COMPUTER NETWORKS LAB REPORT

Name: Abhishek De
Roll no: 001710501027
Class: BCSE-III
Section: A1
Assignment: 1

Problem statement:
Design and implement an error detection module which has four schemes namely LRC, VRC, Checksum and CRC. Please note that you may need to use these schemes separately for other applications (assignments). You can write the program in any language. The Sender program should accept the name of a test file (contains a sequence of 0,1) from the command line. Then it will prepare the data frame (decide the size of the frame) from the input. Based on the schemes, codeword will be prepared. Sender will send the codeword to the Receiver. Receiver will extract the data word from codeword and show if there is any error detected. Test the same program to produce a PASS/FAIL result for following cases.
(a) Error is detected by all four schemes. Use a suitable CRC polynomial.
(b) Error is detected by checksum but not by CRC.
(c) Error is detected by VRC but not by CRC.
[Note: Inject error in random positions in the input data frame. Write a separate method for that.]

Design
The program has 3 stages:
1. Encoding
2. Error generation
3. Decoding and error detection

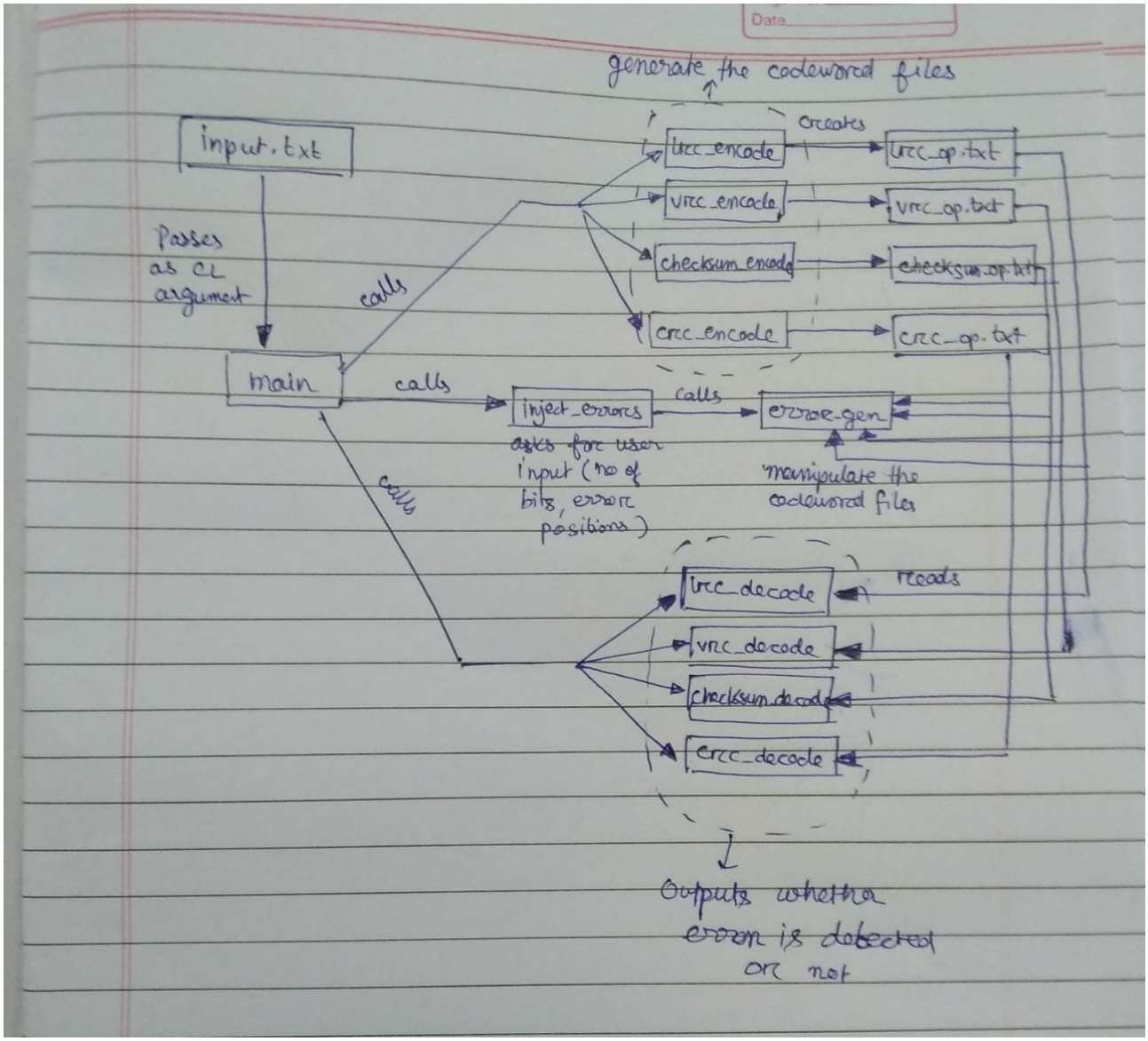
The name of the input file is taken as command line argument. The input file must contain a stream of bits(0s and 1s)

In the first stage, input file is read and split into frames of 8 bit. Then they are encoded using 4 different schemes(LRC, VRC, checksum, CRC) and stored in 4 different files namely `lrc_op.txt, vrc_op.txt,`

`checksum_op.txt`, and `crc_op.txt`. For CRC, generator used is `x^4+x+1`, which gives 3 redundant bits.

In the second stage, user input is taken for number of erroneus bits. Also, user is given choice to inject errors manually or using a random error generator. The first choice is preferred to get the various test cases.

In the third stage, the codewords are decoded using the 4 mentioned scheme and output is given as either "**Error detected**" or "**No error detected**".



Structure diagram of the program

Implementation

Main function

```
int main(int argc, char** argv)
{

    // Reading from input file and store as 8-bit frames
    bitset<8> b[100] = {0};
    int no_of_frames;
    read_frames(argv[1], b, &no_of_frames);

    // Calling functions for the 4 encoding schemes
    lrc_encode(no_of_frames, b);
    vrc_encode(no_of_frames, b);
    checksum_encode(no_of_frames, b);
    crc_encode(no_of_frames, b);

    // Calling error generating function
    inject_errors(no_of_frames);

    // Calling functions for the 4 decoding schemes
    lrc_decode();
    vrc_decode();
    checksum_decode();
    crc_decode();
}
```

LRC encoding function

```
// Function to implement LRC encoding scheme
void lrc_encode(int no_of_frames, bitset<8>* b)
{
    ofstream fout;
    fout.open("lrc_op.txt", ios::out);

    bitset<8> new_frame = 0;
    for(int i = 0; i < no_of_frames; i++)
    {
        fout << b[i];
        new_frame ^= b[i];
    }

    fout << new_frame;
    fout.close();
}
```

VRC decoding and error detection function

```
void vrc_decode()
{
    ifstream fin;
    fin.open("vrc_op.txt", ios::in);
    bitset<9> b[100] = {0};

    //Read frames from file
    int i = 0;
    fin >> b[i];
    while(!fin.eof())
    {
        i++;
        fin >> b[i];
    }
    int no_of_frames = i;
    fin.close();

    //Calculate syndrome
    bitset<1> syndrome = 0;
    for(i = 0; i < no_of_frames; i++)
    {
        syndrome = b[i].count() % 2;
        if(syndrome == 1)
        {
            cout << "VRC: Error detected" << endl;
            return;
        }
    }

    if(syndrome == 0)
        cout << "VRC: No error detected" << endl;
}
```

Error generation function

```
void error_generator(int k, int* error_pos)
{
    string files[] = {"lrc_op.txt", "vrc_op.txt", "checksum_op.txt", "crc_op.txt"};
    srand(time(NULL));
    for(int i = 0; i < 4; i++)
    {
        ifstream fin;
        fin.open(files[i], ios::in);
        string stream;
        fin >> stream;
        fin.close();

        for(int j = 0; j < k; j++)
        {
            int pos = error_pos[j];
            if(stream[pos] == '0') stream[pos] = '1';
            else stream[pos] = '0';
        }
        ofstream fout;
        fout.open(files[i], ios::out);
        fout << stream;
        fout.close();
    }
}
```

Division function for CRC codeword generation

```
bitset<12> division(bitset<12> msg)
{
    bitset<12> temp = msg;
    bitset<5> gen(string("10011"));
    for(int i = 11; i >= 4; i--)
    {

        if(temp[i] == 1)
        {
            for(int j = 4, k = i; j >= 0; j--, k--)
                temp[k] = temp[k] ^ gen[j];

        }
    }

    bitset<12> rem = temp & (bitset<12>)((1<<4)-1);
    return rem;

}
```

Test cases

Case 1: Error detected by all schemes

This occurs almost every time we generate the errors randomly.
An example output is shown below.

```
(base) abhishek@abhishek-HP-Laptop-15g-br0xx:~/Documents/Coursework/Sem_6/Comp
uter_networks/Assignments/Assignment_1_final$ ./a.out input.txt
Enter number of erroneous bits:
4
Inject errors:
1. Manually
2. Randomly
2
LRC: Error detected
VRC: Error detected
Checksum: Error detected
CRC: Error detected
```

Case 2: Error not detected by LRC

This occurs when, even number of bits belonging to same column/bit position are corrupted. In the example below, 4 bits, 2 each from 1st and 4th column are corrupted.

```
(base) abhishek@abhishek-HP-Laptop-15g-br0xx:~/Documents/Coursework/Sem_6/Comp
uter_networks/Assignments/Assignment_1_final$ ./a.out input.txt
Enter number of erroneous bits:
4
Inject errors:
1. Manually
2. Randomly
1
Enter error positions:0 3 8 11
LRC: No error detected
VRC: Error detected
Checksum: Error detected
CRC: Error detected
```

Case 3: Error not detected by VRC

This occurs when, even number of errors are injected in a frame so that parity remains unaffected. In the example below, 4 bits of the same frame are corrupted. Hence VRC cannot detect the error.

```
(base) abhishek@abhishek-HP-Laptop-15g-br0xx:~/Documents/Coursework/Sem_6/Comp
uter_networks/Assignments/Assignment_1_final$ ./a.out input.txt
Enter number of erroneous bits:
4
Inject errors:
1. Manually
2. Randomly
1
Enter error positions:0 1 2 3
LRC: Error detected
VRC: No error detected
Checksum: Error detected
CRC: Error detected
```

Case 4: Error not detected by checksum

This might occur if values of two frames get interchanged and hence checksum remains unaffected. In the example below, the value of 2nd and 4th frames get interchanged, hence checksum remains same.

```
(base) abhishek@abhishek-HP-Laptop-15g-br0xx:~/Documents/Coursework/Sem_6/Comput
er_networks/Assignments/Assignment_1_final$ ./a.out input.txt
Enter number of erroneous bits:
4
Inject errors:
1. Manually
2. Randomly
1
Enter error positions:8 12 24 28
LRC: No error detected
VRC: Error detected
Checksum: No error detected
CRC: Error detected
```

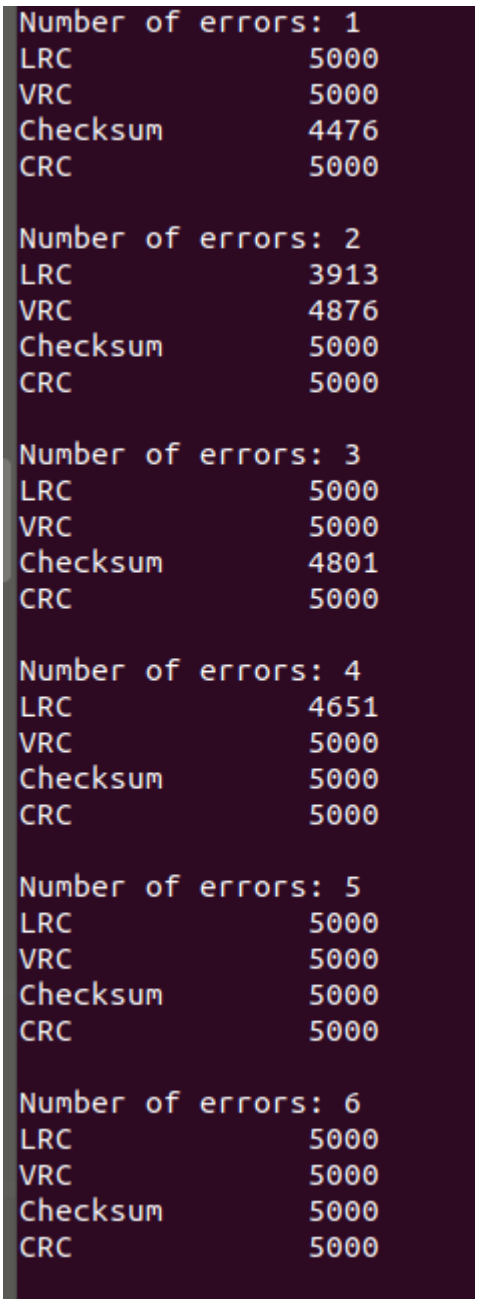
Case 5: Error not detected by CRC.

This occurs when, the error injected is divisible by the generator. In the example given below, generator is 10011 and error is of the form 10011000. Hence it cannot be detected by CRC.

```
(base) abhishek@abhishek-HP-Laptop-15g-br0xx:~/Documents/Coursework/Sem_6/Comp
uter_networks/Assignments/Assignment_1_final$ ./a.out input.txt
Enter number of erroneous bits:
3
Inject errors:
1. Manually
2. Randomly
1
Enter error positions:0 3 4
LRC: Error detected
VRC: Error detected
Checksum: Error detected
CRC: No error detected
```

Results

Using random error generation, the 4 error detection schemes were run 5000 times for error frequencies 1 to 5 bits and the following results were obtained.



Analysis:
From the above table it is clear that:

- LRC and VRC are not so efficient for even number of error bits.
- For 5 or more error bits, all schemes are equally efficient in error detection.
- Cyclic redundancy check is the most robust error detection scheme.

Comments
This lab experiment helped me in better understanding the error detection schemes that I had only learnt in theory and I enjoyed analysing and comparing the various schemes. The CRC algorithm implementation was

most challenging. Overall this was a great starting point and I am eagerly waiting for the next assignment.