res(s);
    res = res.substr(3);

    return res;
}

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

    SetConsoleOutputCP(CP_UTF8);

    int it = 0;
    int ch = 0;
    vector<int> int_args;
    vector<string> string_args;

    Parameter parameters;
    int nn = 10000;
    string f = "rnd.dat";

    for (int i = 1; argv[i]; i++) {
        string str_arg = string(argv[i]);
        if (str_arg[1] == 'g') {

            string ggg = str_arg.substr(3, 2);
            cout << endl << ggg << endl;
            if (ggg == "lc") {
                ch = 1;
                int_args = { 31104, 625, 6571, 33 };
            }
            else if (ggg == "ad") {
                ch = 2;
                int_args = {
5001, 29, 49, 816, 159, 798, 290, 168, 441, 691, 655, 874, 220, 125, 977, 586, 381, 868, 294, 948, 437, 58
8001, 29, 49, 816, 159, 798, 290, 168, 441, 691, 655, 874, 220, 125, 977, 586, 381, 868, 294, 948, 437, 58

```

```

1,181,701,536,11,672,103,601,794,189,12,130,386,828,288,183,117,456,624,807,110,498,2
7,234,474,613,615,341,906,562,778,486,155,276,894,441,226,762,234,762,98,458,399,445,
765,223,879 };
    }
    else if (ggg == "5p") {
        ch = 3;
        int_args = { 4253,1093,2254,3297,16 };
    }
    else if (ggg == "lfsr") {
        ch = 4;
        string_args = { "123","101101101101011" };
    }
    else if (ggg == "nfsr") {
        ch = 5;
        string_args = { "101101101101011","101101101101011","101101101101011"
};
    }
    else if (ggg == "mt") {
        ch = 6;
        int_args = { 10001,8191 };
    }
    else if (ggg == "rc") {
        ch = 7;
        int_args = {
802,720,341,337,961,882,417,785,198,727,899,372,374,425,556,615,813,768,840,183,893,5
68,73,387,18,436,182,125,806,899,485,607,619,825,944,579,707,360,363,904,87,262,276,4
60,687,831,75,499,599,915,681,492,483,754,878,500,189,60,624,994,959,109,600,577,934,
544,156,640,903,519,544,990,781,819,449,468,650,524,967,248,438,647,739,920,400,617,4
19,588,676,43,581,634,151,181,211,84,724,367,723,627,886,267,617,667,85,65,134,735,58
9,100,983,26,747,721,945,147,337,364,734,13,406,315,647,556,496,858,640,220,224,362,8
47,110,629,463,776,713,528,909,448,116,9,430,141,755,151,86,901,488,449,635,500,855,9
50,147,410,446,4,49,665,227,411,511,336,39,974,112,752,501,21,200,617,29,629,757,784,
779,843,684,266,292,319,766,146,269,912,556,714,916,605,378,142,15,889,478,54,862,590
,806,363,610,5,979,638,634,736,421,413,578,105,679,869,424,444,14,692,356,569,405,271
,173,783,413,188,671,891,242,533,480,48,895,89,53,873,727,686,608,147,98,185,252,776,
54,675,220,67,366,576,636,771,846,808,553,259,996,224,149 };
    }
    else if (ggg == "rs") {
        ch = 8;
        int_args = { 7191817,151,69 };
    }
    else if (ggg == "bb") {
        ch = 9;
        int_args = { 8627 };
    }
}
else if (str_arg[1] == 'i' && (ch == 4 || ch == 5)) {
    string_args = splitStr<string>(to_str(argv[i]));
}
else if (str_arg[1] == 'i') {
    int_args = splitStr<int>(to_str(argv[i]));
}
else if (str_arg[1] == 'n') {
    parameters.numbers_count = stoi(to_str(argv[i]));
}
else if (str_arg[1] == 'f') {
    parameters.output_file = to_str(argv[i]);
}
else if (str_arg[1] == 'h') {
    cout << "Введена команда с /h. Допустимые параметры:";
    cout << "\n\n/g:<код_метода> - параметр указывает на метод генерации ПСЧ,
при этом код_метода может быть одним из следующих:\n";

```

```

        cout << "\n  lc - линейный конгруэнтный метод (Вход: модуль, множитель,
приращение, начальное значение)"
        << "\n  add - аддитивный метод(Вход: модуль, младший индекс, старший
индекс, последовательность начальных значений)"
        << "\n  5p - пятипараметрический метод(Вход: p, q1, q2, q3, w)"
        << "\n  lfsr - регистр сдвига с обратной связью(РСЛОС) (Вход: двоич-
ное представление вектора коэффициентов, начальное значение регистра)"
        << "\n  nfsr - нелинейная комбинация РСЛОС(Вход: двоичное представле-
ние векторов коэффициентов для R1, R2, R3)"
        << "\n  mt - вихрь Мерсенна(Вход: модуль, начальное значение x)"
        << "\n  rc4 - RC4(Вход: 256 начальных значений)"
        << "\n  rsa - ГПСЧ на основе RSA(Вход: модуль n, число e, начальное
значение x; e удовлетворяет условиям :  $1 < e < (p - 1)(q - 1)$ , НОД(e,  $(p - 1)(q - 1)$ )
= 1, где  $p * q = n$ . x из интервала[1, n])"
        << "\n  bbs - алгоритм Блюма - Блюма - Шуба(Вход: Начальное значение
x(взаимно простое с n)); \n";
        cout << "\n\n/i:<число> - инициализационный вектор генератора.";
        cout << "\n\n/n:<длина> - количество генерируемых чисел. Если параметр не
указан, - генерируется 10000 чисел.";
        cout << "\n\n/f:<полное_имя_файла> - полное имя файла, в который будут
выводиться данные. Если параметр не указан, данные должны записываться в файл с име-
нем rnd.dat.";
        cout << "\n\n/h - информация о допустимых параметрах командной строки
программы.\n";
    }
}

switch (ch) {
case 1:
    lc(parameters, int_args);
    return 0;
case 2:
    add(parameters, int_args);
    return 0;
case 3:
    p5(parameters, int_args);
    return 0;
case 4:
    lfsr(parameters, string_args);
    return 0;
case 5:
    nfsr(parameters, string_args);
    return 0;
case 6:
    mt(parameters, int_args);
    return 0;
case 7:
    rc4(parameters, int_args);
    return 0;
case 8:
    rsa(parameters, int_args);
    return 0;
case 9:
    bbs(parameters, int_args);
    return 0;
default:
    return 0;
}

return 0;
}

```

Приложение Б

Код задание 2. rnc.cpp

```
#define _USE_MATH_DEFINES
#include <iostream>
#include <vector>
#include <string>
#include <random>
#include <sstream>
#include <fstream>
#include <cmath>
#include <windows.h>
#include <ranges>
#include <algorithm>

using namespace std;

struct Parameter {
    int numbers_count;
    string output_file;

    Parameter() :
        numbers_count(10000),
        output_file("result.txt") {
    }
};

vector<int> convert_file_to_ints(const string& path) {
    ifstream input(path);

    if (!input.is_open()) {
        cout << "ERROR" << endl;
        return {};
    }

    vector<int> res;

    string temp;
    getline(input, temp);
    stringstream splitter(temp);
    while (getline(splitter, temp, ',')) {
        res.push_back(stoi(temp));
    }

    input.close();

    return res;
}

string to_str(char* s) {
    string res(s);
    res = res.substr(3);

    return res;
}

string to_str_p(char* s) {
    string res(s);
    res = res.substr(4);
```

```

    return res;
}

vector<double> U(const vector<int>& arr) {

    auto max_it = max_element(std::begin(arr), std::end(arr));
    double max_value = *max_it;

    max_value++;

    vector<double> res;

    for (auto elem : arr) {
        res.push_back(elem / max_value);
    }

    return res;
}

void st(const Parameter& parameters, const vector<int>& arr, int a, int b) {
    ofstream output(parameters.output_file, ios::out);

    auto max_it = max_element(std::begin(arr), std::end(arr));
    int max_value = *max_it;
    max_value++;

    vector<int> res;
    for (auto elem : arr) {
        res.push_back(b * elem / max_value + a);
        output << res.back() << ", ";
    }

    output.close();
}

void tr(const Parameter& parameters, const vector<int>& v, int a, int b) {
    ofstream output(parameters.output_file, ios::out);

    vector<double> u = U(v);

    vector<int> res;
    for (int i = 0; i < size(v) - 1; i++) {
        double h = (static_cast<double>(a) + static_cast<double>(b) * (u[i] + u[i +
1] - 1));
        res.push_back(h);

        output << (int)res[i] << ", ";
    }

    output.close();
}

void ex(const Parameter& parameters, const vector<int>& v, int a, int b) {
    ofstream output(parameters.output_file, ios::out);

    vector<double> u = U(v);

    vector<int> res;
    for (auto elem : u) {
        if (elem == 0) {
            elem += 0.001;
        }
    }
}

```

```

        if (elem == 1) {
            elem -= 0.001;
        }

        double h = (static_cast<double>(a) - b * log(elem));

        res.push_back(h);

        output << res.back() << ",";
    }

    output.close();
}

void nr(const Parameter& parameters, const vector<int>& v, int a, int b) {
    ofstream output(parameters.output_file, ios::out);

    vector<double> u = U(v);

    vector<int> res;
    for (int i = 0; i < size(v); i += 2) {
        if (u[i] == 0) {
            u[i] += 0.001;
        }

        if (u[i] == 1) {
            u[i] -= 0.001;
        }

        double h1 = a + (double)b * sqrt(-2 * log(1 - u[i])) * cos(2 * M_PI * u[(i +
1) % size(v)]);
        res.push_back(h1);
        output << res[i] << ",";

        double h2 = (a + (double)b * sqrt(-2 * log(1 - u[i]))) * sin(2 * M_PI * u[(i +
1) % size(v)]);
        res.push_back(h2);
        output << res[i + 1] << ",";
    }

    output.close();
}

void gm(const Parameter& parameters, const vector<int>& v, int a, int b) {
    ofstream output(parameters.output_file, ios::out);

    vector<double> u = U(v);

    vector<int> res;

    for (int i = 0; i < size(v); i++) {
        if (u[i] == 0) {
            u[i] += 0.001;
        }

        if (u[i] == 1) {
            u[i] -= 0.001;
        }

        double h = (static_cast<double>(a) - static_cast<double>(b) * log(1 - u[i]));
        res.push_back(h);
        output << res[i] << ",";
    }
}

```



```

    }

    output.close();
}

vector<double> nr_for_ln(const vector<int>& v, int a, double b) {
    vector<double> u = U(v);

    vector<double> res;
    for (int i = 0; i < size(v); i += 2) {
        if (u[i] == 0) {
            u[i] += 0.001;
        }

        if (u[i] == 1) {
            u[i] -= 0.001;
        }

        double h1 = a + b * sqrt(-2 * log(1 - u[i])) * cos(2 * M_PI * u[(i + 1) %
size(v)]);
        h1 += 0.5;
        res.push_back(h1);

        double h2 = (a + b * sqrt(-2 * log(1 - u[i])) * sin(2 * M_PI * u[(i + 1) %
size(v)]));
        h2 += 0.5;
        res.push_back(h2);
    }

    return res;
}

void ln(const Parameter& parameters, const vector<int>& v, int a, int b) {
    ofstream output(parameters.output_file, ios::out);

    vector<double> res;
    if (b <= 42) {
        vector<double> u = nr_for_ln(v, 0, 0.1);
        for (auto elem : u) {
            if (elem == 0) {
                elem += 0.001;
            }

            if (elem == 1) {
                elem -= 0.001;
            }

            double h = (a + exp(b - elem));

            res.push_back(h);

            output << res.back() << ",";
        }
    }
    else {
        cout << "Error!" << endl;
    }

    output.close();
}

void ls(const Parameter& parameters, const vector<int>& v, int a, int b) {
    ofstream output(parameters.output_file, ios::out);

```

```

vector<double> u = U(v);

vector<int> res;
for (auto elem : u) {
    if (elem == 0) {
        elem += 0.001;
    }

    if (elem == 1) {
        elem -= 0.001;
    }

    double h = a + (double)b * log(elem / (1 - elem));
    res.push_back(h);
    output << res.back() << ",";
}

output.close();
}

double fact(double x) {
    return (x == 0 || x == 1) ? 1 : x * fact(x - 1);
}

double Cnk(double n, double k) {
    return fact(n) / (fact(n - k) * fact(k));
}

void bi(const Parameter& parameters, const vector<int>& v, int n, double p) {
    ofstream output(parameters.output_file, ios::out);

    vector<int> res;
    vector<double> u = U(v);
    for (int i = 0; i < size(v); i++) {
        double temp_res = 0;
        for (int k = 0; k <= n; k++) {
            temp_res += Cnk(n, k) * powf(p, k) * powf(1 - p, n - k);

            if (temp_res >= u[i]) {
                res.push_back(k);
                break;
            }
        }

        output << res[i] << ",";
    }

    output.close();
}

int main(int argc, char* argv[]) {
    SetConsoleOutputCP(CP_UTF8);

    string f = "rnd.dat";
    int programm_number = 0;
    vector<int> numbers;
    vector<string> string_numbers;
    int p1 = 0, p2 = 0;
    double p3 = 0;

    Parameter parameters;

```

```

for (int i = 1; argv[i]; i++) {
    string str_arg = string(argv[i]);
    if (str_arg[1] == 'd') {

        string ggg = str_arg.substr(3, 2);

        if (ggg == "st") {
            programm_number = 1;
            numbers = { 31460, 48498, 57017, 28508, 47022, 56279, 60907, 30453,
15226, 7613, 3806, 34671, 17335, 8667, 4333, 34934, 17467 };
        }
        else if (ggg == "tr") {
            programm_number = 2;
            numbers = { 937, 1284, 445, 444, 1335, 1132, 881, 1636, 454, 887,
1075, 1044, 780, 1313, 1059 };
        }
        else if (ggg == "ex") {
            programm_number = 3;
            numbers = { 4253, 1093, 2254, 3297, 16 };
        }
        else if (ggg == "nr") {
            programm_number = 4;
            string_numbers = { "123", "101101101101011" };
        }
        else if (ggg == "gm") {
            programm_number = 5;
            string_numbers = { "101101101101011", "101101101101011",
"101101101101011" };
        }
        else if (ggg == "ln") {
            programm_number = 6;
            numbers = { 10001, 8191 };
        }
        else if (ggg == "ls") {
            programm_number = 7;
            numbers = { 802, 720, 341, 337, 961, 882, 417, 785, 198, 727, 899,
372, 374, 425, 556, 615, 813, 768, 840, 183, 893, 568, 73, 387, 18, 436, 182, 125,
806, 899, 485, 607, 619, 825, 944, 579, 707, 360, 363, 904, 87, 262, 276, 460, 687,
831, 75, 499, 599, 915, 681, 492, 483, 754, 878, 500, 189, 60, 624, 994, 959, 109,
600, 577, 934, 544, 156, 640, 903, 519, 544, 990, 781, 819, 449, 468, 650, 524, 967,
248, 438, 647, 739, 920, 400, 617, 419, 588, 676, 43, 581, 634, 151, 181, 211, 84,
724, 367, 723, 627, 886, 267, 617, 667, 85, 65, 134, 735, 589, 100, 983, 26, 747,
721, 945, 147, 337, 364, 734, 13, 406, 315, 647, 556, 496, 858, 640, 220, 224, 362,
847, 110, 629, 463, 776, 713, 528, 909, 448, 116, 9, 430, 141, 755, 151, 86, 901, 488,
449, 635, 500, 855, 950, 147, 410, 446, 4, 49, 665, 227, 411, 511, 336, 39, 974, 112,
752, 501, 21, 200, 617, 29, 629, 757, 784, 779, 843, 684, 266, 292, 319, 766, 146,
269, 912, 556, 714, 916, 605, 378, 142, 15, 889, 478, 54, 862, 590, 806, 363, 610,
5, 979, 638, 634, 736, 421, 413, 578, 105, 679, 869, 424, 444, 14, 692, 356, 569, 405,
271, 173, 783, 413, 188, 671, 891, 242, 533, 480, 48, 895, 89, 53, 873, 727, 686,
608, 147, 98, 185, 252, 776, 54, 675, 220, 67, 366, 576, 636, 771, 846, 808, 553,
259, 996, 224, 149 };
        }
        else if (ggg == "bi") {
            programm_number = 8;
            numbers = { 7191817, 151, 69 };
        }
    }
    else if (str_arg[1] == 'p') {

        if (str_arg[2] == 'l') {
            pl = stoi(to_str_p(argv[i]));
        }
    }
}

```

```

    }
    else if (str_arg[2] == '2') {
        if (programm_number == 8) {
            p3 = stof(to_str_p(argv[i]));
        }
        else {
            p2 = stoi(to_str_p(argv[i]));
        }
    }
}
else if (str_arg[1] == 'f') {
    string path = to_str(argv[i]);
    numbers = convert_file_to_ints(path);
} else if (str_arg[1] == 'o') {
    string path = to_str(argv[i]);
    parameters.output_file = path;
}
}

switch (programm_number) {
case 1:
    st(parameters, numbers, p1, p2);
    return 0;
case 2:
    tr(parameters, numbers, p1, p2);
    return 0;
case 3:
    ex(parameters, numbers, p1, p2);
    return 0;
case 4:
    nr(parameters, numbers, p1, p2);
    return 0;
case 5:
    gm(parameters, numbers, p1, p2);
    return 0;
case 6:
    ln(parameters, numbers, p1, p2);
    return 0;
case 7:
    ls(parameters, numbers, p1, p2);
    return 0;
case 8:
    bi(parameters, numbers, p1, p3);
    return 0;
default:
    return 0;
}

return 0;
}

```