Лабораторная работа №1
Шкиндер Антон
3 курс 9 группа

Описание стандартного алгоритма:

1) Отправитель и получатель определяют образующий полином G(x) (старший и младший биты равны нулю). r — степень полинома.
2) Пусть M(x) - это кадр, содержащий m бит, тогда строим полином $M_1(x) = x^r M(x)$ — кадр дополненный справа r нулевыми байтами
3) Вычисляем $R(x) = M_1(x) \bmod G(x)$, $\deg(R(x)) <= r$
4) Вычисляем $T(x) = M_1(x) - R(x)$, который будет соответствовать кадру
5) Получатель вычисляет $R_1(x) = T(x) \bmod G(x)$, если $R_1(x)=0$, то кадр передан без ошибок, иначе ошибки существуют

Для наглядности алгоритм можно добавить примером из wikipedia:
Слева — “сторона отправителя”, справа — “сторона получателя”

```
11010011101100 000
1011
01100011101100 000
 1011
00111011101100 000
  1011
00010111101100 000
   1011
00000001101100 000
       1011
00000000110100 000
        1011
00000000011000 000
         1011
00000000001110 000
          1011
00000000000101 000
           101 1
-----------------
00000000000000 100
```

```
11010011101100 100
1011
01100011101100 100
 1011
00111011101100 100

......

00000000001110 100
          1011
00000000000101 100
           101 1
------------------
00000000000000 000
```

Реализация алгоритма выше будет заключаться в простых манипуляциях с реализаций полинома (на строках или же массивах байт), где первый полином — исходные данные, а второй — G(x). Но такая реализация будет иметь очень малую скорость работы (не говоря уже о сложностях связанных с самой реализацией полиномов над полем), поэтому я использовал вариант с таблицей. Таблица по своей сути — это всевозможные результаты операции XOR между полиномом G(x) и содержимого регистра (числа принадлежащие [0, x100) и сдвигаемые влево). Она содержит 256 чисел, которые являются всевозможными метками. Приводить алгоритм не вижу смысла, т.к. это будет просто перепечатка его с этого сайта. Мою реализацию можно увидеть в файле “crc.hpp”, для вычисления я использовал уже написанные в коде таблицы (hardcode).

Для CRC4 я использовал:
- полином = 0x3
- init = 0x0
- XorOut = 0x0
- Check(CRC4(“123456789”)) = 0x8

Для CRC8 я использовал:
- полином = 0x31
- init = 0xFF
- XorOut = 0x0
- Check(CRC4(“123456789”)) = 0xF7

Для CRC16 я использовал:
- полином = 0x1021
- init = 0xFFFF
- XorOut = 0x0
- Check(CRC4(“123456789”)) = 0x29B1

Для CRC32 я использовал (reversed полином, все отличие только в направлении сдвигов при составлении таблицы и нахождении значения в ней: сдвиги делаются вправо, а не влево)
- полином = 0xEDB88320
- init = 0xFFFFFFFF
- XorOut = 0xFFFFFFFF
- Check(CRC4(“123456789”)) = 0xCBF43926

Ответы на вопросы:

1) Найти вероятность обнаружения ошибки

Если длина слова меньше или равно длине порождающего многочлена, то вероятность обнаружения ошибки равна 1, в других случаях $1 - (1 / 2^n)$, где n — степень порождающего многочлена.

2) Как выбирать порождающий многочлен фиксированной степени? На что влияет такой выбор?

Этот вопрос далеко нетривиален и наилучшем выходом будет использование проверенных и даже стандартизированных алгоритмов. Так же G(x) - непроходимый многочлен, старший и младшие биты которого равны 1. Так же необходимо, что бы число бит в G(x) было четно для того, что бы отловить ошибки в нечетном количестве бит (более подробно про это написано здесь и не хотелось бы просто перепечатывать оттуда текст).

Пример: $G(x) = (x+1)p(x)$, где p(x) - непроходимый многочлен n-1 степени, позволяющий обнаруживать одиночные, двойные, тройные и любое нечетное кол-во ошибок.

Выбор многочлена в первую очередь влияет на обнаружение ошибок различного типа (как и сказано выше).

3) Как найти порождающий многочлен, зная k слов с контрольными метками?

$T_i = M_i G(x) + R_i$, где T_i - кодовое слово, R_i - кодовые метки, а i идет от 0 и до k

Следовательно находим все $M_i G(x) = T_i - R_i$ и находим их НОД. Если НОД не является непроходимым многочленом, то разбиваем его на непроходимые и проверяем все линейные комбинации через алгоритм, та линейная комбинация, из которой получим метки и является G(x).

Реализация CRC4, CRC8, CRC16, CRC32 (C++14):
Все файлы можно найти по ссылке: https://gitlab.com/Shkinder/crc

```
// crc.hpp
#pragma once

#include <vector>
#include <sstream>
#include <bitset>
#include <type_traits>

class unit_tests;

using crc4Type = uint8_t;
using crc8Type = uint8_t;
using crc16Type = uint16_t;
using crc32Type = uint32_t;

template< class crcType, crcType polinomal >
class CRC{
public:
    static_assert( std::is_same< crcType, crc4Type>() ||
std::is_same< crcType, crc8Type>() ||
                   std::is_same< crcType, crc16Type>() ||
std::is_same< crcType, crc32Type>(), "Type not in list of crc
types!" );

    CRC(): poly( polinomal ) { };

    virtual crcType encode( std::istream &is ) = 0;

    crcType get_crc_code() const { return crc_code; }

    crcType get_crc_poly() const { return poly; }

    friend class unit_tests;

protected:

    crcType crc_code;

    const crcType poly;

    std::vector< crcType > generate_crc_table() {
        std::vector< crcType > crc_tab( 256 );
        crcType _crc;

        for( uint32_t x = 0; x < 0x100; ++x ) {
```

```
            _crc = typeid( crcType ) != typeid( crc32Type ) ?  x
<< ( sizeof( crcType ) * 8 - 8 ) : x;
            for ( uint32_t y = 0; y < 8; y++ ) {
                if ( typeid( crcType ) != typeid( crc32Type ) )
                    _crc = ( _crc & ( 1 << ( sizeof( crcType ) * 8
- 1 ) ) ) ? ( ( _crc << 1 ) ^ poly ) : ( _crc << 1 );
                else
                    _crc = _crc & 1 ? ( _crc >> 1 ) ^ 0xEDB88320UL
: _crc >> 1;
            }
            crc_tab[ x ] = _crc;
        }

        return crc_tab;
    }

    bool read_stream( std::istream& is, uint8_t& value ) {
        value = 0;
        value |= is.get();
        return !is.eof();
    }
};

template< class crcType, crcType polinomal >
std::ostream& operator<<( std::ostream &os, const CRC< crcType,
polinomal > &crc ){
    crcType crc_code = crc.get_crc_code();
    os << "Binary value: 0b" << std::bitset< sizeof( crc_code ) *
8 >( crc_code ) << std::endl;
    os << "Hex value: 0x" << std::hex << (int)crc_code;
    return os;
}

template< class crcType, crcType polinomal >
std::istream& operator>>( std::istream &os, CRC< crcType,
polinomal > &crc ){
    crc.encode( os );
    return os;
}


// Poly: 0x3
//       x^4 + x + 1
class CRC4 : public CRC< crc4Type, 0x3 > {

public:

    CRC4() { };

    CRC4( std::istream &is ) { encode( is ); }

    crc4Type encode( std::istream &is ) {
```

```
        assert( crc_table.size() );
        crc_code = 0;
        uint8_t buff;

        while( read_stream( is, buff ) ) {
            crc_code = crc_table[ crc_code ^ buff  ];
        }

        return crc_code;
    }

    friend class unit_tests;

private:

    const std::vector< crc4Type > crc_table = {
            0x0,  0x3,  0x6,  0x5,  0xc,  0xf,  0xa,  0x9, 0x18,
            0x1b, 0x1e, 0x1d, 0x14, 0x17, 0x12, 0x11, 0x30,
            0x33, 0x36, 0x35, 0x3c, 0x3f, 0x3a, 0x39, 0x28,
            0x2b, 0x2e, 0x2d, 0x24, 0x27, 0x22, 0x21, 0x60,
            0x63, 0x66, 0x65, 0x6c, 0x6f, 0x6a, 0x69, 0x78,
            0x7b, 0x7e, 0x7d, 0x74, 0x77, 0x72, 0x71, 0x50,
            0x53, 0x56, 0x55, 0x5c, 0x5f, 0x5a, 0x59, 0x48,
            0x4b, 0x4e, 0x4d, 0x44, 0x47, 0x42, 0x41, 0xc0,
            0xc3, 0xc6, 0xc5, 0xcc, 0xcf, 0xca, 0xc9, 0xd8,
            0xdb, 0xde, 0xdd, 0xd4, 0xd7, 0xd2, 0xd1, 0xf0,
            0xf3, 0xf6, 0xf5, 0xfc, 0xff, 0xfa, 0xf9, 0xe8,
            0xeb, 0xee, 0xed, 0xe4, 0xe7, 0xe2, 0xe1, 0xa0,
            0xa3, 0xa6, 0xa5, 0xac, 0xaf, 0xaa, 0xa9, 0xb8,
            0xbb, 0xbe, 0xbd, 0xb4, 0xb7, 0xb2, 0xb1, 0x90,
            0x93, 0x96, 0x95, 0x9c, 0x9f, 0x9a, 0x99, 0x88,
            0x8b, 0x8e, 0x8d, 0x84, 0x87, 0x82, 0x81, 0x83,
            0x80, 0x85, 0x86, 0x8f, 0x8c, 0x89, 0x8a, 0x9b,
            0x98, 0x9d, 0x9e, 0x97, 0x94, 0x91, 0x92, 0xb3,
            0xb0, 0xb5, 0xb6, 0xbf, 0xbc, 0xb9, 0xba, 0xab,
            0xa8, 0xad, 0xae, 0xa7, 0xa4, 0xa1, 0xa2, 0xe3,
            0xe0, 0xe5, 0xe6, 0xef, 0xec, 0xe9, 0xea, 0xfb,
            0xf8, 0xfd, 0xfe, 0xf7, 0xf4, 0xf1, 0xf2, 0xd3,
            0xd0, 0xd5, 0xd6, 0xdf, 0xdc, 0xd9, 0xda, 0xcb,
            0xc8, 0xcd, 0xce, 0xc7, 0xc4, 0xc1, 0xc2, 0x43,
            0x40, 0x45, 0x46, 0x4f, 0x4c, 0x49, 0x4a, 0x5b,
            0x58, 0x5d, 0x5e, 0x57, 0x54, 0x51, 0x52, 0x73,
            0x70, 0x75, 0x76, 0x7f, 0x7c, 0x79, 0x7a, 0x6b,
            0x68, 0x6d, 0x6e, 0x67, 0x64, 0x61, 0x62, 0x23,
            0x20, 0x25, 0x26, 0x2f, 0x2c, 0x29, 0x2a, 0x3b,
            0x38, 0x3d, 0x3e, 0x37, 0x34, 0x31, 0x32, 0x13,
            0x10, 0x15, 0x16, 0x1f, 0x1c, 0x19, 0x1a, 0xb,
            0x8,  0xd,  0xe,  0x7,  0x4,  0x1,  0x2
        };

};
```

```
// Poly: 0x31
//       x^8 + x^5 + x^4 + 1
class CRC8 : public CRC< crc4Type, 0x31 > {

public:

    CRC8() { };

    CRC8( std::istream &is ) { encode( is ); }

    crc4Type encode( std::istream &is ) {
        assert( crc_table.size() );
        crc_code = ~0;
        uint8_t buff;

        while( read_stream( is, buff ) ) {
            crc_code = crc_table[ crc_code ^ buff  ];
        }

        return crc_code;
    }

    friend class unit_tests;

private:

    const std::vector< crc8Type > crc_table = {
        0x00, 0x31, 0x62, 0x53, 0xC4, 0xF5, 0xA6, 0x97,
        0xB9, 0x88, 0xDB, 0xEA, 0x7D, 0x4C, 0x1F, 0x2E,
        0x43, 0x72, 0x21, 0x10, 0x87, 0xB6, 0xE5, 0xD4,
        0xFA, 0xCB, 0x98, 0xA9, 0x3E, 0x0F, 0x5C, 0x6D,
        0x86, 0xB7, 0xE4, 0xD5, 0x42, 0x73, 0x20, 0x11,
        0x3F, 0x0E, 0x5D, 0x6C, 0xFB, 0xCA, 0x99, 0xA8,
        0xC5, 0xF4, 0xA7, 0x96, 0x01, 0x30, 0x63, 0x52,
        0x7C, 0x4D, 0x1E, 0x2F, 0xB8, 0x89, 0xDA, 0xEB,
        0x3D, 0x0C, 0x5F, 0x6E, 0xF9, 0xC8, 0x9B, 0xAA,
        0x84, 0xB5, 0xE6, 0xD7, 0x40, 0x71, 0x22, 0x13,
        0x7E, 0x4F, 0x1C, 0x2D, 0xBA, 0x8B, 0xD8, 0xE9,
        0xC7, 0xF6, 0xA5, 0x94, 0x03, 0x32, 0x61, 0x50,
        0xBB, 0x8A, 0xD9, 0xE8, 0x7F, 0x4E, 0x1D, 0x2C,
        0x02, 0x33, 0x60, 0x51, 0xC6, 0xF7, 0xA4, 0x95,
        0xF8, 0xC9, 0x9A, 0xAB, 0x3C, 0x0D, 0x5E, 0x6F,
        0x41, 0x70, 0x23, 0x12, 0x85, 0xB4, 0xE7, 0xD6,
        0x7A, 0x4B, 0x18, 0x29, 0xBE, 0x8F, 0xDC, 0xED,
        0xC3, 0xF2, 0xA1, 0x90, 0x07, 0x36, 0x65, 0x54,
        0x39, 0x08, 0x5B, 0x6A, 0xFD, 0xCC, 0x9F, 0xAE,
        0x80, 0xB1, 0xE2, 0xD3, 0x44, 0x75, 0x26, 0x17,
        0xFC, 0xCD, 0x9E, 0xAF, 0x38, 0x09, 0x5A, 0x6B,
        0x45, 0x74, 0x27, 0x16, 0x81, 0xB0, 0xE3, 0xD2,
        0xBF, 0x8E, 0xDD, 0xEC, 0x7B, 0x4A, 0x19, 0x28,
        0x06, 0x37, 0x64, 0x55, 0xC2, 0xF3, 0xA0, 0x91,
        0x47, 0x76, 0x25, 0x14, 0x83, 0xB2, 0xE1, 0xD0,
```

```
        0xFE, 0xCF, 0x9C, 0xAD, 0x3A, 0x0B, 0x58, 0x69,
        0x04, 0x35, 0x66, 0x57, 0xC0, 0xF1, 0xA2, 0x93,
        0xBD, 0x8C, 0xDF, 0xEE, 0x79, 0x48, 0x1B, 0x2A,
        0xC1, 0xF0, 0xA3, 0x92, 0x05, 0x34, 0x67, 0x56,
        0x78, 0x49, 0x1A, 0x2B, 0xBC, 0x8D, 0xDE, 0xEF,
        0x82, 0xB3, 0xE0, 0xD1, 0x46, 0x77, 0x24, 0x15,
        0x3B, 0x0A, 0x59, 0x68, 0xFF, 0xCE, 0x9D, 0xAC
    };

};

// Poly: 0x1021
//       x^16 + x^12 + x^5 + 1
class CRC16 : public CRC< crc16Type, 0x1021 > {

public:

    CRC16() { };

    CRC16( std::istream &is ) { encode( is ); }

    crc16Type encode( std::istream &is ) {
        assert( crc_table.size() );
        crc_code = ~0;
        uint8_t buff;

        while( read_stream( is, buff ) ) {
            crc_code = ( crc_code << 8 ) ^ crc_table[ ( crc_code
>> 8 ) ^ buff ];
        }

        return crc_code;
    }

    friend class unit_tests;

private:

    std::vector< crc16Type > crc_table = {
        0x0000, 0x1021, 0x2042, 0x3063, 0x4084, 0x50A5, 0x60C6,
0x70E7,
        0x8108, 0x9129, 0xA14A, 0xB16B, 0xC18C, 0xD1AD, 0xE1CE,
0xF1EF,
        0x1231, 0x0210, 0x3273, 0x2252, 0x52B5, 0x4294, 0x72F7,
0x62D6,
        0x9339, 0x8318, 0xB37B, 0xA35A, 0xD3BD, 0xC39C, 0xF3FF,
0xE3DE,
        0x2462, 0x3443, 0x0420, 0x1401, 0x64E6, 0x74C7, 0x44A4,
0x5485,
        0xA56A, 0xB54B, 0x8528, 0x9509, 0xE5EE, 0xF5CF, 0xC5AC,
0xD58D,
```

```
        0x3653, 0x2672, 0x1611, 0x0630, 0x76D7, 0x66F6, 0x5695,
0x46B4,
        0xB75B, 0xA77A, 0x9719, 0x8738, 0xF7DF, 0xE7FE, 0xD79D,
0xC7BC,
        0x48C4, 0x58E5, 0x6886, 0x78A7, 0x0840, 0x1861, 0x2802,
0x3823,
        0xC9CC, 0xD9ED, 0xE98E, 0xF9AF, 0x8948, 0x9969, 0xA90A,
0xB92B,
        0x5AF5, 0x4AD4, 0x7AB7, 0x6A96, 0x1A71, 0x0A50, 0x3A33,
0x2A12,
        0xDBFD, 0xCBDC, 0xFBBF, 0xEB9E, 0x9B79, 0x8B58, 0xBB3B,
0xAB1A,
        0x6CA6, 0x7C87, 0x4CE4, 0x5CC5, 0x2C22, 0x3C03, 0x0C60,
0x1C41,
        0xEDAE, 0xFD8F, 0xCDEC, 0xDDCD, 0xAD2A, 0xBD0B, 0x8D68,
0x9D49,
        0x7E97, 0x6EB6, 0x5ED5, 0x4EF4, 0x3E13, 0x2E32, 0x1E51,
0x0E70,
        0xFF9F, 0xEFBE, 0xDFDD, 0xCFFC, 0xBF1B, 0xAF3A, 0x9F59,
0x8F78,
        0x9188, 0x81A9, 0xB1CA, 0xA1EB, 0xD10C, 0xC12D, 0xF14E,
0xE16F,
        0x1080, 0x00A1, 0x30C2, 0x20E3, 0x5004, 0x4025, 0x7046,
0x6067,
        0x83B9, 0x9398, 0xA3FB, 0xB3DA, 0xC33D, 0xD31C, 0xE37F,
0xF35E,
        0x02B1, 0x1290, 0x22F3, 0x32D2, 0x4235, 0x5214, 0x6277,
0x7256,
        0xB5EA, 0xA5CB, 0x95A8, 0x8589, 0xF56E, 0xE54F, 0xD52C,
0xC50D,
        0x34E2, 0x24C3, 0x14A0, 0x0481, 0x7466, 0x6447, 0x5424,
0x4405,
        0xA7DB, 0xB7FA, 0x8799, 0x97B8, 0xE75F, 0xF77E, 0xC71D,
0xD73C,
        0x26D3, 0x36F2, 0x0691, 0x16B0, 0x6657, 0x7676, 0x4615,
0x5634,
        0xD94C, 0xC96D, 0xF90E, 0xE92F, 0x99C8, 0x89E9, 0xB98A,
0xA9AB,
        0x5844, 0x4865, 0x7806, 0x6827, 0x18C0, 0x08E1, 0x3882,
0x28A3,
        0xCB7D, 0xDB5C, 0xEB3F, 0xFB1E, 0x8BF9, 0x9BD8, 0xABBB,
0xBB9A,
        0x4A75, 0x5A54, 0x6A37, 0x7A16, 0x0AF1, 0x1AD0, 0x2AB3,
0x3A92,
        0xFD2E, 0xED0F, 0xDD6C, 0xCD4D, 0xBDAA, 0xAD8B, 0x9DE8,
0x8DC9,
        0x7C26, 0x6C07, 0x5C64, 0x4C45, 0x3CA2, 0x2C83, 0x1CE0,
0x0CC1,
        0xEF1F, 0xFF3E, 0xCF5D, 0xDF7C, 0xAF9B, 0xBFBA, 0x8FD9,
0x9FF8,
        0x6E17, 0x7E36, 0x4E55, 0x5E74, 0x2E93, 0x3EB2, 0x0ED1,
0x1EF0
```

```
    };

};

// Poly: 0x04C11DB7
// Revert: true
// Revert poly: 0xEDB88320
//       x^32 + x^26 + x^23 + x^22 + x^16 + x^12 + x^11 + x^10 +
x^8 + x^7 + x^5 + x^4 + x^2 + x + 1
class CRC32 : public CRC< crc32Type, 0xEDB88320 > {

public:

    CRC32() { };

    CRC32( std::istream &is ) { encode( is ); }

    crc32Type encode( std::istream &is ) {
        assert( crc_table.size() );
        crc_code = ~0;
        uint8_t buff;

        while( read_stream( is, buff ) ) {
            crc_code = crc_table[ ( crc_code ^ buff ) & 0xFF ] ^ (
crc_code >> 8 );
        }

        crc_code = ~crc_code;

        return crc_code;
    }

    friend class unit_tests;

private:

    std::vector< crc32Type > crc_table = {
        0,          1996959894, 3993919788, 2567524794, 124634137,
1886057615, 3915621685, 2657392035,
        249268274,  2044508324, 3772115230, 2547177864, 162941995,
2125561021, 3887607047,
        2428444049, 498536548,  1789927666, 4089016648,
2227061214, 450548861,  1843258603,
        4107580753, 2211677639, 325883990,  1684777152,
4251122042, 2321926636, 335633487,
        1661365465, 4195302755, 2366115317, 997073096,
1281953886, 3579855332, 2724688242,
        1006888145, 1258607687, 3524101629, 2768942443, 901097722,
1119000684, 3686517206,
        2898065728, 853044451,  1172266101, 3705015759,
2882616665, 651767980,  1373503546,
```

3369554304, 3218104598, 565507253, 1454621731,
3485111705, 3099436303, 671266974,
1594198024, 3322730930, 2970347812, 795835527,
1483230225, 3244367275, 3060149565,
1994146192, 31158534, 2563907772, 4023717930,
1907459465, 112637215, 2680153253,
3904427059, 2013776290, 251722036, 2517215374,
3775830040, 2137656763, 141376813,
2439277719, 3865271297, 1802195444, 476864866,
2238001368, 4066508878, 1812370925,
453092731, 2181625025, 4111451223, 1706088902, 314042704,
2344532202, 4240017532,
1658658271, 366619977, 2362670323, 4224994405,
1303535960, 984961486, 2747007092,
3569037538, 1256170817, 1037604311, 2765210733,
3554079995, 1131014506, 879679996,
2909243462, 3663771856, 1141124467, 855842277,
2852801631, 3708648649, 1342533948,
654459306, 3188396048, 3373015174, 1466479909, 544179635,
3110523913, 3462522015,
1591671054, 702138776, 2966460450, 3352799412,
1504918807, 783551873, 3082640443,
3233442989, 3988292384, 2596254646, 62317068,
1957810842, 3939845945, 2647816111,
81470997, 1943803523, 3814918930, 2489596804, 225274430,
2053790376, 3826175755,
2466906013, 167816743, 2097651377, 4027552580,
2265490386, 503444072, 1762050814,
4150417245, 2154129355, 426522225, 1852507879,
4275313526, 2312317920, 282753626,
1742555852, 4189708143, 2394877945, 397917763,
1622183637, 3604390888, 2714866558,
953729732, 1340076626, 3518719985, 2797360999,
1068828381, 1219638859, 3624741850,
2936675148, 906185462, 1090812512, 3747672003,
2825379669, 829329135, 1181335161,
3412177804, 3160834842, 628085408, 1382605366,
3423369109, 3138078467, 570562233,
1426400815, 3317316542, 2998733608, 733239954,
1555261956, 3268935591, 3050360625,
752459403, 1541320221, 2607071920, 3965973030,
1969922972, 40735498, 2617837225,
3943577151, 1913087877, 83908371, 2512341634,
3803740692, 2075208622, 213261112,
2463272603, 3855990285, 2094854071, 198958881,
2262029012, 4057260610, 1759359992,
534414190, 2176718541, 4139329115, 1873836001, 414664567,
2282248934, 4279200368,
1711684554, 285281116, 2405801727, 4167216745,
1634467795, 376229701, 2685067896,
3608007406, 1308918612, 956543938, 2808555105,
3495958263, 1231636301, 1047427035,

```
        2932959818, 3654703836, 1088359270, 936918000,
2847714899, 3736837829, 1202900863,
        817233897,  3183342108, 3401237130, 1404277552, 615818150,
3134207493, 3453421203,
        1423857449, 601450431,  3009837614, 3294710456,
1567103746, 711928724,  3020668471,
        3272380065, 1510334235, 755167117
    };

};

// test.cpp
#include "crc.hpp"
#include <iostream>
#include <sstream>
using namespace std;

#define MESSAGE( x ) \
    cout << x << endl;

#define CHECK( x, y ) \
    if( ! ( x ) ) { \
        cout << "Failed! " << y << endl; \
        assert( x ); \
    } \

class unit_tests {
public:
    static void generator_test_4() {
        CRC4 crc;
        auto crc_table = crc.generate_crc_table();

        for( uint32_t i = 0; i < 0x100; ++i ) {
            CHECK( crc_table[i] == crc.crc_table[i],
"generator_test_4()" );
        }

        MESSAGE( "generator_test_4() passed" );
    }

    static void generator_test_8() {
        CRC8 crc;
        auto crc_table = crc.generate_crc_table();

        for( uint32_t i = 0; i < 0x100; ++i ) {
            CHECK( crc_table[i] == crc.crc_table[i],
"generator_test_8()" );
        }

        MESSAGE( "generator_test_8() passed" );
    }
```

```
    static void generator_test_16() {
        CRC16 crc;
        auto crc_table = crc.generate_crc_table();

        for( uint32_t i = 0; i < 0x100; ++i ) {
            CHECK( crc_table[i] == crc.crc_table[i],
"generator_test_16()" );
        }

        MESSAGE( "generator_test_16() passed" );
    }

    static void generator_test_32() {
        CRC32 crc;
        auto crc_table = crc.generate_crc_table();

        for( uint32_t i = 0; i < 0x100; ++i ) {
            CHECK( crc_table[i] == crc.crc_table[i],
"generator_test_32()" );
        }

        MESSAGE( "generator_test_32() passed" );
    }

    static void base_crc_test_4() {
        stringstream ss("123456789");
        CRC4 crc;
        ss >> crc;
        std::cout << crc << std::endl;
        CHECK( crc.get_crc_poly() == 0x3, "CRC4 poly != 0x3");
        CHECK( crc.get_crc_code() == 0x8, "CRC4(123456789) != 0x8"
);

        MESSAGE( "base_crc_test_4() passed" );
    }

    static void base_crc_test_8() {
        stringstream ss("123456789");
        CRC8 crc;
        ss >> crc;
        CHECK( crc.get_crc_poly() == 0x31, "CRC8 poly != 0x31");
        CHECK( crc.get_crc_code() == 0xF7, "CRC8(123456789) !=
0xF7" );

        MESSAGE( "base_crc_test_8() passed" );
    }

    static void base_crc_test_16() {
        stringstream ss("123456789");
        CRC16 crc;
        ss >> crc;
```

```
        CHECK( crc.get_crc_poly() == 0x1021, "CRC16 poly !=
0x1021");
        CHECK( crc.get_crc_code() == 0x29B1, "CRC16(123456789) !=
0x29B1" );

        MESSAGE( "base_crc_test_16() passed" );
    }

    static void base_crc_test_32() {
        stringstream ss("123456789");
        CRC32 crc;
        ss >> crc;
        CHECK( crc.get_crc_poly() == 0xEDB88320, "CRC32 poly !=
0xEDB88320");
        CHECK( crc.get_crc_code() == 0xCBF43926, "CRC32(123456789)
!= 0xCBF43926" );

        MESSAGE( "base_crc_test_32() passed" );
    }
};

int main(){
    cout << "Tests strated!" << endl << endl;

    cout << "Table generator tests" << endl << endl;

    unit_tests::generator_test_4();

    unit_tests::generator_test_8();

    unit_tests::generator_test_16();

    unit_tests::generator_test_32();

    cout << endl << "Base CRC tests (check \"123456789\")" << endl
<< endl;

    unit_tests::base_crc_test_4();

    unit_tests::base_crc_test_8();

    unit_tests::base_crc_test_16();

    unit_tests::base_crc_test_32();

    cout << endl << "Tests passed!" << endl << endl;
}

// source.cpp
// main program
#include "crc.hpp"
#include <iostream>
```

```
#include <fstream>
#include <utility>

void print_help_message(){
    std::cout << "CRC calculator, available: crc4/crc8/crc16/
crc32." << std::endl;
    std::cout << "Usage:" << std::endl;
    std::cout << "\t-t [type]  -- type = crc4/crc8/crc16/crc32
(used crc8 by default)" << std::endl;
    std::cout << "\t-h         -- print this message" <<
std::endl;
    std::cout << "[arg1] -- path to file (no default value)" <<
std::endl;
}

uint8_t get_type( std::string arg ){
    if( arg == "crc4" ){
        return 4;
    } else if( arg == "crc8" ){
        return 8;
    } else if( arg =="crc16" ){
        return 16;
    } else if( arg == "crc32" ){
        return 32;
    } else {
        return 0;
    }
}

std::pair<uint8_t, std::ifstream> parse_agrs( int argc, char**
argv ){
    if( argc != 4 && argc != 2 ){
        std::cout << "Invalid args!" << std::endl;
        print_help_message();
        exit(1);
    }

    if( argc == 2 ){
        if( argv[1][1] == 'h' ) {
            print_help_message();
            exit(0);
        } else{
            return std::make_pair( 8, std::ifstream( argv[1] ) );
        }
    } else {
        uint8_t bits = 0;
        std::string arg1( argv[1] ), arg2( argv[2] ),
arg3( argv[3] );
        if( arg1 == "-t" ){
            bits = get_type( arg2 );
            return make_pair( bits, std::ifstream( arg3 ) );
        } else if( arg2 == "-t" ) {
```

```
            bits = get_type( arg3 );
            return make_pair( bits, std::ifstream( arg1 ) );
        } else {
            std::cout << "Invalid args!" << std::endl;
            print_help_message();
            exit(1);
        }
    }

}

template< class crcType, crcType polinomal >
void print_result( const CRC< crcType, polinomal > &crc, uint8_t
bits ){
    std::cout << "Result CRC" << bits << std::endl << crc <<
std::endl;
}

int main( int argc, char** argv ){
    auto pair_of_crc = parse_agrs( argc, argv );
    if( !pair_of_crc.second.is_open() ){
        std::cout << "Invalid path to file! (or invalid args, use
-h to see help" << std::endl;
    }
    switch( pair_of_crc.first ) {
        case 0: print_help_message(); break;
        case 4:  { CRC4  crc4 ( pair_of_crc.second );
print_result( crc4,  4  ); break; }
        case 8:  { CRC8  crc8 ( pair_of_crc.second );
print_result( crc8,  8  ); break; }
        case 16: { CRC16 crc16( pair_of_crc.second );
print_result( crc16, 16 ); break; }
        case 32: { CRC32 crc32( pair_of_crc.second );
print_result( crc32, 32 ); break; }
    }
}

// CMakeLists.txt
cmake_minimum_required(VERSION 3.12)
set( CMAKE_CXX_STANDARD 14 )

add_executable( crc source.cpp crc.hpp )

add_executable( test test.cpp )
```