**Липецкий государственный технический университет**

Университетский колледж

Кафедра автоматизированных систем управления

Лабораторная работа №6

по «Защита информации»

Алгоритм хеширования CRC

Студент Щеглов Д.A

Группа T-ИС-20

Руководитель Седых Ю.И

Липецк 2022г.

Оглавление

[Цель работы 3](#_Toc98097801)

[Задание 4](#_Toc98097802)

[Вариант 9 5](#_Toc98097803)

[Программный код 6](#_Toc98097804)

# **Цель работы**

Исследование алгоритмов построение CRC-кодов и их применения для контроля правильности передачи и хранения данных

# **Задание**

Реализуйте программу вычисляющую хэш-функцию CRC для сообщения с порождающим многочленом, заданным для вашего варианта. Переберите все числа с размером 1 байт и найдите все коллизии для вашей хэш-функции.

# Вариант 9

Алгоритм контрольного суммирования CRC расшифровывается, как циклический избыточный код (Cyclic redundancy code), и предназначается для контроля целостности данных. Он широко используется в проводных и беспроводных сетях, и в устройствах хранения данных, для проверки информации на подлинность и защиты от несанкционированного изменения. Он основывается на свойствах деления с остатком многочлена на многочлен. По сути, результатом контрольного суммирования CRC является остаток от деления многочлена, соответствующего исходным данным, на порождающий многочлен фиксированной длины.

Очевидно, что количество различных остатков от деления многочлена на многочлен меньше, чем количество различных исходных многочленов. Таким образом, контрольное суммирование CRC может однозначно дать ответ, что два массива данных отличаются друг от друга, если отличаются их контрольные суммы. Но, если две контрольные суммы совпали, нельзя однозначно утверждать, что для их формирования использовался один и тот же исходный массив данных.

В зависимости от вида порождающего многочлена и его длины, изменяется вероятность совпадения контрольных сумм для различных исходных данных и время контрольного суммирования. Наиболее популярными являются алгоритмы CRC, работающие с порождающими многочленами: восьмой (CRC-8), шестнадцатой (CRC - 16) и тридцать второй (CRC – 32) степени.

Вариант 9

G(x) = 1 + x2 + x6  - полином

x6 \* 1 + x5 \* 0 + x4 \* 0 + x3 \* 0 + x2 \* 1 + x1 \* 0 + x0 \*1 = 1000101

Реализация хеширования CRC

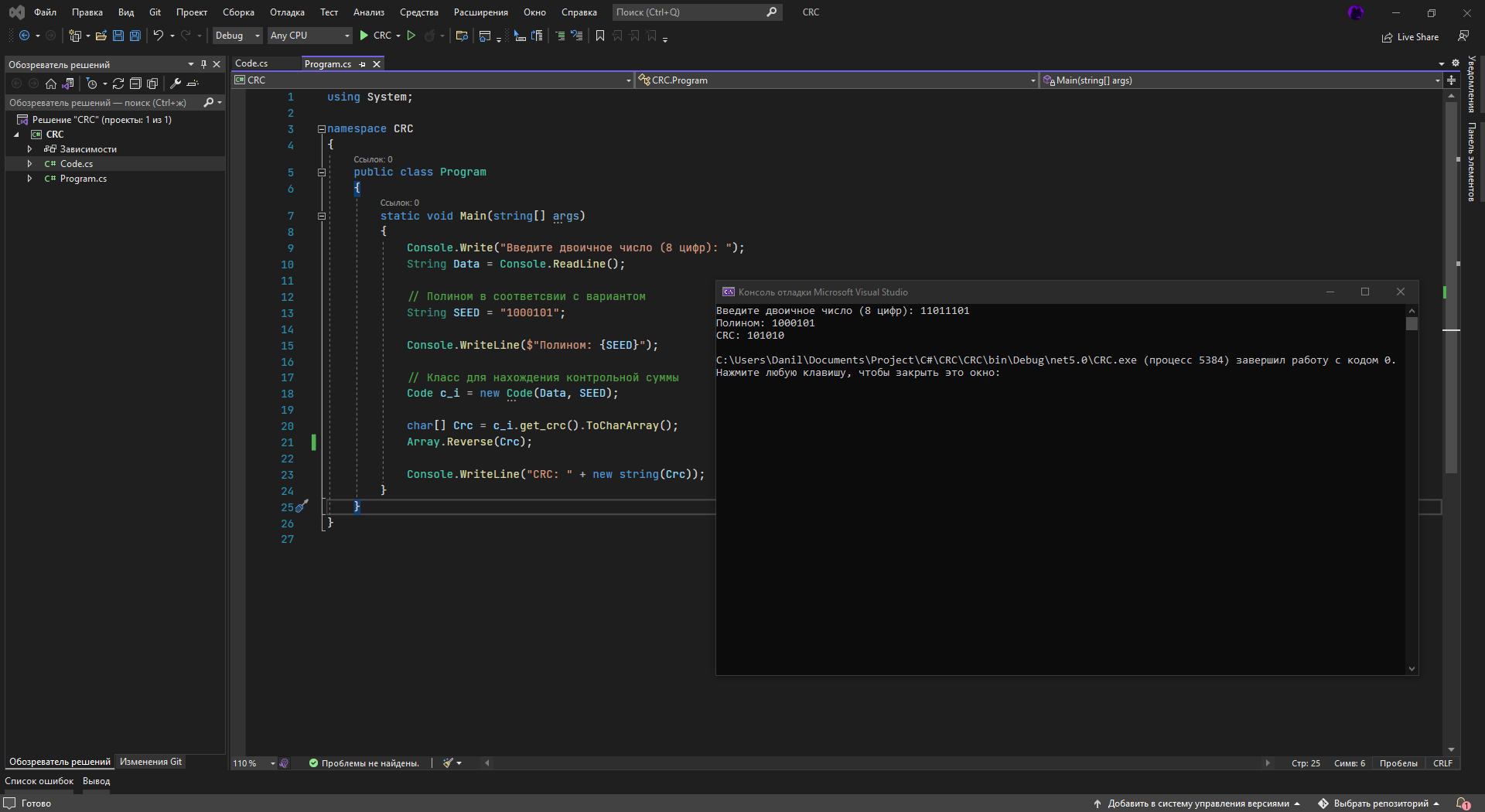


Рисунок 1.

# Программный код

using System;

namespace CRC

{

public class Code

{

private String BitStream;

// Дефолтное значение если полином не выбран

private String SEED = new String("000001");

private String crc = new String("");

private int[] c = new int[6];

private int[] ch = new int[6];

private int[] d = new int[1];

private String calc()

{

for (int i = 0; i < 6; ++i)

{

this.c[6 - 1 - i] = (int)(this.SEED[i]) - (int)('0');

}

for (int i = 0; i < this.BitStream.Length; i = i + 1)

{

for (int j = 0; j < 1;

j = j + 1)

{

this.d[1 - 1 - j] = (int)(this.BitStream[i + j]) - (int)('0');

}

for (int k = 0; k < 6;

k = k + 1)

{

this.ch[k] = this.c[k];

}

this.c[5] = this.ch[4];

this.c[4] = this.ch[3];

this.c[3] = this.ch[2];

this.c[2] = this.ch[1] ^ this.ch[5] ^ this.d[0];

this.c[1] = this.ch[0];

this.c[0] = this.ch[5] ^ this.d[0];

}

for (int i = 0; i < 6; ++i)

{

this.crc += Convert.ToString(this.c[6 - 1 - i]);

}

return this.crc;

}

public Code(String BitStream\_in, String SEED\_in)

{

this.SEED = SEED\_in;

this.BitStream = BitStream\_in;

this.calc();

}

public Code(String BitStream\_in)

{

this.BitStream = BitStream\_in;

this.calc();

}

public int set\_data(String BitStream\_in)

{

this.BitStream = BitStream\_in;

return 0;

}

public int set\_seed(String SEED\_in)

{

this.SEED = SEED\_in;

return 0;

}

public String get\_crc()

{

return this.crc;

}

}

}