NETWORK LAB REPORT

NAME : Dibyendu Mukhopadhyay
CLASS : BCSE-III
ROLL NO. : 001710501077
GROUP : A3
ASSIGNMENT NO. : 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 dataword 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 (list is given in next page).
(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.]

DEADLINE DATE : 25th January , 2019
SUBMISSION DATE : 27th January, 2019

DESIGN :

The purpose of the program is to detect whether the error is present in the codewords from the receiver's side . Error is caused by the noise or other impairments during transmission medium from the sender . There are basically 4 types of error detection schemes namely - LRC , VRC , Checksum and CRC .

During the programming way of design , there are mainly five files which handles the various aspects of the code . The following files are created to design in a chronological manner .

- sender.py - This file performs the role of sender which basically read the input_raw.txt (contains the data of 0 and 1) as a datawords ,then converting it into codewords with the help of suitable schemes and then transmit to the medium .

- transmission_medium.py - This file performs the role of sending the codeword to the receiver but the codeword is injected during that process of sending with the help of inject_error method .

- reciever.py - This file performs the duty of rejection and acceptance of codeword with the help of certain said schemes . It receives codewords from the transmission media .

- ErrorDetectionTechnique.py - The most important file and contains all the implementation of the error detection algorithms in different functions .

- user.py - This file is only file where user to do the task with the choice of cases to run the desired modules .

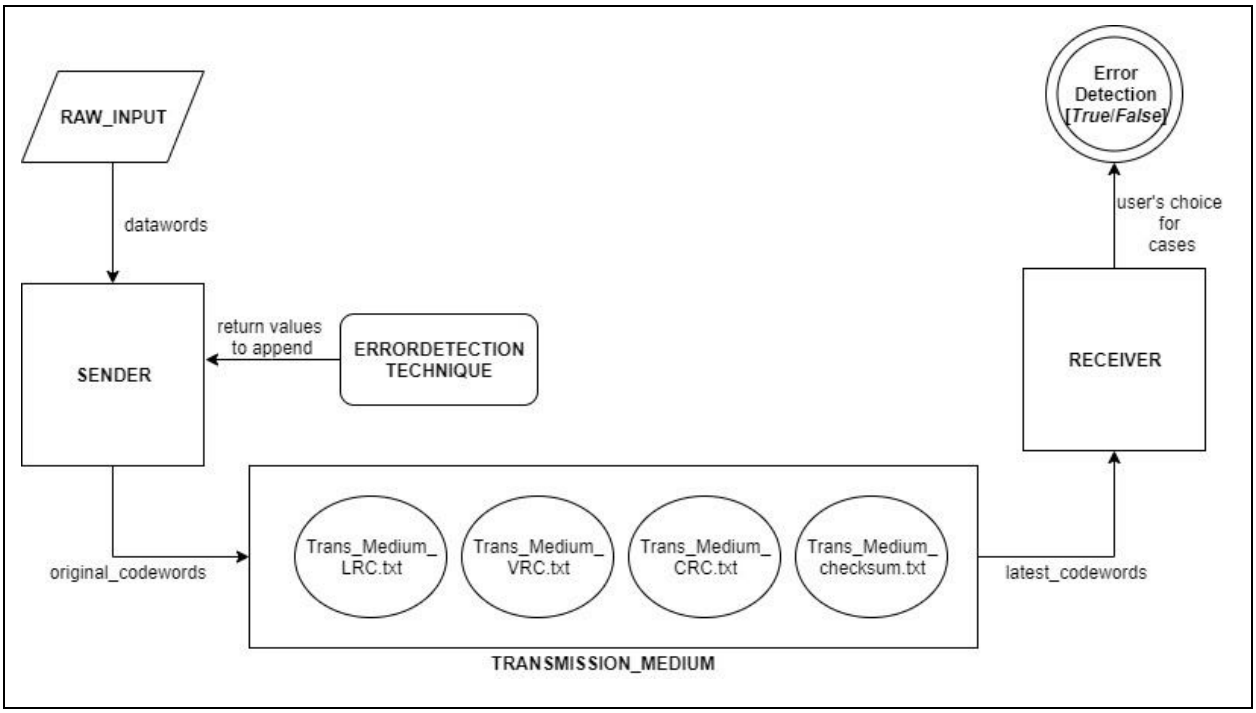


Fig.1. Schematic diagram of programming design

The diagram depicts the sender code reads the data from the input file, which is then converted into codewords by using the said algorithm and split into frames and then the transmission medium injects the error while sending the codewords and then writes the codewords to the respective files as shown. The receiver reads that file and splits into frames . Finally, the algorithm is applied to check whether there is an error or not in the output.

Input Format : The test file,i.e “raw_input.txt” which contains 48 characters in a sequence of 0 and 1.
Output Format : Checks whether the error is detected or not by the respective schemes.

IMPLEMENTATION :
In this problem statement , the entire program is implemented in Python3.
The detailed method description is written below with the suitable code snippets along with comments for better understanding code overview.

- ***Error Detection Technique***
  Here , all the 4 techniques are present
  1) This code snippet defines the CRC polynomial and frame size .

```
# CRC-7 -> x^7 + x^3 + 1
generator_poly='10001001'
no_of_bits=len(generator_poly) # n = 8
```

2) In this method, it takes a list of frames and frame size as a parameter.After that, it returns the LRC value of all frames within the list of frames.

```
# Function to generate the LRC code for a list of frames
def lrc(list_of_frames, no_of_bits):
    lsum=0

    for frame in list_of_frames:
        lsum=lsum^int(frame,2)
    lsum=bin(lsum)[2:]

    # Stuffing
    while(len(lsum)<no_of_bits):
        lsum='0'+lsum
    return lsum
```

3)In this method, it takes a list of frames as a parameter and a frame by adding one redundant bit at the end of every frame so that the total number of 1s in the unit becomes even,i.e., even parity .

```
# Returns codeword for each dataword using vrc
def vrc(list_of_frames):

    codewords=[]
    for i in range(len(list_of_frames)):
        # For every frame check if the parity in even or odd
        dataword=list_of_frames[i]
        if(dataword.count('1')%2==0):
            dataword+='0'
        else:
            dataword+='1'
        codewords.append(dataword)

    return codewords
```

4)In this method,it takes a list of frame and frame size as a parameter and returns the checksum of all frames . Finally the checksum is appended at the end of the last frame.

```
# Function to find checksum of a number of frames
def checksum(list_of_frames, no_of_bits):
    chksum=0

    for frame in list_of_frames:
        chksum=chksum+int(frame,2) # Computing the sum
    # Wrapping the sum
    csum=bin(chksum) # binary form
    csum=csum[2:] # Adding csum starting from index 2 (0 based indexing)

    while(len(csum)>no_of_bits):
        first=csum[0:len(csum)-no_of_bits]
        second=csum[len(csum)-no_of_bits:]
        s=int(first,2)+int(second,2)
        csum=bin(s)
        csum=csum[2:]

    # Perform 1s complement
    while(len(csum)<no_of_bits):
        csum='0'+csum
    chksum=''
    for i in range(len(csum)):
        if(csum[i]=='0'):
            chksum+='1'
        else:
            chksum+='0'

    return chksum
```

```
# Function of xor for two binary strings which is typecasted to integer
def xor(a,b):
    a=int(a,2) # returns the integer value which is equivalent to binary string
    b=int(b,2)
    a=a^b
    a=bin(a)[2:]
    return a
```

```
# Function to perform modulo 2 division
def modulo2div(dataword, generator):

    # Number of bits to be XORed a time
    l_xor=len(generator)
    tmp=dataword[0:l_xor]

    while (l_xor<len(dataword)):
        if(tmp[0]=='1'):
            # If leftmost bit is 1 simply xor and bring the next bit down
            tmp=xor(generator,tmp)+dataword[l_xor]
        else:
            # If leftmost bit is 0 then use all 0 divisor
            tmp=xor('0'*len(generator),tmp)+dataword[l_xor]
        tmp='0'*(len(generator)-len(tmp))+tmp

        l_xor+=1

    # For the last bit
    if(tmp[0]=='1'):
        tmp=xor(generator,tmp)
    else:
        tmp=xor('0'*len(generator),tmp)
    tmp='0'*(len(generator)-len(tmp)-1)+tmp
    checkword=tmp
    return checkword
```

5)In this method, it takes a list of frame and frame size as a parameter. In order to create the codeword for CRC, modulo 2 division function is there to help . This modulo 2 division function takes the dataword and the generator polynomial as an input parameter which performs the CRC division to return the codewords.

```
# Return the codeword for crc
def crc(list_of_frames, generator, no_of_bits):

    codewords=[]
    for i in range(len(list_of_frames)):
        # For every dataword perform crc division
        dataword=list_of_frames[i]
        # Append length of generator-1 bits to dataword
        aug_dataword=dataword+'0'*(len(generator)-1)

        # Now perform the modulo 2 division
        checkword=modulo2div(aug_dataword,generator)
        # Append the remainder
        codeword=dataword+checkword
        codewords.append(codeword)

    return codewords
```

- ***Sender***

This module is responsible for performing the role of sender as it creates the codewords from the datawords with the help of the error detection techniques.

1)This function reads the input file which is passed as argument , splits into frames and then returns a list of frames in order to perform the error detection.

```
# Function to read from the input file and convert it to a list of frames
def readfile(filename, no_of_bits):
    # Open the file in read mode
    f=open(filename,'r')
    data=f.read()

    # Now split the data into frames
    list_of_frames=[data[i:i+no_of_bits] for i in range(0, len(data), no_of_bits)]
    return list_of_frames
```

2)This function takes a list of frames and the frame size as a parameter and it creates LRC value with the help of error detection technique modules and it append at the end of the last frame and also prints the list of final frames which contains codewords.

```
# Function to write the lrc frames to file
def write_lrc(list_of_frames, no_of_bits):

    lrcval=err.lrc(list_of_frames=list_of_frames, no_of_bits=no_of_bits)

    list_of_frames2=list_of_frames[:]
    list_of_frames2.append(lrcval) #Append the resulted LRC at the last frame of datawords

    # Printing the frames
    print('Codeword frames sent:')
    print(list_of_frames2)

    with open('Original_sender.txt', 'w') as fw:
        fw.write("\n\nOriginal LRC ---\n")
        for item in list_of_frames2:

            fw.write("%s | " % item)
```

3)This function takes a list of frames and the frame size as a parameter and it creates VRC value with the help of error detection technique modules and it append at the end of every frame and also prints the list of final frames which contains codewords.

```
# Function to write the vrc frames to file
def write_vrc(list_of_frames, no_of_bits):

    list_of_frames2=err.vrc(list_of_frames=list_of_frames)[:]

    # Printing the frames
    print('Codeword frames sent:')
    print(list_of_frames2)

    with open('Original_sender.txt', 'a') as fw:
        fw.write("\n\nOriginal VRC ---\n")
        for item in list_of_frames2:

            fw.write("%s | " % item)
```

4)This method is responsible for writing the checksum with the help of error detection technique from the checksum algorithm and then finally append at the last frame. Then it created a codewords which is to send the medium transmission.

```
# Function to write the checksum frames to the file
def write_chksum(list_of_frames, no_of_bits):

    chksum=err.checksum(list_of_frames=list_of_frames, no_of_bits=no_of_bits)

    list_of_frames2=list_of_frames[:] # added all frames
    list_of_frames2.append(chksum) #put checksum at the end of last frame

    # Printing the frames
    print('Codeword frames sent:')
    print(list_of_frames2)

    with open('Original_sender.txt', 'a') as fw:
        fw.write("\n\nOriginal checksum ---\n")
        for item in list_of_frames2:

            fw.write("%s | " % item)
```

5) This method is responsible for creating the CRC value with the help of error detection module from the CRC algorithm.Then, codewords is created to send the medium.

```
# Function to write the crc frames to the file
def write_crc(list_of_frames, generator):

    list_of_frames2=err.crc(list_of_frames=list_of_frames, generator=err.generator_poly,
                            no_of_bits=err.no_of_bits)[:]

    # Printing the frames
    print('Codeword frames sent :')
    print(list_of_frames2)

    with open('Original_sender.txt', 'a') as fw:
        fw.write("\n\nOriginal CRC ---\n")
        for item in list_of_frames2:

            fw.write("%s | " % item)
```

6)It is a wrapper for calling the above methods to write and print the modules.

```
# Coverts dataword to codeword and wrote to the appropriate file
def dataword_to_codeword(list_of_frames, no_of_bits):

    print('\n\n++++++++++++++++++++++Sender++++++++++++++++++++++++++++++++\n\n')
    # Longitudinal Check Redundancy
    print('\nWriting to lrc file :')
    write_lrc(list_of_frames, no_of_bits)

    # Vertical Redundancy Check
    print('\n\nWriting to vrc file :')
    write_vrc(list_of_frames, no_of_bits)

    # Checksum
    print('\n\nWriting to checksum file :')
    write_chksum(list_of_frames, no_of_bits)

    # Circular Redundancy Check
    print('\n\nWriting to crc file :')
    write_crc(list_of_frames, no_of_bits)
```

- Transmission medium
  This module is responsible for injecting the error while transmitting the list of frames.

  1)This method is used to insert the errors.

```
# Function to introduce error as well as insert the error
def inject_error(list_of_frames, frame_no, list_of_bit):

    list_of_frames2=list_of_frames[:]
    frame=list_of_frames2[frame_no]
    new=list(frame)

    # Inserting error in the given bit position here or toggling problem
    for i in range(len(list_of_bit)):

        if(new[list_of_bit[i]]=='0'):
            new[list_of_bit[i]]='1'
        elif (new[list_of_bit[i]]=='1'):
            new[list_of_bit[i]]='0'
    list_of_frames2[frame_no]=''.join(new)

    return list_of_frames2
```

2)This method is to send the list of frames where the error was injected in the frame with the appropriate bit position with the help of error injection method.

```
def sending_codeword(list_of_frames, no_of_bits, error_list_frames, error_bit_list):
    global no_of_errors

    medium_lrc(list_of_frames, no_of_bits, error_list_frames, error_bit_list)

    medium_vrc(list_of_frames, no_of_bits, error_list_frames, error_bit_list)

    medium_chksum(list_of_frames, no_of_bits, error_list_frames, error_bit_list)

    medium_crc(list_of_frames, no_of_bits, error_list_frames, error_bit_list)
```

3)This method of LRC where the error is injected in the frame and the final codewords is sending to the receiver.

```
def medium_lrc(list_of_frames, no_of_bits, error_list_frames, error_bit_list):

    lrcval=err.lrc(list_of_frames=list_of_frames, no_of_bits=no_of_bits)

    list_of_frames2=list_of_frames[:]
    list_of_frames2.append(lrcval)

    # Inserting error
    for i in range(len(error_list_frames)):
        list_of_frames2=inject_error(list_of_frames2, error_list_frames[i], error_bit_list[i])

        #Reciever will read this txt file
    with open('Trans_Medium_LRC.txt', 'w') as fw:
        for item in list_of_frames2:
            item='0'*(len(err.generator_poly)-1)+item #for list maintaining order
                                                      #without this, bug arises

            fw.write("%s" % item)

    with open('Latest_reciver.txt', 'w') as fw:
        fw.write('\n\nReciever LRC ---\n')
        for i in list_of_frames2:

            fw.write("%s | " % i)
```

4)In this method which is almost similar to the above method where the list of frames is injected error with the help of error injection modules.

```
# Function to insert error toggling the vrc introduced frames to file
def medium_vrc(list_of_frames, no_of_bits, error_list_frames, error_bit_list):

    list_of_frames2=err.vrc(list_of_frames=list_of_frames)[:]

    # Inserting error
    for i in range(len(error_list_frames)):
        list_of_frames2=inject_error(list_of_frames2, error_list_frames[i], error_bit_list[i])

    with open('Trans_Medium_VRC.txt', 'w') as fw:
        for item in list_of_frames2:
            item='0'*(len(err.generator_poly)-2)+item
            fw.write("%s" % item)

    with open('Latest_reciver.txt', 'a') as fw:
        fw.write('\n\nReciever VRC ---\n')
        for i in list_of_frames2:

            fw.write("%s | " % i)
```

5)This function take list of frames, the generator polynomial, frame size, the positions where any error is to be introduced and the respective bit position for every frame where error is to be introduced.The checksum is calculated for the frames and the frames are written to the respective file. .

```
# Function to insert error toggling the checksum frames to file
def medium_chksum(list_of_frames, no_of_bits, error_list_frames, error_bit_list):

    chksum=err.checksum(list_of_frames=list_of_frames, no_of_bits=no_of_bits)

    list_of_frames2=list_of_frames[:]
    list_of_frames2.append(chksum)

    # Inserting error
    for i in range(len(error_list_frames)):
        list_of_frames2=inject_error(list_of_frames2, error_list_frames[i], error_bit_list[i])

    with open('Trans_Medium_Checksum.txt', 'w') as f:
        for item in list_of_frames2:
            item=item='0'*(len(err.generator_poly)-1)+item
            f.write("%s" % item)

    with open('Latest_reciver.txt', 'a') as fw:
        fw.write('\n\nReciever checksum ---\n')
        for i in list_of_frames2:

            fw.write("%s |" % i)
```

6)This function take list of frames, the generator polynomial, frame size, the positions where any error is to be introduced and the respective bit position for every frame where error is to be introduced.

```
# Function to insert error toggling the CRC frames to file
def medium_crc(list_of_frames, generator, error_list_frames, error_bit_list):

    list_of_frames2=err.crc(list_of_frames=list_of_frames,
        generator=err.generator_poly, no_of_bits=err.no_of_bits)[:]

    # Inserting error
    for i in range(len(error_list_frames)):
        list_of_frames2=inject_error(list_of_frames2, error_list_frames[i], error_bit_list[i])

    with open('Trans_Medium_CRC.txt', 'w') as f:
        for item in list_of_frames2:
            f.write("%s" % item)

    with open('Latest_reciver.txt', 'a') as fw:
        fw.write('\n\nReciever CRC ---\n')
        for i in list_of_frames2:

            fw.write("%s | " % i)
```

- ***Receiver***

  This module is responsible for performing the role of receiver and it checks the error present or not with the help of error detection technique module. The codewords which received from the transmission medium where the error may injects in the particular frames within the list of bits.

  1)In this method, it takes a list of frames and size of frames as an input parameter to detect the error with the help of error detection technique module. It returns true if no error is present and prints the list of frames , otherwise rejects the frame.

```
# Check for error by lrc
def check_lrc(list_of_frames, no_of_bits):

    # Removing padding
    list_of_frames=[list_of_frames[i][len(err.generator_poly)-1:]
     for i in range(len(list_of_frames))]

    lrcval=err.lrc(list_of_frames=list_of_frames, no_of_bits=no_of_bits)
    #if the appended value is zero
    if(int(lrcval,2)==0):
        print('No error in data detected by LRC')
        print('Dataword frames are')
        print(list_of_frames[0:-1])

    else:
        print('False!\n Error detected by LRC')
```

2)In this module, it takes a list of frames as a parameter and checks if there is an error present in the list of frames.The parity 1 of every frame is checked and if it is 0 then there is no error in that frame, prints the list of frame otherwise there is an error in that frame.It is done by said technique module.

.

```
# Check for error by vrc
def check_vrc(list_of_frames):

    # Removing padding
    list_of_frames=[list_of_frames[i][len(err.generator_poly)-2:]
     for i in range(len(list_of_frames))]
    flag=True

    for i in range(len(list_of_frames)):
        if(list_of_frames[i].count('1')%2!=0):
            print('False! Error detected in frame '+str(i+1)+' by VRC')
            flag=False

    if(flag):
        # No error extract dataword
        print("No error detected in data by VRC")
        list_of_frames=[list_of_frames[i][0:-1] for i in range(len(list_of_frames))]
        print('Dataword frames are')
        print(list_of_frames)
```

3)This function takes a list of frames and the number of bits as an input parameter. It is responsible to check whether the error is present in the list of frames and returns zero if

there is no error. This is calculated by the help of error detection technique module . If it returns false then it is rejected.

```
# Check for error by checksum
def check_checksum(list_of_frames, no_of_bits):

    # Removing padding
    list_of_frames=[list_of_frames[i][len(err.generator_poly)-1:]
    for i in range(len(list_of_frames))]

    chksum=err.checksum(list_of_frames=list_of_frames, no_of_bits=no_of_bits)

    if(int(chksum,2)==0):
        # In case of no error detected then dataword is printed
        print('No error in data is detected by checksum')
        print('Dataword frames are')
        print(list_of_frames[0:-1])

    else:
        print('False!\n Error detected by checksum')
```

4)This function takes a list of frames and the CRC polynomial passed as a parameter. For every frame, the **modulo2div()** method is applied and if the remainder is zero then there is no error otherwise there is an error in the frame.

```
# Check for error by crc
def check_crc(list_of_frames, generator):

    flag=True
    for i in range(len(list_of_frames)):
        if(int(err.modulo2div(list_of_frames[i],err.generator_poly),2)!=0):
            print('False ! Error detected in frame '+str(i+1)+' by CRC')
            flag=False

    if(flag):
        list_of_frames=[list_of_frames[i][0:err.no_of_bits]
        for i in range(len(list_of_frames))]
        print('Dataword frames are')
        print(list_of_frames)
        print("No error detected in data by CRC")
```

- ***User***

- This module combines all the above modules and gives the appropriate output with the help of user choice of cases .

```
#error list in frames and which position are manually done
def case1() :

    list_of_frames=(se.readfile('raw_input.txt',no_of_bits=err.no_of_bits))
    print('Case1: All 4 schemes can detect the error')
    se.dataword_to_codeword(list_of_frames, no_of_bits=err.no_of_bits)
    tm.sending_codeword(list_of_frames,no_of_bits=err.no_of_bits,
                        error_list_frames=[0, 1], error_bit_list=[[6], [4]])
    re.modules()
    print('--------------------------------------------------------------------------------')

def case2() :

    list_of_frames=(se.readfile('raw_input.txt',no_of_bits=err.no_of_bits))
    print('Case2: Error detected by checksum but not by CRC')
    se.dataword_to_codeword(list_of_frames, no_of_bits=err.no_of_bits)
    tm.sending_codeword(list_of_frames,no_of_bits=err.no_of_bits,
                        error_list_frames=[0], error_bit_list=[[0, 4, 7]])
    re.modules()
    print('--------------------------------------------------------------------------------')

def case3() :

    list_of_frames=(se.readfile('raw_input.txt',no_of_bits=err.no_of_bits))
    print('Case3: Error detected by VRC but not by CRC')
    se.dataword_to_codeword(list_of_frames, no_of_bits=err.no_of_bits)
    tm.sending_codeword(list_of_frames,no_of_bits=err.no_of_bits,
                        error_list_frames=[1], error_bit_list=[[0, 4, 7]])
    re.modules()
    print('--------------------------------------------------------------------------------')
```

TEST CASES :

For the 3 cases , errors have been manually inserted in order to get the required results . A text file named ***raw_input.txt*** consists of 48 characters with the sequence of 0/1. Obviously , the data created are exhaustive and thorough.

Sample Test Data : 100100011100111100111100101010111100111011010001

CRC Polynomial : CRC-7 : $x^7 + x^3 + 1$

Note that the frame size is based on the given CRC polynomial, therefore the given frame size is 8 as it has the same number of terms equal to CRC polynomial.

So , with the frame size of 8 bits , the given input file has a total of 6 frames .

CASE 1 : *Error is detected by all four schemes.*

During the transmission medium , error injects in frame 0 and 1 with the bit position of 6 and 4 respectively . Therefore, all schemes detects error .

Output :

```
Enter your choice : 1
Case1: All 4 schemes can detect the error
++++++++++++++++++++++Sender++++++++++++++++++++++++++++++++
Writing to lrc file :
Codeword frames sent:
['10010001', '11001111', '00111100', '10101011', '11001110', '11010001', '11010110']

Writing to vrc file :
Codeword frames sent:
['100100011', '110011110', '001111000', '101010111', '110011101', '110100010']

Writing to checksum file :
Codeword frames sent:
['10010001', '11001111', '00111100', '10101011', '11001110', '11010001', '00010110']

Writing to crc file :
Codeword frames sent :
['100100011010001', '110011111010010', '001111001000111', '101010110100000', '110011101011011', '110100010110101']
```

```
++++++++++++++++++++++Reciever++++++++++++++++++++++++++++++++

Reading file Trans_Medium_LRC.txt
Codeword frames received:
['000000010010011', '000000011000111', '000000000111100', '000000010101011', '000000011001110', '000000011010001', '000000011010110']
False!
 Error detected by LRC

Reading file Trans_Medium_VRC.txt
Codeword frames received:
['000000100100111', '000000110001110', '000000001111000', '000000101010111', '000000110011101', '000000110100010']
False! Error detected in frame 1 by VRC
False! Error detected in frame 2 by VRC

Reading file Trans_Medium_CRC.txt
Codeword frames received:
['100100111010001', '110001111010010', '001111001000111', '101010110100000', '110011101011011', '110100010110101']
False ! Error detected in frame 1 by CRC
False ! Error detected in frame 2 by CRC

Reading file Trans_Medium_Checksum.txt
Codeword frames received:
['000000010010011', '000000011000111', '000000000111100', '000000010101011', '000000011001110', '000000011010001', '000000000010110']
False!
 Error detected by checksum
```

CASE 2 : *Error is detected by checksum but not by CRC.*

During the transmission medium , error injects in frame 0 with the bit position of 0, 4 and 7 . Therefore, all schemes detects error except CRC as per case 2 .

Output :

```
Enter your choice : 2
Case2: Error detected by checksum but not by crc

++++++++++++++++++++++Sender++++++++++++++++++++++++++++++++

Writing to lrc file :
Codeword frames sent:
['10010001', '11001111', '00111100', '10101011', '11001110', '11010001', '11010110']

Writing to vrc file :
Codeword frames sent:
['100100011', '110011110', '001111000', '101010111', '110011101', '110100010']

Writing to checksum file :
Codeword frames sent:
['10010001', '11001111', '00111100', '10101011', '11001110', '11010001', '00010110']

Writing to crc file :
Codeword frames sent :
['100100011010001', '110011111010010', '001111001000111', '101010110100000', '110011101011011', '110100010110101']
```

```
++++++++++++++++++++++++Reciever++++++++++++++++++++++++++++++++++++

Reading file Trans_Medium_LRC.txt
Codeword frames received:
['000000000011000', '000000011001111', '000000000111100', '000000010101011', '000000011001110', '000000011010001', '000000011010110']
False!
 Error detected by LRC

Reading file Trans_Medium_VRC.txt
Codeword frames received:
['000000000110001', '000000110011110', '000000001111000', '000000101010111', '00
0000110011101', '000000110100010']
False! Error detected in frame 1 by VRC

Reading file Trans_Medium_CRC.txt
Codeword frames received:
['000110001010001', '110011111010010', '001111001000111', '101010110100000', '110011101011011', '110100010110101']
Dataword frames are
['00011000', '11001111', '00111100', '10101011', '11001110', '11010001']
No error detected in data by CRC

Reading file Trans_Medium_Checksum.txt
Codeword frames received:
['000000000011000', '000000011001111', '000000000111100', '000000010101011', '00
0000011001110', '000000011010001', '000000000010110']
False!
 Error detected by checksum
```

CASE 3 : *Error is detected by VRC but not by CRC.*

During the transmission medium , error injects in frame 1 with the bit position of 0, 4 and 7. Therefore, all schemes detects error except CRC as per case 3.

Output :

```
Enter your choice : 3
Case3: Error detected by VRC but not by CRC

++++++++++++++++++++++Sender++++++++++++++++++++++++++++++++

Writing to lrc file :
Codeword frames sent:
['10010001', '11001111', '00111100', '10101011', '11001110', '11010001', '11010110']

Writing to vrc file :
Codeword frames sent:
['100100011', '110011110', '001111000', '101010111', '110011101', '110100010']

Writing to checksum file :
Codeword frames sent:
['10010001', '11001111', '00111100', '10101011', '11001110', '11010001', '00010110']

Writing to crc file :
Codeword frames sent :
['100100011010001', '110011111010010', '001111001000111', '101010110100000', '110011101011011', '110100010110101']
```

```
++++++++++++++++++++++Reciever++++++++++++++++++++++++++++++++++++

Reading file Trans_Medium_LRC.txt
Codeword frames received:
['000000010010001', '000000001000110', '000000000111100', '000000010101011', '000000011001110', '000000011010001', '000000011010110']
False!
 Error detected by LRC

Reading file Trans_Medium_VRC.txt
Codeword frames received:
['000000100100011', '000000010001100', '000000001111000', '000000101010111', '00
0000110011101', '000000110100010']
False! Error detected in frame 2 by VRC

Reading file Trans_Medium_CRC.txt
Codeword frames received:
['100100011010001', '010001101010010', '001111001000111', '101010110100000', '11
0011101011011', '110100010110101']
Dataword frames are
['10010001', '01000110', '00111100', '10101011', '11001110', '11010001']
No error detected in data by CRC

Reading file Trans_Medium_Checksum.txt
Codeword frames received:
['000000010010001', '000000001000110', '000000000111100', '000000010101011', '000000011001110', '000000011010001', '000000000010110']
False!
 Error detected by checksum
```

RESULT :

Performance Metric :

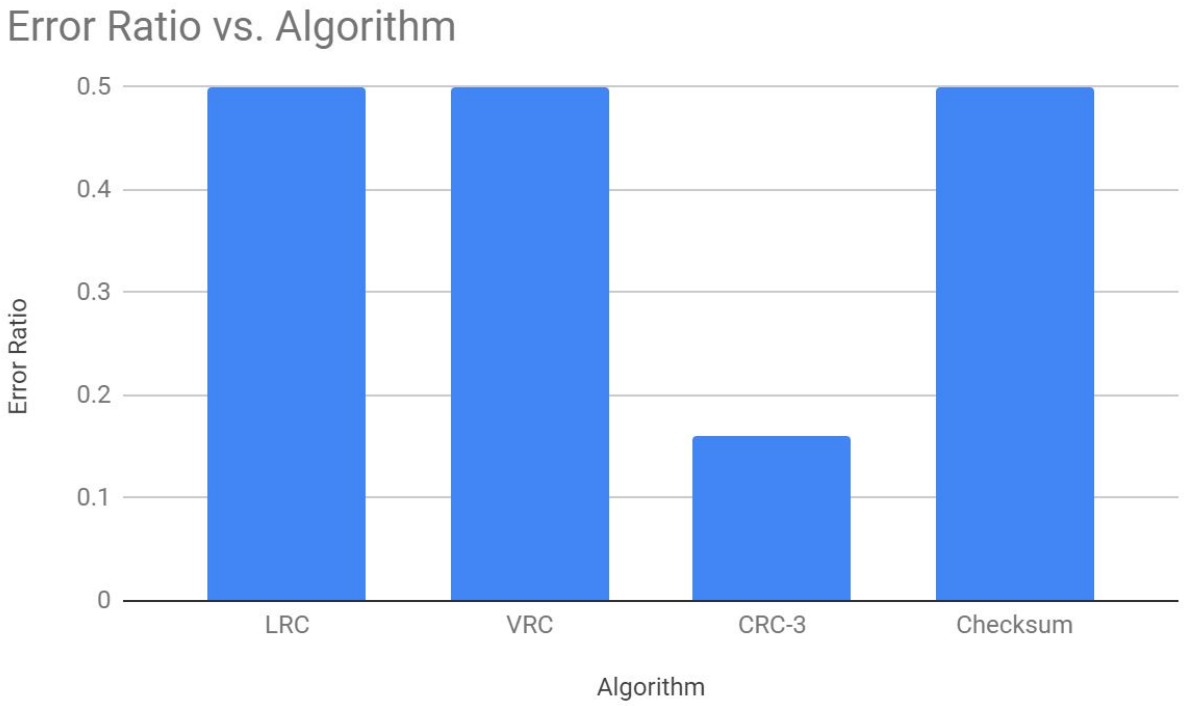
Error Ratio = $\frac{False\ Positive}{Frames\ Transmitted}$

Where , False Positive means the error is injected(i.e. The frame has changed) during transmission media but still accepted by scheme .

Evaluation Tables :

Algorithm	Error Ratio
LRC	0.5
VRC	0.5
CRC-3	0.16
Checksum	0.5

- On average of 10 independent executions have been taken
- Total number of frames for each scheme per excution is 60



ANALYSIS

- The implementation of the problem and test cases are more or less correct. As the above graph shown, the performance metrics of the methods is sturdy as how well the algorithm can detect errors. After evaluating the results, we can say CRC is more sturdier than most of the known schemes.

- The possible known bugs can arise due to the assumption that the input size is a multiple of frame size. However , it is overcome by padding the last frame of the input data with 0's so that it is a multiple of frame size .

- The possible improvements can be applied by :-
    - Specify the more error injection techniques.
    - Output format can be understandable.
    - Non-uniform formats make the test case to be understood by scheme more difficult and thus robustness decreases.

COMMENTS

The lab assignment was great to learn since we got to know the idea of how to implement the Error Detection Algorithms and how the process works while transmitting the codewords and how did the receiver detects the error. Overall, It was, of course, the great learning experience.

The assignment was moderately difficult since the idea of implementing the algorithm was bit of taking more time to handle the aspect of overview and how to overcome of test cases to get the desired output was thorough and exhaustive.

My suggestion for improving the program is to use the socket programming to implement the said schemes .