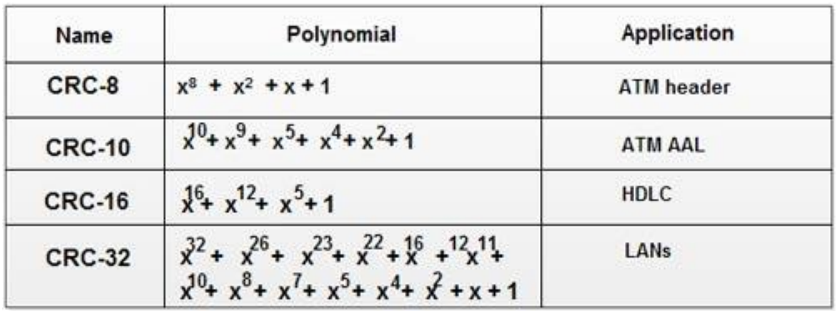
# Experiment 2: Error Detection using CRC-CCITT

**Aim:** To apply CRC (CCITT Polynomial) for error detection

**Objective:** After carrying out this experiment, students will be able to:

- Apply CRC CCITT to develop codes for error detection
- Analyse how this CRC is able to detect bit errors irrespective of their length and position in the data

**Problem statement:** You are required to write a program that uses CRC to detect burst errors in transmitted data. Initially, write the program using the CRC example you studied in class. Your final program should ask the user to input data and choose a generator polynomial from the list given in the figure below. Your program is required to calculate the checksum and the transmitted data. Subsequently, the user enters the received data. Applying the same generator polynomial on the received data should result in a remainder of 0.



| Name | Polynomial | Application |
|---|---|---|
| CRC-8 | $x^8 + x^2 + x + 1$ | ATM header |
| CRC-10 | $x^{10} + x^9 + x^5 + x^4 + x^2 + 1$ | ATM AAL |
| CRC-16 | $x^{16} + x^{12} + x^5 + 1$ | HDLC |
| CRC-32 | $x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$ | LANs |

**Analysis:** While analyzing your program, you are required to address the following points:

- How is this method different from 2D parity scheme that you have implemented previously?
- What are the limitations of this method of error detection?

**MARKS DISTRIBUTION**

| Component | Maximum Marks | Marks Obtained |
|---|---|---|
| Preparation of Document | 7 | |
| Results | 7 | |
| Viva | 6 | |
| **Total** | **20** | |

Submitted by: Satyajit Ghana

Register No: 17ETCS002159

1. Algorithm/Flowchart

**calc_crc(message, crc_poly)**

```
1. remainder = 0x0
2. for (every byte in message)
3.       remainder = byte << (crc_poly.length - 8)
4.       for i = 0 to 7
5.       if (TOP_BIT(remainder) == 1)
6.                   remainder = (remainder << 1) XOR crc_poly
7.             else remainder = remainder << 1
```

**check_crc(message, crc_poly)**

```
1. crc = calc_crc(message, crc_poly)
2. if any bit of crc is ON return false
3. else return true
```

2. Program

**main.c**

```
#include <iostream>
#include <bitset>
#include <vector>

#include "crc_lib.hpp"

/**
 * Implement :
 * CRC-8
 * CRC-10
 * CRC-16
 * CRC-32
 */

std::ostream& operator<<(std::ostream& out, std::vector<std::bitset<8>> message) {
    for (auto& byte : message) {
        out << byte << " ";
    }
    return out;
}

int main(int, char**) {

    std::cout << "Enter your message (in binary) : ";
    std::string input;
    std::getline(std::cin, input);

    // pad extra zero to make the number of bits a multiple of 8
    unsigned to_pad = input.size() % 8 != 0 ? (8 - input.size() % 8) : 0;
    for (int i = 0 ; i < to_pad ; i++) {
        input = "0"+input;
    }

    std::vector<std::bitset<8>> message;

    for (int i = 0 ; i+8 <= input.size() ; i += 8) {
        message.push_back(std::bitset<8>(input.substr(i, 8)));
    }

    std::cout << "MESSAGE : " << message << std::endl;

    CRC_OPTION crc_option;

retake_choice:
    std::cout <<
        "Select a CRC Polynomial : \n"
        "1.\tCRC8\n"
        "2.\tCRC10\n"
        "3.\tCRC16\n"
        "4.\tCRC32\n"
        "Your Choice : ";
    char choice;
    std::cin >> choice;

    switch(choice) {
        case '1':
            crc_option = CRC8;
            break;
        case '2':
            crc_option = CRC10;
            break;
        case '3':
            crc_option = CRC16;
            break;
        case '4':
            crc_option = CRC32;
            break;
        default:
            std::cout << "Wrong Choice !";
            goto retake_choice;
    }

    std::tuple<std::bitset<32>, unsigned> crc_calculated = CRC::calc_crc(message, crc_opt
ion);

    std::cout << "CRC : " << std::get<0>(crc_calculated).to_string().substr(32-
std::get<1>(crc_calculated)) << std::endl;

    std::cin.ignore();
    std::cout << "Enter the data received (with CRC padded at the end) : ";
    std::getline(std::cin, input);

    to_pad = input.size() % 8 != 0 ? (8 - input.size() % 8) : 0;
    for (int i = 0 ; i < to_pad ; i++) {
        input = "0"+input;
    }

    message.clear();
    for (int i = 0 ; i+8 <= input.size() ; i += 8) {
        message.push_back(std::bitset<8>(input.substr(i, 8)));
    }

    std::cout << "MESSAGE : " << message << std::endl;

    CRC::check_crc(message, crc_option) ? std::cout << "PASS" : std::cout << "FAIL";

    std::cout << std::endl;
}
```

## crc_lib.hpp

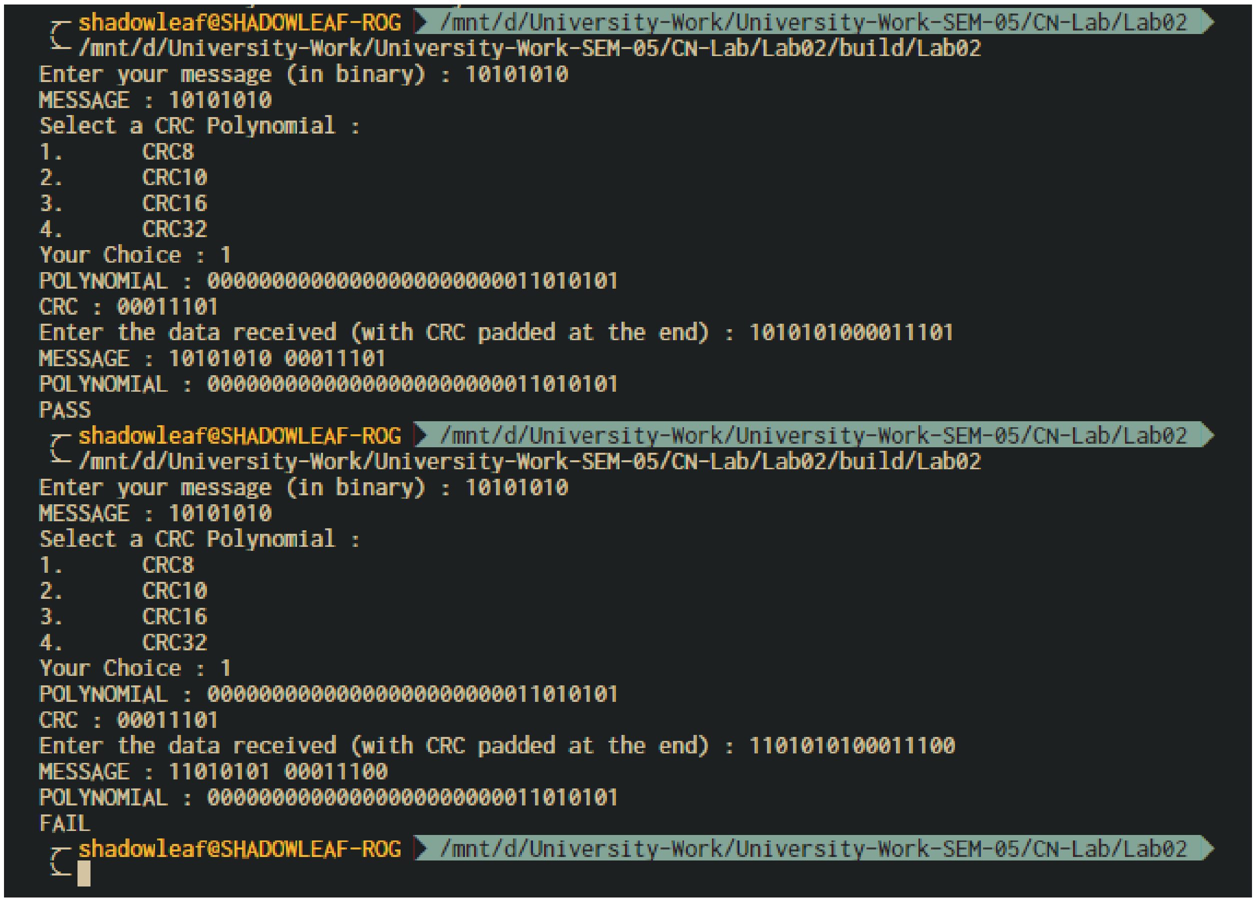
```
#pragma once

#include <bitset>
#include <vector>
#include <tuple>
```

```
enum CRC_OPTION {
    CRC8, CRC10, CRC16, CRC32
};

class CRC {
public:
    static std::tuple<std::bitset<32>, unsigned> calc_crc(std::vector<std::bitset<8>> &me
ssage, CRC_OPTION crc_option);
    static std::tuple<std::bitset<32>, unsigned> get_crc_polynomial(CRC_OPTION CRC_OPTION
);
    static bool check_crc(std::vector<std::bitset<8>>& message, CRC_OPTION crc_option);
};
```

**crc_lib.cpp**

```
#include "crc_lib.hpp"

#include <iostream>
#include <tuple>

// returns the CRC polynomial with the CRC polynomial length
std::tuple<std::bitset<32>, unsigned> CRC::get_crc_polynomial(CRC_OPTION crc_option) {

    std::bitset<32> crc_poly;
    unsigned crc_poly_len;

    switch(crc_option) {
        case CRC8:
            crc_poly = std::bitset<32>(0xD5);
            crc_poly_len = 8;
            break;
        case CRC10:
            crc_poly = std::bitset<32>(0x233);
            crc_poly_len = 10;
            break;
        case CRC16:
            crc_poly = std::bitset<32>(0x1021);
            crc_poly_len = 16;
            break;
        case CRC32:
            crc_poly = std::bitset<32>(0x04C11DB7);
            crc_poly_len = 32;
            break;
    }

    return std::make_tuple(crc_poly, crc_poly_len);
}

std::tuple<std::bitset<32>, unsigned> CRC::calc_crc(std::vector<std::bitset<8>>& message,
 CRC_OPTION crc_option) {
    // Fetch the CRC polynomial
    std::tuple<std::bitset<32>, unsigned> _crc_poly = CRC::get_crc_polynomial(crc_option)
;

    std::bitset<32> crc_poly = std::get<0>(_crc_poly);
    unsigned crc_poly_len = std::get<1>(_crc_poly);

    std::cout << "POLYNOMIAL : " << crc_poly << std::endl;
    // initialize the remainder with 0
    std::bitset<32> remainder(0x0);

    // Perform for every byte in message
    for (auto& byte : message) {
        // load the byte to the remainder
        remainder ^= ( (std::bitset<32>(byte.to_string())) << (crc_poly_len - byte.size()
) );

        // perform for every bit in the byte
        for (unsigned i = 0 ; i < 8 ; i++) {
            // std::cout << "REM : " << remainder << std::endl;
            // check if the top bit is 1
            if (remainder.test(crc_poly_len-1)) {
                remainder = (remainder << 1) xor crc_poly;
            } else {
                remainder = remainder << 1;
            }
        }
    }

    // std::cout << "CRC : " << remainder << std::endl;

    return std::make_tuple(remainder, crc_poly_len);
}

bool CRC::check_crc(std::vector<std::bitset<8>>& message, CRC_OPTION crc_option) {
    std::tuple<std::bitset<32>, unsigned> crc = CRC::calc_crc(message, crc_option);
    std::bitset<32>val(std::get<0>(crc).to_string().substr(32-std::get<1>(crc)));

    if (val.any()) {
        return false;
    } else {
        return true;
    }
}
```

3. Results

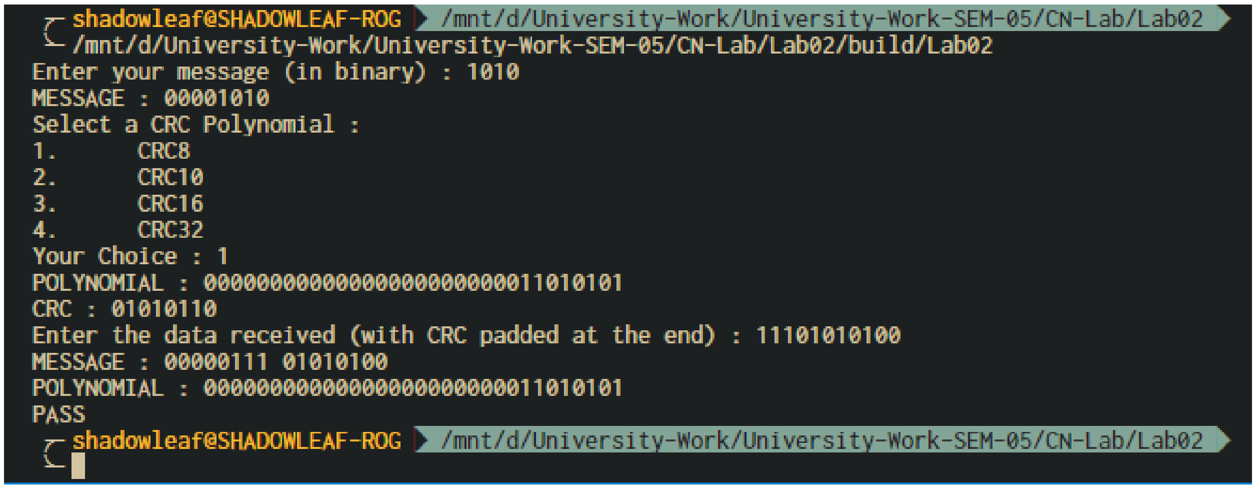


```
 ┌ shadowleaf@SHADOWLEAF-ROG  /mnt/d/University-Work/University-Work-SEM-05/CN-Lab/Lab02
 └ /mnt/d/University-Work/University-Work-SEM-05/CN-Lab/Lab02/build/Lab02
Enter your message (in binary) : 10101010
MESSAGE : 10101010
Select a CRC Polynomial :
1.      CRC8
2.      CRC10
3.      CRC16
4.      CRC32
Your Choice : 1
POLYNOMIAL : 00000000000000000000000111010101
CRC : 00011101
Enter the data received (with CRC padded at the end) : 1010101000011101
MESSAGE : 10101010 00011101
POLYNOMIAL : 00000000000000000000000111010101
PASS
 ┌ shadowleaf@SHADOWLEAF-ROG  /mnt/d/University-Work/University-Work-SEM-05/CN-Lab/Lab02
 └ /mnt/d/University-Work/University-Work-SEM-05/CN-Lab/Lab02/build/Lab02
Enter your message (in binary) : 10101010
MESSAGE : 10101010
Select a CRC Polynomial :
1.      CRC8
2.      CRC10
3.      CRC16
4.      CRC32
Your Choice : 1
POLYNOMIAL : 00000000000000000000000111010101
CRC : 00011101
Enter the data received (with CRC padded at the end) : 1101010100011100
MESSAGE : 11010101 00011100
POLYNOMIAL : 00000000000000000000000111010101
FAIL
 ┌ shadowleaf@SHADOWLEAF-ROG  /mnt/d/University-Work/University-Work-SEM-05/CN-Lab/Lab02
 └
```

*Figure 0-1 OUTPUT 1*

Explanation for Figure 0-1 OUTPUT 1:

Here the data bits are taken as 10101010, and the CRC chosen was CRC8 which is 111010101, performing manual CRC calculation resulted in the CRC to be 00011101, which is same as the output from the program, the message is then padded with the CRC at the end and the received message is given to the program which then verifies the output. Printing PASS if the message is error free, FAIL otherwise. Following the output, is the input for the same message but the message that is received is altered, here the error is detected since the remainder from the message is non-zero.



```
┌ shadowleaf@SHADOWLEAF-ROG  /mnt/d/University-Work/University-Work-SEM-05/CN-Lab/Lab02
└ /mnt/d/University-Work/University-Work-SEM-05/CN-Lab/Lab02/build/Lab02
Enter your message (in binary) : 1010
MESSAGE : 00001010
Select a CRC Polynomial :
1.      CRC8
2.      CRC10
3.      CRC16
4.      CRC32
Your Choice : 1
POLYNOMIAL : 00000000000000000000000011010101
CRC : 01010110
Enter the data received (with CRC padded at the end) : 11101010100
MESSAGE : 00000111 01010100
POLYNOMIAL : 00000000000000000000000011010101
PASS
┌ shadowleaf@SHADOWLEAF-ROG  /mnt/d/University-Work/University-Work-SEM-05/CN-Lab/Lab02
└ 
```

*Figure 0-2 OUTPUT 2*

Explanation for Figure 0-2 OUTPUT 2:

Here the data taken is 1010, and the respective CRC is calculated, assuming the data received by the receiver is mutated and now the data is a factor of the CRC polynomial, here the error in the data is not detected, which is one of the drawbacks of CRC.

4. Analysis and Discussions

The conventional method also generally utilizes cyclic redundancy check (CRC) bits for error detection purposes. In particular, a fixed number of CRC bits are appended to the end of each message block and have a predetermined relationship with the corresponding message block. The receiver receives both the message block and the CRC bits following that message block, and tries to re-establish the relationship therebetween. If the relationship is satisfied, the message block is considered without error. Otherwise, an error has occurred during the transmission of that block. This method is further explained in greater detail below.

First, a CRC generating polynomial, $g_l(x)$, of order 1, is chosen. A common way of choosing the CRC generating polynomial is that $g_l(x)$ should satisfy $gcd(g_l(x), x) = 1$ for each and every i between 0 and 1, inclusive, wherein 1 and i are integers, and the function $gcd(A(X). B(x))$ is defined as the greatest common divider of polynomials A(X) and B(x). Examples of suitable g(x) include $g_4(x) = x^4 + x^3 + x^2 + x + 1$ for l=4; $g_7(x) = x^7 + x^6 + x^4 + 1$ for l=7:$g_8(x) = x^8 + x^7 + x^4 + x^3 + x + 1$ for l=8; and $g(x) = x^{12} + x^{11} + x^3 + x^2 + x + 1$ for l=12. The information of CRC generating polynomial is stored in both the transmitter and the receiver. **(Shien S.L, 2007)**

Shien, S.L., Industrial Technology Research Institute, 2007. *Cyclic redundancy check modification for message length detection and error detection*. U.S. Patent 7,240,273.

2D parity implemented in the previous lab is prone to leave errors made in a square pattern undetected, a large part of the burst errors can be detected using this method, although still another huge segment of the burst errors go undetected, this is where the CRC comes in, the number of bits in CRC is fixed, i.e. the number of redundant bits are fixed $O(1)$, unlike where in 2D parity the number of bits increase $O(n)$ as the message length increases, in terms of space complexity CRC is better.

Albeit there are a few limitations associated with CRC, such as:

(i) The error in the data can do undetected if the message is mutated such that it is a factor of the CRC polynomial itself, as shown in Figure 0-2 OUTPUT 2.

(ii) Because a CRC is based on division, no polynomial can detect errors consisting of a string of zeroes prepended to the data, or of missing leading zeroes.

(iii) Single bit errors will be detected by any polynomial with at least two terms with non-zero coefficients.

(iv) Burst errors of length n will be detected by any polynomial of degree n or greater which has a non-zero $x^0$ term.

## 5. Conclusions

CRC is a much better upgrade to the 2D parity methods, with very low probability of errors left undetected by the receiver. It is sweet, short, although takes a little amount of computation, which can be fixed by using a lookup table, works great and is used practically in Networking.

## 6. Comments

### a. Limitations of the experiment

The implementation part of the program is inefficient, i.e. it calculates the CRC for every byte in the data, although this is not required, since a lookup table can be created that will solve the problem in $O(1)$ time for every bit, and $O(n)$ in total, where n is the number of bytes in the message.

### b. Limitations of the results obtained

The results are checked and the limitation of the CRC is shown, other than that and a few other limitations due to the maths behind CRC, no other limitations are known.

### c. Learning

Through this lab the concept of CRC was learnt that is used for error detection.

### d. Recommendations

Make the implementation more generic and more efficient, the program should work for any sort of data, files, strings, everything, this can be achieved by treating the data as a character array of bytes, then each byte can be processed individually.