# Министерство образования и науки Российской Федерации

## Федеральное государственное бюджетное образовательное учреждение высшего образования

## «Новосибирский государственный технический университет»

NSTU_Logo_blue

## Кафедра теоретической и прикладной информатики

### Лабораторная работа № 2 по дисциплине «Информационная безопасность»

|  |  |  |
| --- | --- | --- |
| сигма градиент синий1 | Факультет: | ПМИ |
| Группа: | ПМИ-61 |
| Студенты: | Ершов П.К., Мамонова Е.В., Цыденов З.Б. |
| Вариант: | 2 |
| Контактный адрес | Ершов П.К. - [PMI62.Ershov@outlook.com](mailto:PMI62.Ershov@outlook.com)  Мамонова Е.В. - [PMI62.Mamonova@outlook.com](mailto:PMI62.Mamonova@outlook.com)  Цыденов З.Б.- [tsydebov.zanabazar@gmail.com](mailto:tsydebov.zanabazar@gmail.com) |
| Преподаватель: | Авдеенко Т.В. |

Новосибирск

2020

1. **Цель работы**

Изучить существующие алгоритмы вычисления дайджестов сообщений и написать программу, реализующую заданный алгоритм хэширования.

1. **Задание**
2. Реализовать приложение с графическим интерфейсом, позволяющее выполнять следующие действия.
3. Генерировать псевдослучайную последовательность с помощью заданного в варианте алгоритма:
4. все входные параметры генератора должны задаваться из файла или вводиться в приложении;
5. сгенерированная последовательность, состоящая из 0 и 1, должна сохраняться в файл;
6. Проверять полученную псевдослучайную последовательность на равномерность и случайность с помощью трех рассмотренных тестов:
7. результат проверки каждого теста должен отображаться в приложении;
8. все вычисляемые промежуточные значения (все шаги алгоритма теста) могут отображаться в приложении или сохраняться в файл.
9. С помощью реализованного приложения выполнить следующие задания.
10. Протестировать правильность работы разработанного приложения.
11. Сгенерировать последовательность из не менее 10 000 бит и исследовать ее на равномерность и случайность.
12. Сделать вывод о случайности сгенерированной последовательности и о возможности ее использования в качестве криптографически безопасной псевдослучайной последовательности.

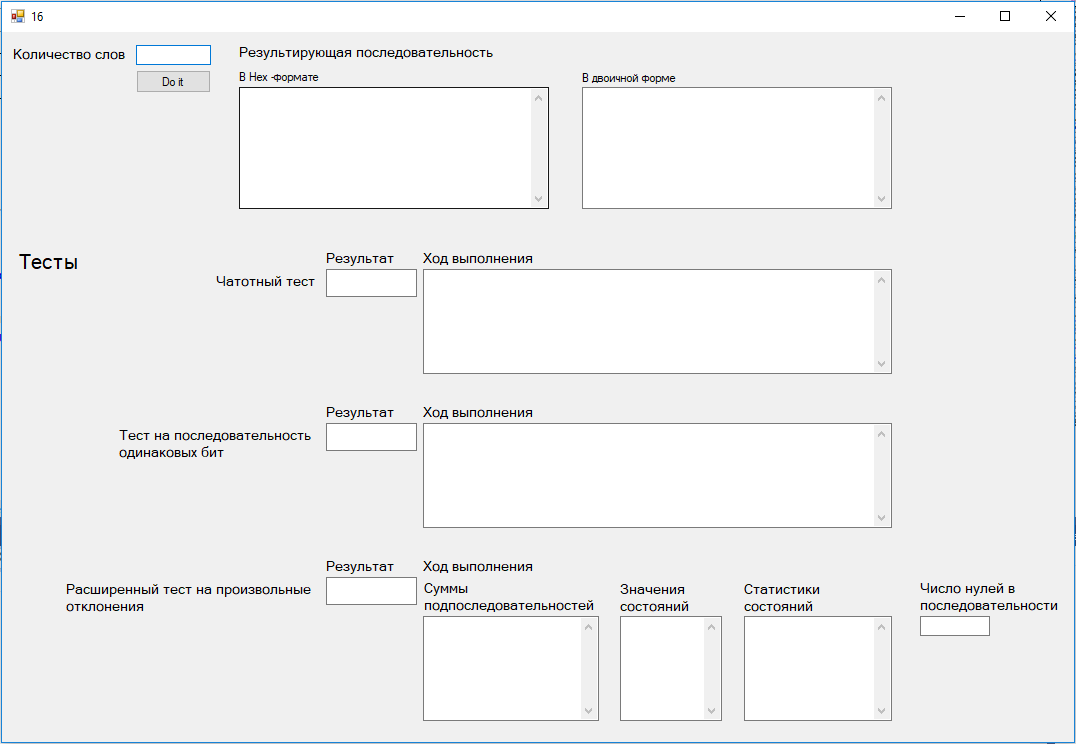
**Вариант:** Алгоритм ANSI X9.17

**3.** Описание разработанного программного средства

Разработанная программа способна генерировать псевдослучайную последовательность чисел заданной длины. Программа выводит сгенерированную последовательность в 16-ричном (для наглядности отличия 64-битных чисел друг от друга) и в двоичном формате, так же занося двоичный формат в выходной файл. Программа тестирует последовательность на случайность и равномерность применением трёх тестов и выводит как результаты тестов, так и их промежуточные результаты.

**Для запуска программного средства необходимо запустить файл IS\_lab\_2.exe, расположенный в IS\_lab\_2\IS\_lab\_2\bin\Debug\.**

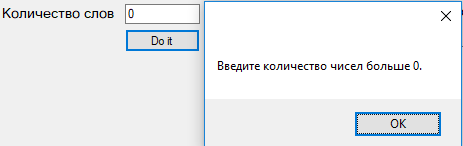
Интерфейс приложения:

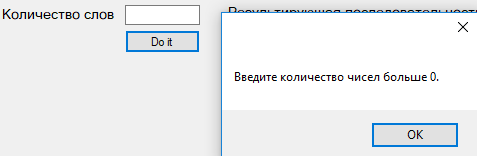


1. Исследования
   1. Демонстрация работоспособности на примере хэширования нескольких файлов.

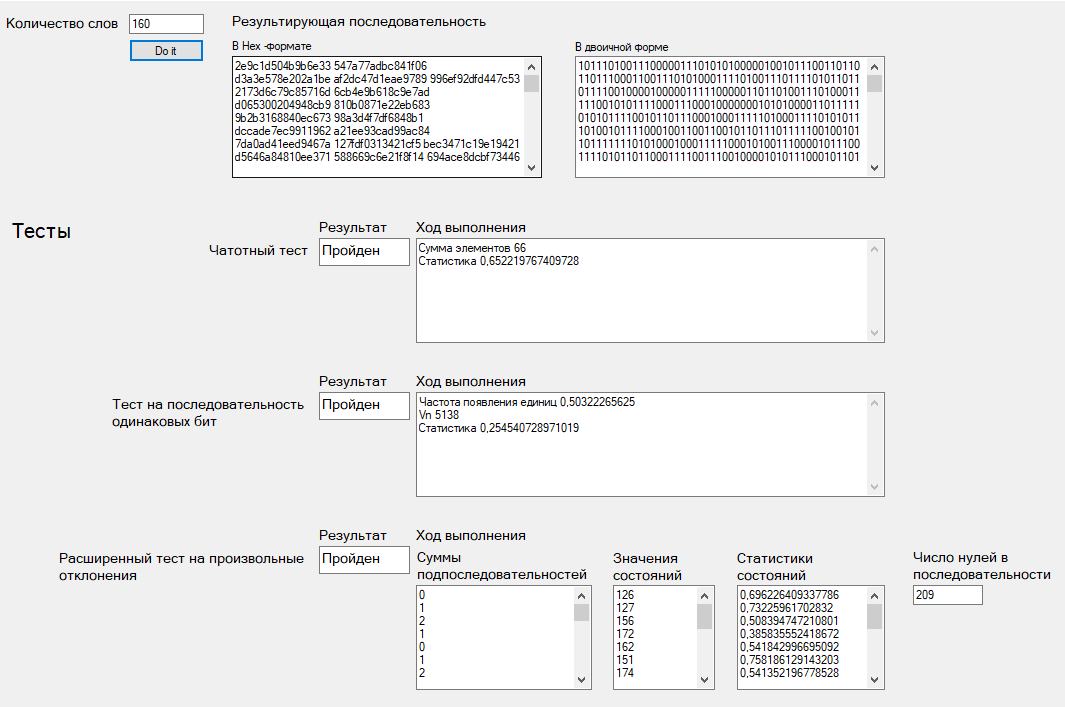
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № теста | Длина сообщения | Результаты | | Частотный тест | Тест на последовательность одинаковых бит | Расширенный тест на отклонения |
| 16-ричная форма | 2-чная форма |
| 1 | 2 | bccc5ed626046afe 9cceee895d70c197 | 10111100110011000101111011010110001001100000010001101010111111101001110011001110111011101000100101011101011100001100000110010111 | Пройден | Пройден | Пройден |
| 2 | 10 | 444987a9e5de38d9 b2c6abafbee7c88c a4d74487093661c6 1e5e853f93c3bbfc d033a88c925bbea7 53ef4ff0562bcede 799424a0981fb9cb b4db18496f08ddff d931f2a9605dd948 c46d75c37220f71b | 1000100010010011000011110101001111001011101111000111000110110011011001011000110101010111010111110111110111001111100100010001100101001001101011101000100100001110000100100110110011000011100011011110010111101000010100111111100100111100001110111011111111001101000000110011101010001000110010010010010110111011111010100111101001111101111010011111111000001010110001010111100111011011110111100110010100001001001010000010011000000111111011100111001011101101001101101100011000010010010110111100001000110111011111111111011001001100011111001010101001011000000101110111011001010010001100010001101101011101011100001101110010001000001111011100011011 | Пройден | Пройден | Пройден |
| 3 | 50 |  |  | Пройден | Пройден | Пройден |

Примеры ошибочной длины последовательности:





* 1. Исследование последовательности длиной более 10000 бит.



1. **Код программы**

Cripto.cs

class Cripto

{

const double F\_S = 1.82138636; // значение статистики для частотного теста

public void Write(string fileName, string[] outw) // Запись входной строки и ключа в файл

{

using (StreamWriter sw = new StreamWriter(fileName))

{

foreach (string str in outw)

sw.WriteLine(str);

}

}

public byte[] TriDES(byte[] Data, byte[] Key, byte[] IV) // 3DES алгоритм

{

MemoryStream mStream = new MemoryStream(); // создаём поток памяти

TripleDESCryptoServiceProvider tdes = new TripleDESCryptoServiceProvider(); // объект класса 3DES

tdes.Padding = PaddingMode.None; //отключаем дополнение выходного шифротекства справа

tdes.Mode = CipherMode.CFB; // включаепм режим CFB, чтобы установить длину шифротекста в 64 бита

CryptoStream cStream = new CryptoStream(mStream,

tdes.CreateEncryptor(Key, IV),

CryptoStreamMode.Write); // создаём поток для шифрования

byte[] toEncrypt = Data;

cStream.Write(toEncrypt, 0, toEncrypt.Length); // помещаем данные в поток шифрования

cStream.FlushFinalBlock();

byte[] ret = mStream.ToArray(); // извлекаем шифротекст

cStream.Close(); // закрываем потоки

mStream.Close();

return ret;

}

public byte[] key\_gen() // генератор 128-битного ключа

{

byte[] key = new byte[128]; // основной ключ

byte[] b1 = new byte[64]; // 64-битная часть ключа

byte[] b2 = new byte[64]; // 64-битная часть ключа

byte[] xor = new byte[64]; // 64-битная блок случайных бит для равномерного преобразования ключа

byte[] bit = new byte[1]; // случайный бит для выбора совмещения двух частей ключа (1 - !b1 + b2, 0 - b2 + !b1)

var rand = new Random(); // структура для рандома

rand.NextBytes(b1); // создаём случайную половину ключа

rand.NextBytes(b2); // создаём случайную половину ключа

rand.NextBytes(xor); // создаём случайную последовательность для оперции xor с половинами ключа

ulong k1 = BitConverter.ToUInt64(b1, 0); // переводим в ulong формат

ulong k2 = BitConverter.ToUInt64(b2, 0); // переводим в ulong формат

ulong xr = BitConverter.ToUInt64(xor, 0); // переводим в ulong формат

k1 ^= xr; // приводим к равномерному виду

xr = BitConverter.ToUInt64(xor, 0);

k2 ^= xr; // приводим к равномерному виду

rand.NextBytes(bit); // создаём случайное число (0 или 1), чтобы выбрать, какую половину ключа поставить первой и какую инвертировать

if (bit[0] == 1)

{

b1 = BitConverter.GetBytes(~k1); // обрачаем число

b2 = BitConverter.GetBytes(k2);

key = b1.Concat(b2).ToArray(); // соединяем половины 128-битного ключа

}

else

{

b1 = BitConverter.GetBytes(k1);

b2 = BitConverter.GetBytes(~k2); // обрачаем число

key = b2.Concat(b1).ToArray(); // соединяем половины 128-битного ключа

}

return key;

}

public ulong[] ANSI(int m) // алгоритм ANSI X9.17

{

ulong[] X = new ulong[m]; // результирующая последовательность из m 64-битных чисел

DateTime date1 = DateTime.Now; // структура для извлечения даты и времени

var rand = new Random(); // структура для получения рандомных чисел

byte[] s\_0 = new byte[64]; // начальное случайное секретное 64-битное число

rand.NextBytes(s\_0); //

for (int i = 0; i < m; i++) // основной цикл

{

byte[] d = BitConverter.GetBytes((ulong)date1.ToBinary()); // получаем дату и время в 64-битном формате

byte[] IV = new byte[128]; // 128-битный вектор инициализации для шифрования чисел 3DES

rand.NextBytes(IV); // получаем случайный вектор (использование случайного вектора позволяет избежать дублирующихся блоков в шифротексте)

byte[] buf = TriDES(d, key\_gen(), IV); // зашифрованные дата и время

ulong b1 = BitConverter.ToUInt64(buf, 0); // преобразуем зашифрованную дату и время в ulong для битовых операций

ulong s = BitConverter.ToUInt64(s\_0, 0); // преобразуем секретное число в ulong для битовых операций

s ^= b1; // проводим xor операцию с секретным числом и зашифрованными датой, временем

buf = BitConverter.GetBytes(s); // переводим результат xor обратно в массив байтов для повторного шифрования

rand.NextBytes(IV); // получаем новый случайный вектор инициализации

buf = TriDES(buf, key\_gen(), IV); // шифруем результат xor

X[i] = BitConverter.ToUInt64(buf, 0); // получаем одно из результирующих псевдослучайных чисел

b1 ^= X[i]; // проводим операцию xor с результирующим числом и предыдущим xor

buf = BitConverter.GetBytes(b1); // переводим xor в массив байтов

rand.NextBytes(IV); // шифруем результат второго xor чтобы получить новое начальное случайное число для следующей итерации цикла

}

return X;

}

public int[] zero\_invert(int[] X) // функция замены нулей на -1

{

int[] bin\_seq = new int[X.Length];

for (int i = 0; i < bin\_seq.Length; i++)

{

bin\_seq[i] = X[i];

if (bin\_seq[i] == 0)

bin\_seq[i] = -1;

}

return bin\_seq;

}

public int[] num\_convert(ulong[] X) // перевод массива чисел ulong в int массив 0 и 1

{

int[] bin\_seq = new int[X.Length \* 64]; // результирующий массив длиной в 64\*число элементов во входном массиве

int k = 0;

for(int i = 0; i < X.Length; i++)

{

BitArray buf = new BitArray(BitConverter.GetBytes(X[i])); // превод элемента массива ulong в масив байтов, а затем в массив битов

for (int j = 0; j < buf.Length; j++)

{

if (buf[j])

bin\_seq[k] = 1;

else

bin\_seq[k] = 0;

k++;

}

}

return bin\_seq;

}

public double[] freq\_test(int[] Seq) // частотный тест

{

int S = 0; // сумма всех элементов последовательности

double[] res = new double[3]; // результирующий массив (нужет для регистрации всех шагов теста)

for (int i = 0; i < Seq.Length; i++) // получаем сумму элементов последовательности

S += Seq[i];

double Stat = Math.Abs(S) / Math.Sqrt(Seq.Length); // находим статистику

res[0] = S; // сохраняем сумму

res[1] = Stat; // сохраняем статистику

if (Stat <= F\_S) // если статистика меньше или равна тестовой - тест пройден

res[2] = 1;

else

res[2] = 0;

return res;

}

public double[] same\_bits\_test(int[] Seq) // тест на последовательность одинаковых бит

{

double pi = 0; // частота появленяи единиц в последовательности

int V = 1; // количество ситуаций, при которых соседние числа последовательности не равны друг другу

double[] res = new double[4]; // результирующий массив (нужет для регистрации всех шагов теста)

for (int i = 0; i < Seq.Length; i++) // находим частоту встречи единиц в последовательности

pi += Seq[i];

pi /= Seq.Length;

for (int i = 0; i < Seq.Length - 1; i++) // находим все ситуации, когда соседние элементы не равны друг другу

if (Seq[i] != Seq[i + 1])

V += 1;

double Stat = Math.Abs(V - 2 \* pi \* Seq.Length \* (1 - pi)) /

(2 \* pi \* (1 - pi) \* Math.Sqrt(2 \* Seq.Length)); // статистика

res[0] = pi;

res[1] = V;

res[2] = Stat;

if (Stat <= F\_S)

res[3] = 1;

else

res[3] = 0;

return res;

}

public int[] state\_check(int[] S, int[] eps) // подсчёт количества встреч чисел от -9 до 9 (кроме 0) в последовательности

{

for (int i = -9; i < 0; i++)

for (int j = 0; j < S.Length; j++)

if (S[j] == i)

eps[i + 9]++;

for (int i = 1; i < 10; i++)

for (int j = 0; j < S.Length; j++)

if (S[j] == i)

eps[i + 8]++;

return eps;

}

public bool Y\_check(double[] Y) // получение статистик для ситуаций, полученных в state\_check

{

for (int i = 0; i < Y.Length; i++)

if (Y[i] > F\_S)

return false;

return true;

}

public double[] rand\_dev\_test(int[] Seq) // расширенный тест на произвольные отклонения

{

int[] S = new int[Seq.Length + 2]; // новая последовательность возростающих сумм (содержит 0 в начальном и последнем элементах)

int[] eps = new int[18]; // количество встреч чисел от -9 до 9 (кроме 0) в S

double[] Y = new double[18]; // статистики для eps

int k = 0; // число нулей в S

int L = 0; // число нулей - 1 в S

for (int i = 1; i < Seq.Length + 1; i++) // получаем возростающие суммы

S[i] += S[i - 1] + Seq[i - 1];

for (int i = 0; i < S.Length; i++) // получаем число нулей

if (S[i] == 0)

k++;

L = k - 1;

eps = state\_check(S, eps); // получаем число встреч чисел

for (int i = -9; i < 0; i++) // получаем статистики для чисел от -9 до -1

Y[i + 9] = Math.Abs(eps[i + 9] - L) / Math.Sqrt(2 \* L \* (4 \* Math.Abs(i) - 2));

for (int i = 1; i < 10; i++) // получаем статистики для чисел от 1 до 9

Y[i + 8] = Math.Abs(eps[i + 8] - L) / Math.Sqrt(2 \* L \* (4 \* Math.Abs(i) - 2));

double[] res = new double[S.Length + 2 \* eps.Length + 2]; // результирующий массив (нужет для регистрации всех шагов теста)

for (int i = 0; i < S.Length; i++) // сохраняем возростающие суммы

res[i] = S[i];

for (int i = 0; i < eps.Length; i++) // сохраняем число встреч

res[i + S.Length] = eps[i];

for (int i = 0; i < eps.Length; i++) // сохраняем статистики

res[i + S.Length + eps.Length] = Y[i];

res[S.Length + 2 \* eps.Length] = L; // сохраняем число нулей

if (Y\_check(Y))

res[S.Length + 2 \* eps.Length + 1] = 1;

else

res[S.Length + 2 \* eps.Length + 1] = 0;

return res;

}

}

Form1.cs

private void button1\_Click(object sender, EventArgs e)

{

string mess = textBox1.Text; // получаем количество 64-битных чисел

if (mess == "" || Convert.ToInt32(mess) == 0) // проверка на корректное количество чисел

MessageBox.Show("Введите количество чисел больше 0.");

else

{

Cripto Cr = new Cripto(); // создаём основной класс

// очищаем все textBox при нажании на кнопку

textBox2.Clear();

textBox3.Clear();

textBox4.Clear();

textBox5.Clear();

textBox6.Clear();

textBox7.Clear();

textBox8.Clear();

textBox9.Clear();

textBox10.Clear();

textBox11.Clear();

textBox12.Clear();

ulong[] ansi = Cr.ANSI(Convert.ToInt32(mess)); // массив псевдослучайных чисел

int[] Seq = Cr.num\_convert(ansi); // подготовка последовательности преобразуем все числа

//в массив int значений(1, 0)

string[] out\_seq = new string[1]; // выходной массив для записи строки двоичных чисел в файл

for (int i = 0; i < Seq.Length; i++) // получение строки двоичных чисел

out\_seq[0] += Seq[i].ToString();

string binary = "";

for (int i = 0; i < ansi.Length; i++) // вывод чисел в hex и в двоичном формате

{

textBox2.Text += ansi[i].ToString("x") + " ";

binary += Convert.ToString((long)ansi[i], toBase: 2);

}

textBox3.Text = binary;

Cr.Write("out.txt", out\_seq); // запись в файл выходной двоичной последовательности

int[] Seq\_m = Cr.zero\_invert(Seq); // получение массива значений -1 и 1

double[] Freq\_res = Cr.freq\_test(Seq\_m); // частотный тест

// вывод шагов теста

textBox4.Text += "Сумма элементов " + Freq\_res[0].ToString() + Environment.NewLine;

textBox4.Text += "Статистика " + Freq\_res[1].ToString() + Environment.NewLine;

// если частотный тест не пройден, то нет смысла в других тестах

if (Freq\_res[2] == 1)

{

textBox7.Text = "Пройден ";

double[] same\_res = Cr.same\_bits\_test(Seq); // тест на последовательность одинаковых бит

// вывод шагов теста

textBox5.Text += "Частота появления единиц " + same\_res[0].ToString() + Environment.NewLine;

textBox5.Text += "Vn " + same\_res[1].ToString() + Environment.NewLine;

textBox5.Text += "Статистика " + same\_res[2].ToString() + Environment.NewLine;

// проверка прохождения теста

if (same\_res[3] == 1)

textBox8.Text = "Пройден";

else

textBox8.Text = "Не пройден";

double[] rand\_res = Cr.rand\_dev\_test(Seq\_m); // расширенный тест на произвольные отклонения

// вывод шагов теста

string rt = "";

for (int i = 0; i < rand\_res.Length - 2 - 2 \* 18; i++)

rt += rand\_res[i].ToString() + Environment.NewLine;

textBox6.Text = rt;

rt = "";

for (int i = rand\_res.Length - 2 - 2 \* 18; i < rand\_res.Length - 2 - 18; i++)

rt += rand\_res[i].ToString() + Environment.NewLine;

textBox10.Text = rt;

rt = "";

for (int i = rand\_res.Length - 2 - 18; i < rand\_res.Length - 2; i++)

rt += rand\_res[i].ToString() + Environment.NewLine;

textBox11.Text = rt;

textBox12.Text = rand\_res[rand\_res.Length - 2].ToString();

// проверка прохождения теста

if (rand\_res[rand\_res.Length - 1] == 1)

textBox9.Text = "Пройден";

else

textBox9.Text = "Не пройден";

}

else

{

textBox7.Text = "Не пройден ";

MessageBox.Show("Дальнейшие тесты не требуются, последовательность не случайная.");

}

}

}

1. **Выводы**

В ходе выполненной лабораторной было разработано программное средство, предназначенное для генерации псевдослучайной последовательности алгоритмом ANSI X9.17, а также для исследования полученной последовательности на случайность и равномерность.

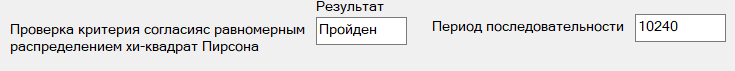
В ходе исследования последовательности длиной более 12000 бит, все тесты были успешно пройдены, исходя из чего можно сделать вывод, что последовательность можно считать достаточно случайной для использования в качестве криптографически безопасной псевдослучайной последовательности.

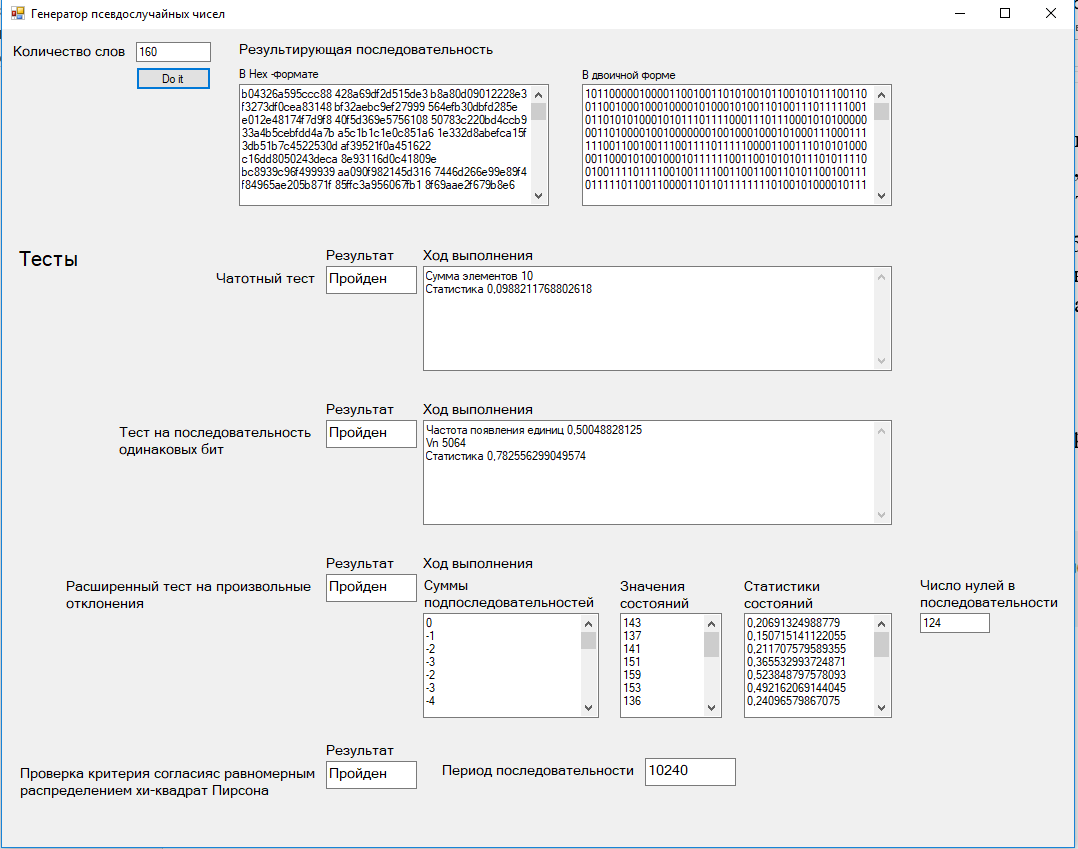
1. **Защита**

В качестве защиты было дано два задания:

1. Проверить критерий согласия с равномерным распределением хи-квадрат Пирсона.
2. Найти период последовательности.

Обе задачи выполнены:





Как видно из результатов, последовательность псевдослучайных чисел можно считать равномерной. Также период последовательности равен длине последовательности в битах. Это значит, что в последовательности не возможно выделить какие-то подпоследовательности определённой длины, а, следовательно, последовательность можно считать случайной.

Листинг кода для критерия согласия и поиска периода:

public double crit(double S, int r) // вычисление статистики

{

double coef = 1 / (Math.Pow(2, (r / 2)) \* SpecialFunctions.Gamma(r / 2));

double f = MathNet.Numerics.Integration.NewtonCotesTrapeziumRule.IntegrateAdaptive( x => Math.Pow(x, (r / 2) - 1) \* Math.Pow(Math.E, -x / 2), S, 10000000, 1e-5);

return coef \* f;

}

public double[] hi\_2(int[] Seq) // Критерий согласия хи-квадрат Пирсона

{

double[] res = new double[3];

int Max = 1;

int Min = -1;

double K = 5 \* Math.Log10(Seq.Length);

double[] P = new double[1];

double[] v = new double[(int)K];

double[] inter = new double[(int)K + 1];

double u = Min;

for (int i = 0; i < (int)K + 1; i++)

{

inter[i] = u;

u += ((Max - Min) / K);

}

for(int i = 0; i < Seq.Length; i++)

for(int j = 0; j < inter.Length - 1; j++)

{

if (Seq[i] >= inter[j] && Seq[i] <= inter[j + 1])

v[j]++;

}

for (int i = 0; i < v.Length; i++)

v[i] /= Seq.Length;

for(int i = 1; i < inter.Length; i++)

{

P[i - 1] = Rav(inter[i], 0, 1) - Rav(inter[i - 1], 0, 1);

Array.Resize(ref P, P.Length + 1);

}

Array.Resize(ref P, P.Length - 1);

double summ = 0;

for (int i = 0; i < (int)K; i++)

summ += Math.Pow(v[i] - P[i], 2) / P[i];

summ \*= Seq.Length;

res[0] = summ;

double S\_t = crit(summ, (int)K - 2);

res[1] = S\_t;

if (S\_t <= summ)

res[2] = 1;

else

res[2] = 0;

return res;

}

public int Peroid(int[] seq) //Вычисление периода

{

int per = 1;

int step = 0;

while (step + per != seq.Length)

{

if (seq[step] != seq[per + step])

{

++per;

step = 0;

}

else

{

++step;

}

}

return per;

}